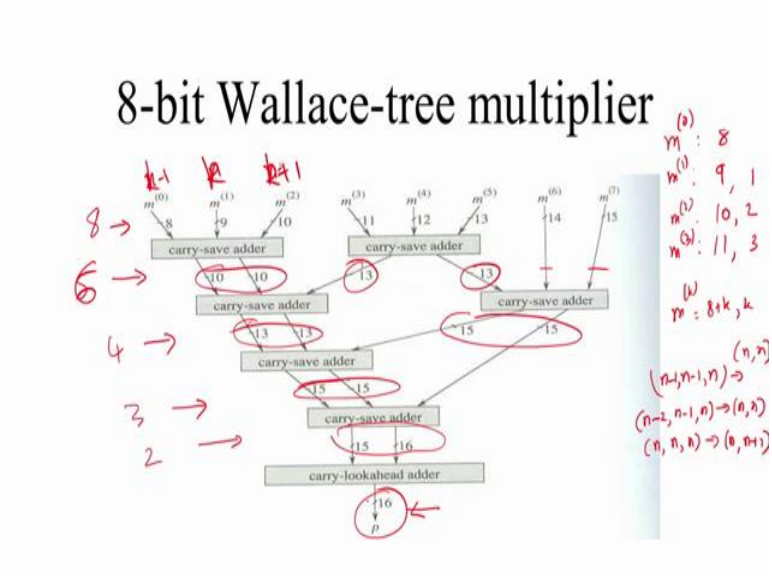
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 06
Fast Multiplier Circuit (Contd)

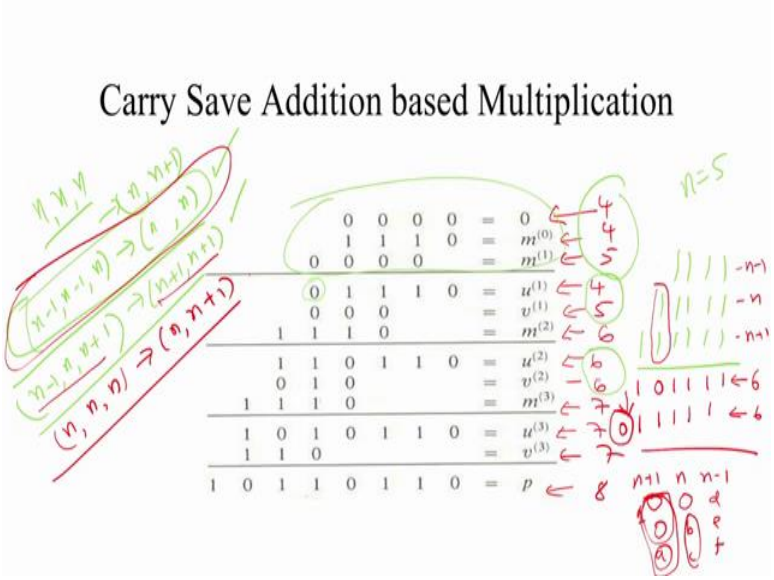
So, we start with where we left yesterday. So, we are doing the carry save multiplication, multiplication based on carry save addition and basically landed up with the tree basic.

(Refer Slide Time: 00:36)



This is the 16 bit 8 bit multipliers which manipulates to 8 bit numbers and basically gives you a 16 bit number. As you see this is a tree of carry save adders and some of the things that we learnt or that we saw yesterday some of the things that we saw some of the points that we noted yesterday was that.

(Refer Slide Time: 00:54)



If there are 2 n minus 1 bit and 1 n bit number the carries the result in 2 numbers that will come out of this 3 are n, n and n bits; n minus 1, 1 n plus 1 we get n plus 1 n plus 1 n n and n we get n and n plus 1. So, this is something that we need to keep in mind, and we use this particular stuff to show that when I multiply two 8 bit numbers we do get 16 bit number.

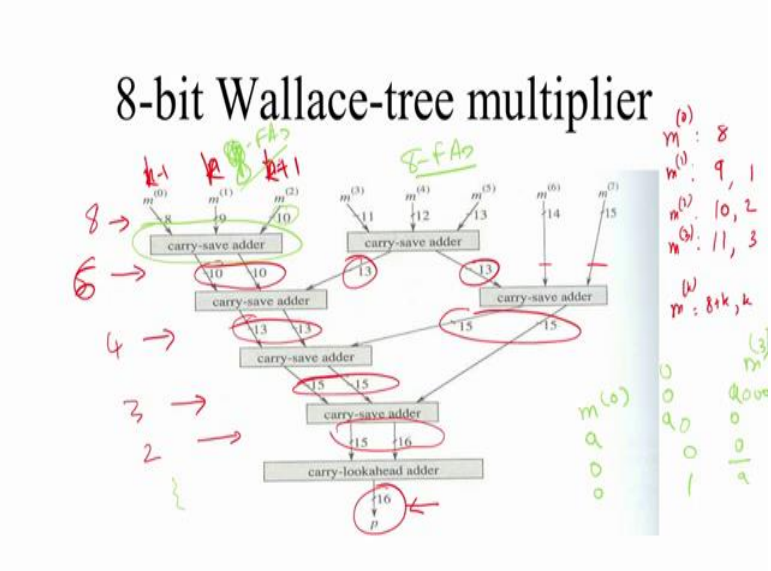
So, typically where when I add 5 9 and 10 I get 10 10 11 12 and 13 I get 13 and 13, then I add 13 14 and 15, I get 15 15 13 and 13, and 15 I get 15 and 15 and so on so forth. So, so when we add 3 15 bit numbers then we get a 15 and 16 bit number, and we add a 15 bit number to a 16 bit number we get a 16 bit answer.

So, this particular whatever we saw here was is extremely necessary for us for 2 reasons first to appreciate the point that to 8 bit numbers indeed gives us a 16 bit result, but then we can also do quite a bit of optimizations here right. So, let me just copy down those 3 here. So, we when I have n minus 1, n minus 1 and n this gives me n and n, the next one is that n minus 1, n plus 1 will give me n plus 1 n plus 1. So, I can just write it as n minus 2, n minus 1, n will also give me n right both being same and n and n will give me n comma n plus 1.

I think these are the 3 formulas that we need to keep in mind. So, with this these 3 were actually used to show the 16 bit answer, but now let us go and see how we can optimize

it a bit further ok. So, let us see how we can go and optimize this a bit I will use a different color good.

(Refer Slide Time: 03:30)



So, for this particular addition please note that we need. So, I have an 8 bit number, 9 bit number and a 10 bit number. So, I need basically let us say worse case I need 10 full adders for this carry save addition. So, carry save addition is a collection of full adders in parallel right.

So, in the worst case I need 10 carry 10 full adders in parallel, but actually the last number if you take the tenth bit here it is basically 0 0 and 1, the tenth input to the tenth bit are you realizing this right it will be 0 0 1. So, I may even I will be happy with just 9 full adders, where the last bit is anyway 1 and the carry bit is 0. So, I can hard code that. So, it is not that I require to 10 full adders here I could manage it with 9. So, I am just optimizing 1 here. Now here I am now I am taking a 11 bit number, 12 bit number, and 13 bit number.

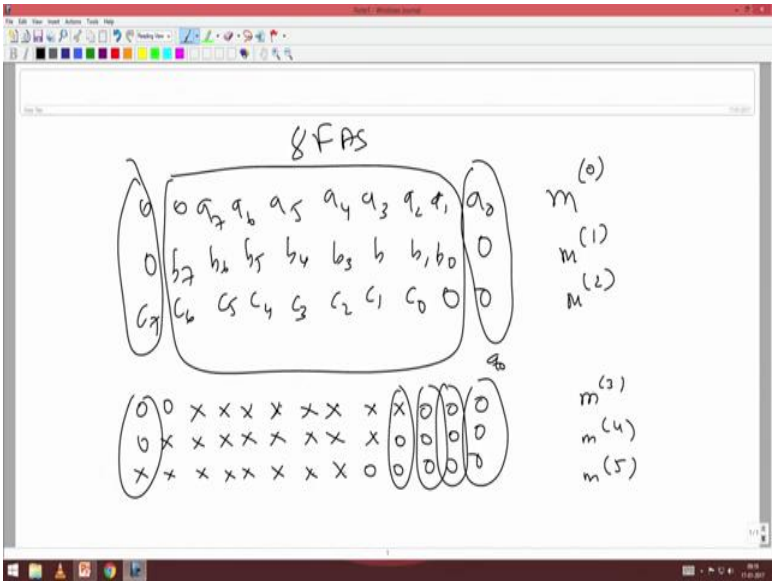
But you please note that the last 3 bits are zeros for all right; because this m 3 is an 8 bit number with 3 more zeros at the end, now it is a partial product right when you look at the partial product m 3 is a 8 bit number with 3 zeros at the end. M 4 is n again an 8 bit number, but with 4 zeros, m 5 is again an 8 bit number with 5 zeros right. So, I again require please note here that I could remove of it is not the that I need 13 full adders

here, it is enough if I have 10 full adders because the last 3 bits are 0, and also note that the first bit after that would have 1 0 0 or 0 0 0 whatever right.

The eleventh m 3 basically we will have you know the first bit after it has 3 zeros and then it will have some a here the remaining things are zeros. So, I just need to take that a right. So, I can cut 1 adder there. So, I can cut up to 4 adders at the front, and again at the last the first bit would be the most significant bit for the m 13 will have 0 0 and whatever a right. So, I need not have one adder at the last and one adder at the beginning. So, 3 plus 2 5 adders I can get. So, it is enough if I have just 8 full adders right; again here please note that even if I take that 8 bit number m 0, m 0 will be the first bit will be some a, and the for m 1 and m 2 this is 0. So, I need not have that last adder also.

So, I can manage this with 8 full adders in same argument. So, I do not need 10 full adders here nor do I need 13 full adders here, I can just manage with two 8 full adders right. So, let me again explain you in a little more whatever.

(Refer Slide Time: 06:46)



So, m 0 will be some a naught a 1, a 2, a 3, a 4, a 5, a 6, a 7; m 1 will be again let me say b naught b 1, b 2, b 3 b 4, b 5, b 6, b 7, m 2 will be. So, this is 0 c naught c 1, c 2, c 3, c 4, c 5, c 6 c 7. So, this is 0 and this is 0 rights. So, you do not need an adder for here right to generate.

So, you know you know the some bit is 0 and the carry is 0 all, and you do not need an full adder for this right. So, essentially you need only 8 8. So, again when I take m 0, m 1, m 2 when I take m 3, m 4 and m 5 the partial products, please note that there will be 3 zeros for m 3 again 3 zeros for m 4 and 3 zeros for m 5 already, and then you will have 7 number 1 2 3 4 5 sorry 5 5 6 7 8, and I will have 1 2 3 4 5 6 7 8, and I have 1 2 3 4 5 6 7 8.

So, I do not need an adder for this, for this, for this, for this and also for this right these are I do not need a full adder for that. So, I need a full adder only for 1 2 3 4 5 6 7 8 are you able to appreciate this right. So, let us go back to the slide right. So, 8 full adders are sufficient for this again 8 full adders for this right; and now what happens I get two 10 bit numbers again, and here I am getting a 13 bit number, but there are 3 zeros at the end right and this is a 10 bit number so there 3 zeros at the end. So, 10 10 and 13 again I you can actually see that if you just work it out.

I will just do it as a simple exercise you can finish off in 9 full adders right and so on. So, ultimately if you sum up the amount of full adders right you may you will get very close to n log n. So, we can prove that in a very. So, I just want you to work out for as 8 bit adder and just appreciate how this is going to figure out. So, those are circuit looks sort of big, but we can actually optimize it very carefully in the way we mix these numbers. now what is the depth of this circuit how will you prove that the depth of the circuit how you will calculate the depth of the circuit, suppose I am multiplying 2 n bit numbers what is the depth of the circuit?

Student: (Refer Time: 10:32).

How do you prove that any ideas, how will you go ahead and prove this circuit depth we have done so much of data structures come on cannot you just say what are the data structures we have done CSE-IIT madras.

Student: Trees.

Trees you have done yeah come on.

Student: (Refer Time: 11:04).

[FL] you do it in third standard now a days trees binary trees, binary search trees (Refer Time: 11:14) heaps nova he says yes you say no what are the data structure can somebody tell Kavya can you tell me what are errors names of the data structures you say.

Student: Min heaps max heap.

Ah.

Student: (Refer Time: 11:25) heap.

Ah then?

Student: (Refer Time: 11:26).

What are type of heaps there are many type of heap mini heap max heap nah other than that.

Student: (Refer Time: 11:43).

Either you do it specific or not is not too specific is not half answer is binary 0 or 1 only heap then spanning trees.

Student: (Refer Time: 12:01).

One semester you just did only till heaps.

Student: Graphs (Refer Time: 12:04).

What graphs b f s d f s ah.

Student: ah.

Ah means what I am asking you ok.

Student: (Refer Time: 12:17).

Right. So, you did complexity analysis of this right, for search trees and all. So, would you do automatic balancing auto balancing trees like a red black tree, a v l trees, red black trees you have done so much, come on show this man this trivial thing come on;

hello Palakkad what are things what are data structures you have learnt list of data structures that you learnt black shirt first row.

Student: (Refer Time: 12:48).

Yeah, correct, yourself yeah.

Student: (Refer Time: 12:53) binary trees. (Refer Time: 12:59).

Then this is 1 week or 2 weeks max then after that, what is this man all of you have short term memory loss or what

Student: (Refer Time: 13:16).

The yellow shirt behind.

Student: (Refer Time: 13:21) yeah yes exactly.

Student: That much only (Refer Time: 13:26).

That much only you did?

Student: (Refer Time: 13:31) yes sir (Refer Time: 13:31).

You are doing now oh data structure is in this semester ok sorry good. So, correct 2 weeks you have done 3 things yeah very good. So, anybody, you all did computational engineering right.

Student: (Refer Time: 13:47).

You did binary search yes or no.

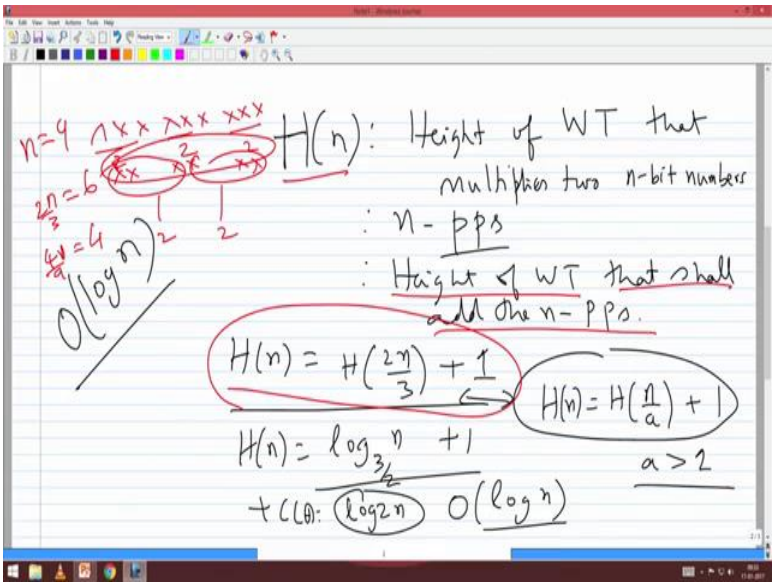
Student: Yes (Refer Time: 13:54).

Yes. So, how do you analyze the complexity of a binary search, suppose I am given an array of n elements what is the time to query whether an a element whether a given query element exist or not in the that array what is the time required.

Student: (Refer Time: 14:10).

Order log n now how do you prove that let me just see.

(Refer Slide Time: 14:14)

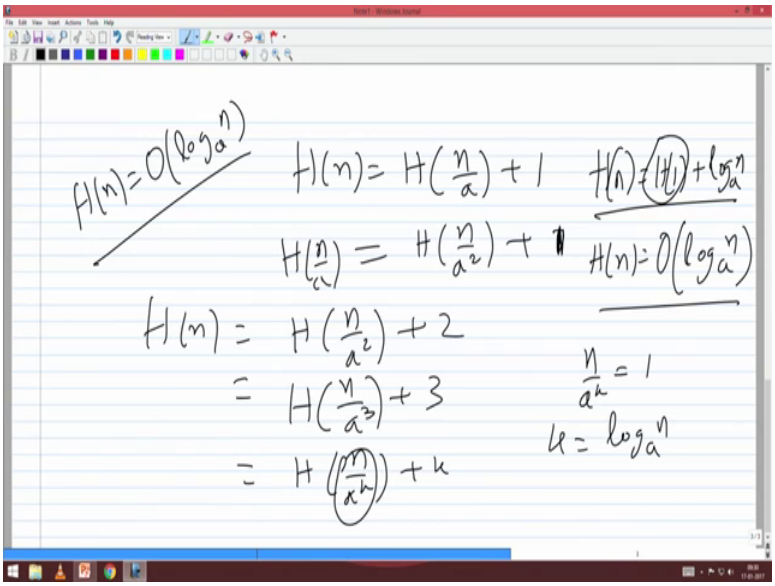


Now let; so I will not use H, I will use H; let what happens is initially you start with n numbers after the first step what happens? You want to add n numbers, so you start with let H n be the height of the tree, let H n denote the height of a tree which multiplies 2 n bit numbers. So, H n is height of Wallace tree I will say W T that adds that multiplies 2 n bit numbers; now what happens after one step this n. So, how many partial products you will have. So, essentially you have n partial products to start with.

So, essentially H n, let H n be the height of w tree that adds n partial products both are same right. So, let H n also denote the height of the Wallace tree that shall add the n partial products or the n numbers right. Now what happens after the first step? It becomes 2 n by 3 and you are going to recursively do the same thing for that right that is what I we have seen right. So, what did we see here? So, we took the first step and we did whatever we did for the first step the same thing we are continuing for the second step. So, these are recursive construction. So, H n will now become H of 2 n by 3, plus you have added 1 level. So, H n will essentially become H of 2 n by 3 plus 1.

So, this falls into the equation of what we call as H n is equal to H of n by a plus 1, a greater than 1 correct. So, how do you solve H of how do I solve this?

(Refer Slide Time: 16:51)



H of n is equal to H of n by a plus 1, H of n by a is equal to H of n by a square plus one. So, H of n essentially becomes H of n by a square plus 1 plus 1 so plus 2. So, like this we can say that it is H of n by a cube plus 3. So, H of n by a power k plus k; when this will stop when n by a power k equal to 1 or k is equal to log n to the base a. So, H of n is H of 1 plus log n to the base a correct, n by a power k becomes 1 k becomes log n to the base a. So, H of n is equal to H of 1 H of 1 will be a constant. So, H of n is I can go and say it is order log n to the base a.

Can you do that? Now let us go back to the previous slide. So, we have a is 3 by 2.

Student: (Refer Time: 18:13).

Yeah, sorry.

Student: Then how did you do (Refer Time: 18:18) how do we get 2 by 2.

Ah.

Student: 2 by 3 instead of (Refer Time: 18:26).

So, this you understood this whatever this slide you understood? Do you did you follow that H of n is equal to H of n by a plus 1 that equation the answer for H of n is order log n to the base a, did you understand this yes or no.

Student: (Refer Time: 18:52) yes sir.

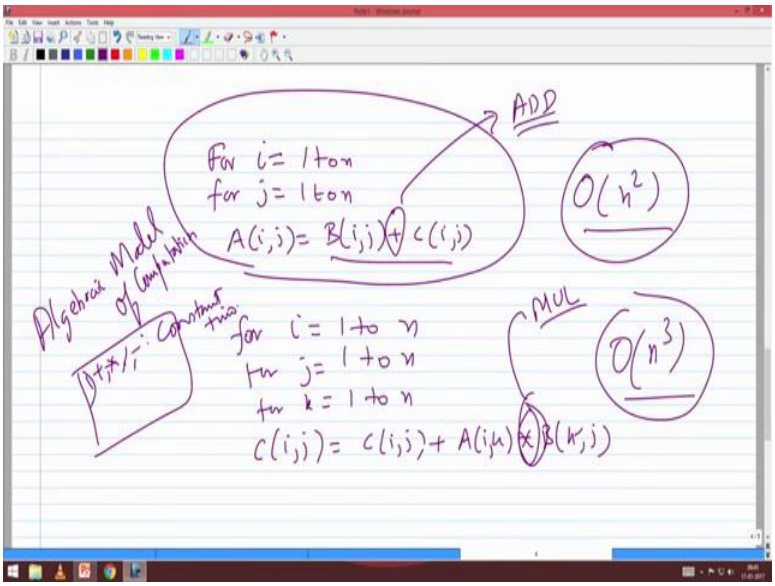
Yes. So, I will come to your question 1 minute in a minute, now when I substitute a is equal to 3 by 2 right this equation both these equations are same. So, what is the answer for this? H of n is log n to the base 3 by 2 plus 1, which is again order log n logarithmic in; n now 1 minute. So, I will come to your question very shortly after bringing it to some bringing it to 2 numbers right, then what we do we again do a carry look ahead addition. Now what is the complexity of carry look ahead addition to I have again say to I have 2 n bit numbers right 2 2 n bit numbers it is again logarithmic in n.

So, the carry look ahead addition also takes this plus CLA which also takes again log n log 2 n is again log n order log n time right, log 2 n is log 2 plus log n 1 plus log n. So, basically takes this much amount of time. So, the total complexity of multiplication is also order log n right; it is all some constant times log n are you able to follow right. Now coming to his question how did I get this equation right; I start so H n is the height of the Wallace tree that shall add the n partial products, in one step right I reduce the number of. So, it is actually n numbers right I am trying to add n numbers n each partial product is a number I am adding n numbers, in one step in the first step we have n numbers.

In the second step what happens I have 2 n by 3 numbers because every 3 numbers became 2. So, if I have 9 numbers for example, then this essentially these 3 become 2, these 3 become 2, this three. So, I will have 6 numbers here, from 9 it becomes 6 right do you understand this? And so I will have 2 numbers here, 2 numbers here now I can club these 3 and become these 3 can become 2. So, from 9 it became 6 from 6 it became 4. So, n is 9, 2 n by 3 is 6 right 4 n by 9 is 4 right. So, every time I will get two thirds of the number that is there in the previous level because every 3 numbers now every triplet is converted into a pair.

You understand this and that is how I get H n is equal to H 2 n by 3 plus 1; are you able to follow good; teacher feels extremely happy if the student understands thank you now is all clear are you able to follow.

(Refer Slide Time: 22:25)



Now, let us look at some very interesting. So, why should I. So, what is. So, we say how in an architectural course we just say that no this is how Wallace tree is constructed this is how you know addition is done, but we took some effort to basically tell you this takes log n time. So, by showing that carry look ahead adder is also log n order log n and multiplication and addition both can be done in log n time, what point did I drive you what is the point that I am trying to drive here?

Student: Along you like (Refer Time: 23:06) all are computer science courses (Refer Time: 23:07) addition and multiplication (Refer Time: 23:09).

Exactly. So, where did you make this as? So, in many of the courses including data structures, suppose I say what is this algorithm and what is this time complexity.

Student: (Refer Time: 23:48).

What is it tell fully what is this algorithm; what is this, what does this piece of code do.

Student: (Refer Time: 24:00). Matrix (Refer Time: 24:02).

Matrix additions. So, what is the time complexity?

Student: Order n square.

Order n square what does this do.

Student: (Refer Time: 24:35).

What are the time complexity?

Student: (Refer Time: 24:36).

Now, what assumption did we make here that this multiplication is also constant time this addition is also constant time. So, we made an assumption here when we are analyzing the algorithm that multiplication and addition takes almost the same time right. So, when we are analyzing the complexity of algorithms, we do have something called a model of computation right; we do have something called a model of computation and we analyze the complexity based on the model of computation correct right. So, one of the model of computation is called algebraic model of computation.

This algebraic model of computation basically assumes that the several things that all this plus star, division, you know subtraction all takes constant time, accessing memory takes constant time like that it make certain assumptions. And when you analyze the algorithm you are analyzing it we assuming and underlying model of computation, if you say order n square order n squared of what; order n square of order n square time what me what do we mean by time there n square time means what time, time who gives you the time, that is the model of the computation which says that these are all the operations that will take unit time and if I analyze using this model of computation then my algorithm is n squared right; if I use another model of computation may algorithm may take some other time ok.

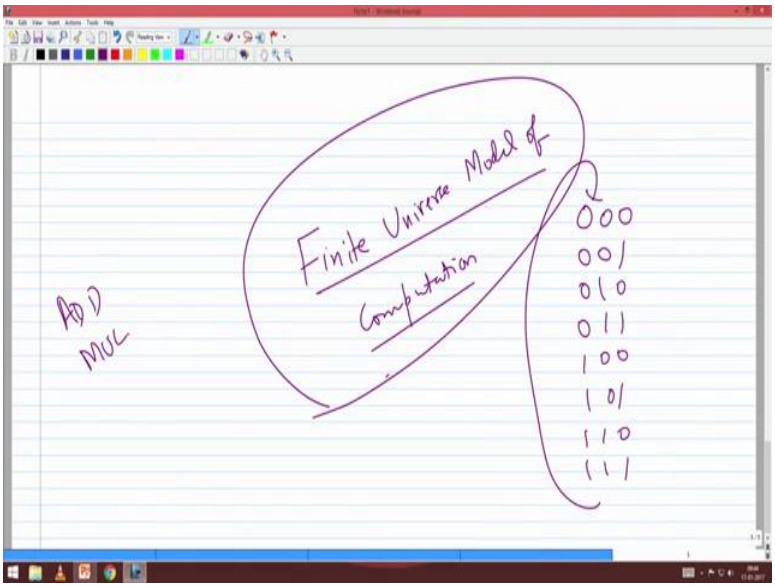
So, now given that addition and multiplication are taking the same amount of time almost the same amount of time, we are now justify it in telling that in actually stating that this is order n square and this is order n cube able to follow correct right. Now there is some comparison that if I actually go and implement in this in a real world machine, which basically has a carry look ahead adder and a you know Wallace tree multiplier then certainly we will get a very good the actual timing that you measure will be you know proportional to this n square and n cube; able to follow now this is the relation between architecture and you know actually relation between theory and practice.

So, we assume we make certain assumptions in theory and that is based on certain practical implementations right. Now algebraic model of computation is valid provided I

use these type of arithmetic circuits internet, if I do not use this type of arithmetic circuit if I use a touring machine to add 2 numbers then my arithmetic algebraic model of computation is no more valid are. You able to see that thing where theory meets practice then this is very very interesting and important example for all computer science and electrical engineers to understand right.

Now also keep in mind so I want you to go and explore a bit I am going to give you some extra reading. So, I will not ask any exam I do not plan to ask any questions in the quiz.

(Refer Slide Time: 28:11)



But I want you to go and look at this finite universe model of computation. Please understand that the arithmetic that we do the type of you know computations that we do on systems it is not infinite arithmetic its finite arithmetic correct. If I have a 32 bit architecture the 2 numbers that I could add is maximum bounded by 32 bits. So, if I go and add something more than 32 bits, I cannot represent anything a number which is which will take 33 bits here right.

So, if I go and add more than 32 bit then it becomes what 0 right. So, because the bits above 32 will be lost, the system does not have the width to hold anything beyond 32 bits. So, if I am going to handle numbers which cannot be represented in 32 bits, I will end up with a overflow. So, the system will tell I have a over flow, but then the result

will not be there it will. So, if you keep adding 0 1. So, let me say it is a 3 bit system. So, 0 0 0 0 0 1 add one add one more add one more add one more.

Now, you add 1 more you will go back to 0 0 that 1 will be lost, but what the architecture will tell you hey there was a overflow, but then the result is lost forever. So, this is safe. So, the systems on which we are working are basically finite or finite arithmetic systems finite addressable system. So, the memory if we have 32 bits I can address maximum 4 g b of ram, so it is not an a infinite system right. So, something which is very close to our computers today is this finite inverse model of computation. So, people have started working on this in 1980s right late 80s. So, you can see lot of papers and literature on finite universe model of computation just Google it and you will certainly get a couple of wiki links, just to have an understanding of what we mean by finite universe model of computation ok.

Now, I have a model of computation that is realized by the hardware, now how do how does this manifest itself in the program. It manifest itself in the program through instructions machine instructions. So, there will be a machine instruction called add there will be a machine instruction called mul. So, you write a program in you know in the c language like this this basically gets translated to some add it is a machine language, this basically gets translated to this mul. So, this program essentially becomes zeros and ones and some where this plus will become add, and this add is given to the architecture and that will basically use the carry look ahead adder, this mul goes to the system and that will basically use the Wallace tree multiplier right.

And since we use these pass circuits this addition will take almost the same time as multiplication or vice versa. So, this is how the entire system works. So, this is very obvious, but you need to understand this obvious fact also fair enough right. So, the last thing that I want to convey today afternoon we have a lab, in the lab we will talk about you know basically how C programs can get translated in to assembly code right. So, we will just basically take some programs and see how we can translate them to an assembly code I will give you some examples of this.

And we will also post the Intel manual this is close to thousand pages I do not want you to read those 1000 pages obviously, but then you can use it for quick referencing of there are 750 odd instructions. So, you can quickly refer to some instructions, so that you can

start your first programming exercise next week. So, today we will give you some translation from a c code to normal code. So, what we will be doing next I started asking some questions in twos compliment, I hope you all remember twos compliment yes or no. So, we will we will just very quickly go through twos compliment in tomorrow's morning class and then we will start you know division or floating point representation 1 of this I will do.

Floating point representation is extremely important and its specifically the I triple E 7 5 4 standard; because that is the actually building bridge between engineering computation and a computer science or a electrical and for e c or w e at Palakkad you will be doing lot of MATLAB work you will be doing lot of distance signal processing, fast Fourier transform these are all going to be brighten better for you in the coming semester or probably you are done in something in this direction earlier and all of these are numerical computations.

So, it is very very important that you understand how these numbers are represented within the computer and there could be truncation errors. So, we will be showing you lot of errors that could come in this entire business right. So, if you are not careful about handling floating point that to because it is a finite arithmetic again. So, we have limitations in number representation then you will land up with so many stupid results right, and you will also assume you if you do not if you are not careful in actually doing the computation modeling the computation.

We will show how you may get answers which are significantly erroneous is not that some 1 into 10 power minus 9 is the error, it will be significantly erroneous right and that error is sufficient enough for example, a rocket takes up that and there you have written your program and solve porting point and something goes wrong there that is enough to actually bend down the rocket and bring it back to you. So, assume becomes (Refer Time: 35:11) right. So, actually a rocket launch failure was due to floating points called a billion dollar bug. So, because one rocket went up and it fell down the reason was a floating point error and they lost one billion dollar in designing that rocket. So, that is called a billion dollar billion dollar or billion dollar bug. So, it is costly bug. So, we will discuss about floating point and you should understand that completely and thoroughly.

Thank you.