

Computer Architecture
Prof. Smruti Ranjan Sarangi
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Lecture - 19
Computer Arithmetic Part-III

Welcome back.

(Refer Slide Time: 00:26)

$O(\log(n)^2)$ Multiplier

- * Consider an n bit multiplier and multiplicand
- * Let us create n partial sums

$$\begin{array}{r} \times 1001 \text{ (m)} \\ 1101 \text{ (M)} \\ \hline 1001 \\ 0000 \\ 100100 \\ 1001000 \\ \hline \end{array}$$

partial sums

2^{n-1} $n-1$

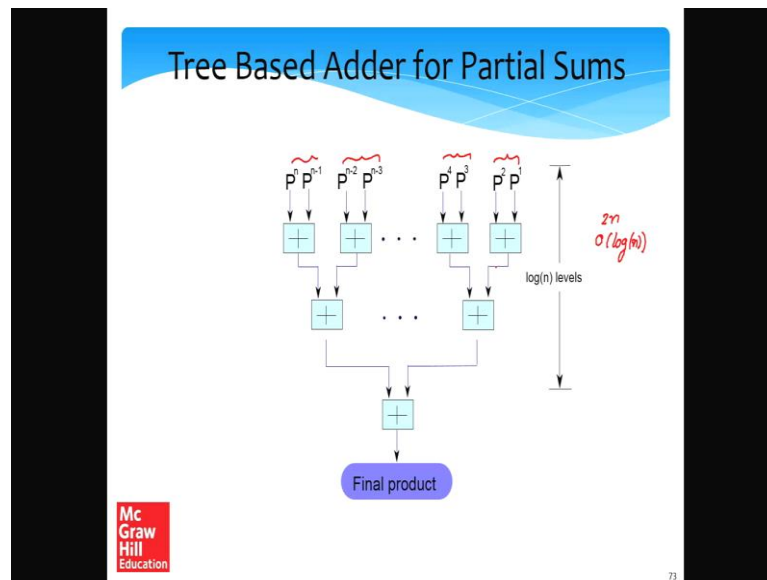
Mc
Graw
Hill
Education

72

Let us take A look at faster multipliers namely; an order $\log n$ square multiplier. So, this is fairly simple to construct. So, let us consider an n bit multiplier and n bit multiplicand. So, if you write the multiplier and the multiplicand, multiply here and the multiplicand here. So, what we can do is, we can construct n , partial sums, where reconstructing is partial sum is very simple and also it can be done in parallel. So, whenever we have a 1, we write the multiplicand as is if you have 0, we just writes 0s, and every partial sum is offset by one position to the left, as compared to the partial sum before it. So, if you are multiplying n bits, we will have n partial sums, small n mind you.

So all and all of these partial sums can be generated in parallel. So, this partial sum can be generated simultaneously with this, which can be generated simultaneously with this. So, generating all the partial since is very easy, it takes a constant amount of time. So, after that we need to add n numbers. So, for adding n numbers what we can do is, we can have a tree of adders.

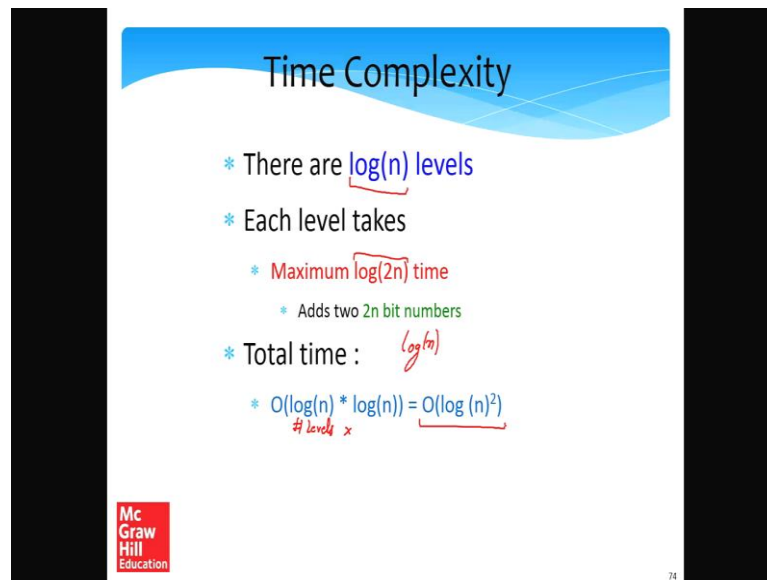
(Refer Slide Time: 01:45)



So, we can add the first and second here, we can add the third and fourth numbers here, the n minus 2th and then the n minus 3th numbers here. Similarly we can have a tree of adders. So, if we take a look at the sizes of the numbers; the first partial sum is n bits long, and the last partial sum, is $2n$ minus 1 bits long. So, this is n over here, and n minus 1 over here, n minus 1 0s.

So, at each level adding even $2n$, you know even numbers with $2n$ bits with take order of $\log 2n$ time, which is the same as order of $\log n$ time, because we ignore constants, and even the base of the log is not important. So, after that we have $\log n$ levels, because in each level we reduce the number of, numbers that we need to add by a factor of 2.

(Refer Slide Time: 02:55)



Time Complexity

- * There are $\log(n)$ levels
- * Each level takes
 - * Maximum $\log(2n)$ time
 - * Adds two $2n$ bit numbers
- * Total time : $\log(n)$
 - * $O(\log(n) * \log(n)) = O(\log(n)^2)$
levels x

Mc
Graw
Hill
Education

74

So, now if we realize that we have $\log n$ levels, and each level takes maximum of \log of $2n$ time, because they $2n$ bit numbers, which is the same as $\log n$, because we always ignore Constant in our work. So the total time taken is the number of levels, the number of levels multiplied with the time per level which is $\log n$ times $\log n$ which is order of $\log n$ square. So, this is the complexity with a new multiplier that we have, and this is significantly better than our previous complexity which was order of $n \log n$ right. So, $\log n$ square is much better than $n \log n$.

(Refer Slide Time: 03:37)



Time Complexity

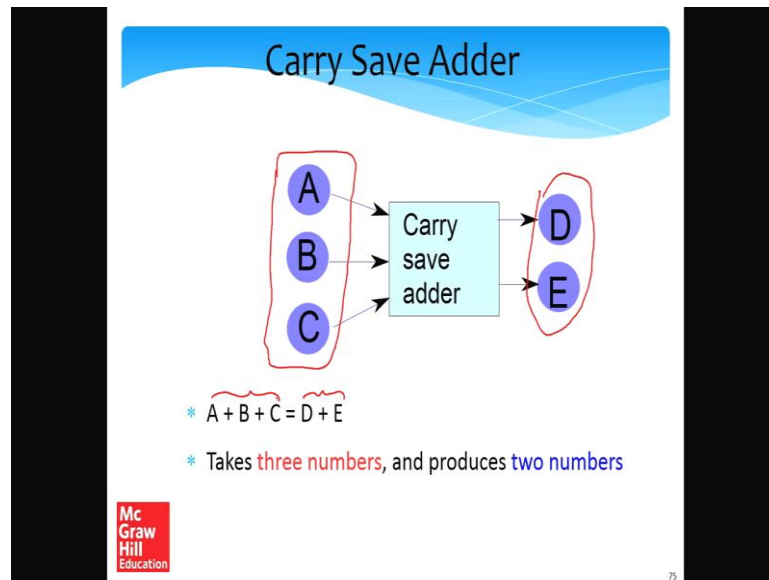
- * $O(n \log(n))$
- * Worst case input
 - * Multiplier = 10101010... 10

Mc
Graw
Hill
Education

71

So, this is a faster multiplier in that sense, but we can do better we can do much better as it turns out.

(Refer Slide Time: 03:50)



So, before we discuss how we can do better, it is important for me to mention, that there is a new kind of device that we should use, we can use and we have to use. It is a carry save adder. So, what the carry save adder does, is that it takes three numbers A B and C, and it generates two numbers D and E such that A plus B plus C is equal to D plus E. So, essentially it takes three inputs which are the three numbers and it, produces two numbers is output; such that the sum of D and E in this case is A same as the sum of A plus B plus C.

(Refer Slide Time: 04:37)

1 bit CSA Adder


- * Add **three bits** – a, b, and c
- * such that, $a + b + c = 2d + e$
- * d and e are also **single bits**
- * We can conveniently set
 - * e to the **sum bit**
 - * d to the **carry bit**

$$\begin{array}{r} 11 \\ 2 \times 1 + 1 \\ (d) (e) \end{array}$$

$$\begin{array}{r} a \\ b \\ c \\ \hline d \quad e \\ 2d + e \end{array}$$

$$\begin{array}{r} a \\ b \\ c \\ \hline d \quad e \\ 2d + e \end{array}$$

$$\boxed{a+b+c} = \begin{array}{r} d \quad e \\ 2d + e \end{array}$$



So, the way that we can do it is like this. So, let us consider a one bit carry save adder or A CSA adder. So, let us add three bits A B and C; such that A plus B plus C is equal to two D plus C. So, D and E are also single bits, and this can be done. So, I will tell you why this can be done. So, let us consider you know the smallest value is the A B and c, and largest values. So, the smallest values are 0 0 and 0. So, we will have D is 0 where the sum is 0, and we will have E is 0 if we consider the largest values, which are 1 1 and 1, then the sum is 1 plus 1 plus 1 3 which is 1 1, which is pretty much equal to two times 1 plus 1

So in this case D is 1 and E is 1. So, what we can do, is that if we add A plus B plus C right. So, since we are adding three 1 bit numbers, the potential output right can be A 2 bit number, as we just saw in this case. If he let us say we can you know conveniently set. So, in the 2 bit number, the left bit is a carry bit, and the right bit is the sum bit. We said this bit to D and this bit to E then the value of this number would be equal to 2 D plus e, which is exactly what we need.

So, what we says that in the case of one bit CSA adder D and E can be in A computed as follows, that D can be the carry bit, the more significant bit, and E can be the sum bit, which is the less significant bit, the lesser significant bit in that sense. so let me just explain once again, if you are adding three 1 bit numbers A B and C the smallest value is 0, and A largest value is 1 plus 1 plus 1, which is 3 which is 1 1. So, the final result will

fit in 2 bits. If I keep the 2 bits over here let me do one thing. So, let me call the sum bit which is this bit as e, and let me call the carry bit which is the most significant bit is D. So, as we can see the final result can also be expressed as two D plus E, which is exactly what we want to do, this is exactly what we want to do. So, we basically have A plus B plus C, is equal to 2 D plus E.

So if now if I take a look at the previous diagram, what I have done is, I have taken three 1 bit numbers in produce two numbers, where 2 D can be thought as D and e can be thought as E. So, basically produce two numbers is a sum of these two is equal to the sum of these three. Now let us do this for more bits right, at the moment A B and C where only 1 bits each. So, let us consider wider versions of A B and C.

(Refer Slide Time: 07:57)

n-bit CSA Adder

$$\begin{aligned}
 A + B + C &= \sum_{i=1}^n A_i 2^{i-1} + \sum_{i=1}^n B_i 2^{i-1} + \sum_{i=1}^n C_i 2^{i-1} \\
 &= \sum_{i=1}^n (A_i + B_i + C_i) 2^{i-1} \\
 &= \sum_{i=1}^n (2D_i + E_i) 2^{i-1} \\
 &= 2 \sum_{i=1}^n D_i 2^{i-1} + \sum_{i=1}^n E_i 2^{i-1} \\
 &= 2D + E
 \end{aligned}$$

Mc Graw Hill Education

So, let us do A little bit of algebra, to compute the two numbers. To compute two numbers D and E such that A plus B plus C is equal to D D plus E. So, way the way that we will do it, is that we will use the same logic that was used here, and we will just extend our results. So let us express A as sum of you know it is bit. So, any number A can be A expressed as. So, let us me consider A right, let the bits in A B A 1 to A n. So, let the number A be A n to A 1.

So, this can be expressed as this number here is A. Similarly we can express B in the same way as A sum of parts of two, and each part is multiplied with the corresponding bit of B, and we can do the similar things the C. Then what we can do is, we can group

all the terms together as $A_i + B_i + C_i$ and multiply them by 2^{i-1} . So, what we have seen, in the previous result for 1 bit. This can be expressed as $2^i D_i + E_i$ times 2^{i-1} , you know this is something that exactly we have seen here. So, we can do the same you know the same thing sum of three bits is the same as sum of two numbers.

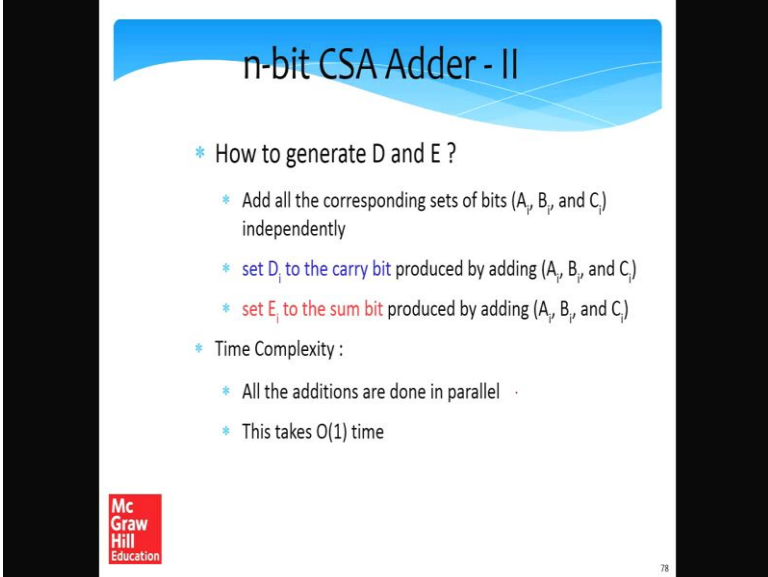
Once we do the same, we can then. So, this, since you are seen in the previous slide, slide number 76 I am not repeating. Once you have done this again we can break up this expression over here. So, we will have two terms i equal to 1 and $2^n D_i$ times to the power i . So, this is one binary number right, which we can treat as d , and this is one more number which we can treat as E right. So, what we see here is that we have replaced the sum of A of A and B and C $A + B + C$ as two numbers D and E right. So, let me just get rid of the ink, and you will be able to see it better. Or computing D and E is also very easy; the reason computing it is very easy is that, we take. So, in parallel we consider. So, since in hardware everything is parallel.

So, we considered the corresponding i th bit, we add them. So, we get this D and E , where E is the sum bit as we have seen just before in the previous slide. Sorry E is the sum bit and D is the carry bit right. So as we have seen here we just add them up. Then all that we need to do is that. So, actually what we can do is that in A here, we can take the two out and this written $D_i \times 2^{i-1}$. So, this is one binary number which is composed entirely out of the sum bits, where we after adding individual bits. And this is one more binary number which is composed entirely out of the carry bits. Carry that is obtained by adding the three bits at the same positions; so ones we get E , E is as is but ones you get this number which is left shift multiplied by 2, which means left shift it by one position and we get the number D .

So essentially computing both D and E can be done very quickly in constant time, or an order one time, and the way it is done is that we consider the i th bit of all three numbers A B and C , we add three 1 bit numbers. This can be done in parallel. So, will get a sum bit and carry bit. So, we from one number just with the some bits of all the, you know individual sums right, and one more number with the carry bits that we obtained by adding all trip let us of numbers right, of the form $A_i + B_i + C_i$ and then we just take the number that we get and left shifted by one position, which is to effect this multiplication by 2, and one number becomes D and the other number becomes E

So this can be done. So, this particular step of extracting the i th bit, adding the three i th bit getting D_i and E_i all of this can be done in parallel. So, this takes constant time, then forming the number with these bits also takes constant time, and left shifting also takes constant time. So, it is not an issue right by one position. So, that is a reason getting D plus E from A plus B plus C . The entire operation is a constant time operation.

(Refer Slide Time: 13:44)



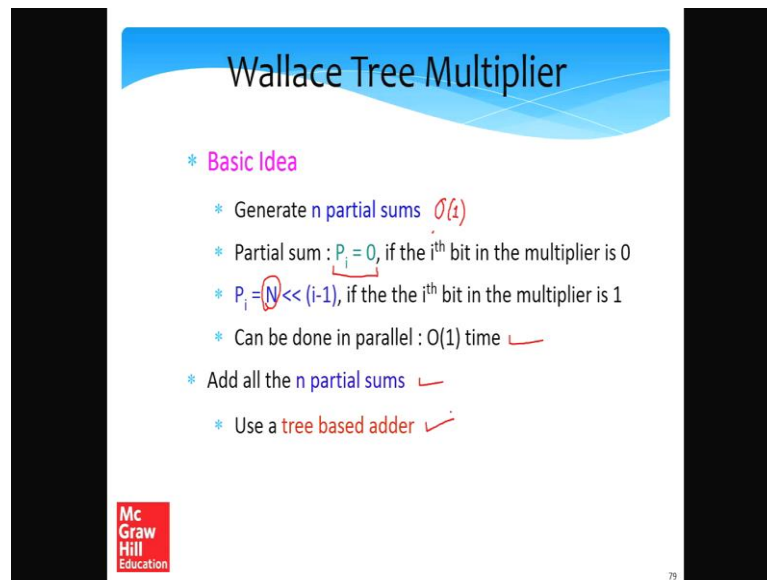
The slide is titled "n-bit CSA Adder - II" and contains the following text:

- * How to generate D and E ?
 - * Add all the corresponding sets of bits (A_i , B_i , and C_i) independently
 - * set D_i to the carry bit produced by adding (A_i , B_i , and C_i)
 - * set E_i to the sum bit produced by adding (A_i , B_i , and C_i)
- * Time Complexity :
 - * All the additions are done in parallel
 - * This takes $O(1)$ time

Mc Graw Hill Education logo is visible in the bottom left corner of the slide.

So, I just summary of what we just discussed, how do we generate D and E. Well we add all the corresponding sets of bits A_i , B_i , and C_i independently. Simultaneously independently and simultaneously we said D_i to the carry bit produced by adding A_i , B_i , and C_i and we said E_i to the sum bit produced by adding A_i , B_i , and C_i . So, since all the operations are done in parallel, it takes order of one time.

(Refer Slide Time: 14:13)



The slide is titled "Wallace Tree Multiplier" in a blue header. Below the title, there is a section labeled "* Basic Idea" in pink. This section contains five bullet points, each with a blue asterisk. The first bullet point says "Generate n partial sums $O(i)$ ". The second says "Partial sum : $P_i = 0$, if the i^{th} bit in the multiplier is 0". The third says " $P_i = N \ll (i-1)$, if the the i^{th} bit in the multiplier is 1". The fourth says "Can be done in parallel : $O(1)$ time". The fifth says "Add all the n partial sums". Below these points, there are two more items: "Use a tree based adder" and the McGraw Hill Education logo. A small number "79" is in the bottom right corner of the slide.

- * Basic Idea
 - * Generate n partial sums $O(i)$
 - * Partial sum : $P_i = 0$, if the i^{th} bit in the multiplier is 0
 - * $P_i = N \ll (i-1)$, if the the i^{th} bit in the multiplier is 1
 - * Can be done in parallel : $O(1)$ time
 - * Add all the n partial sums
 - * Use a tree based adder

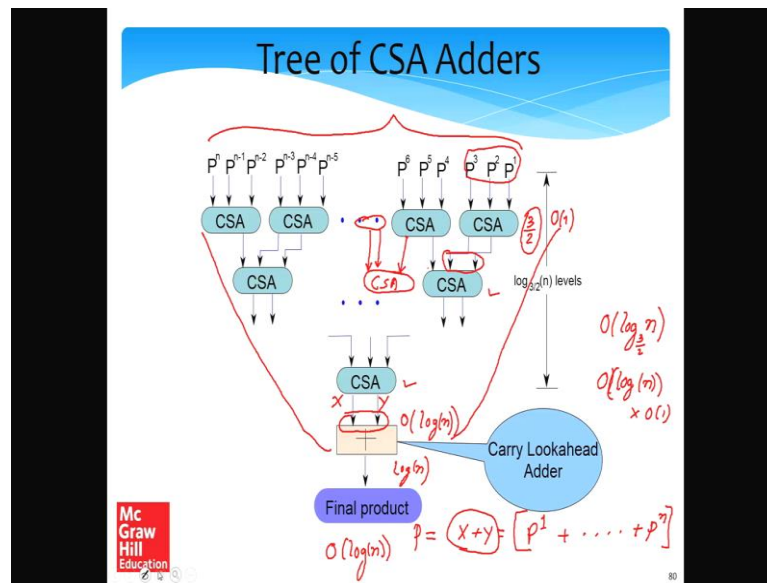
McGraw Hill Education

79

So, let us now discuss the Wallace Tree Multiplier, the idea is like this that we generate n partial sums. So, this as we have seen in the previous example can be done very quickly. So, this can be done in order of one time, where we generate the n partial sums. How do we do this? we take a look at each bit in the multiplier if it is one, the partial sum is the multiplicand; of course, shifted by a certain number of positions to the left; otherwise the partial sum is 0; that is the exactly what we do.

If the i^{th} bit in the multiplier is 0 the partial sum is 0 P_i is 0; otherwise that P_i is the multiplicand, left shifted by i minus 1 positions, if the i^{th} bit in the multiplier is 1. So, this can be done in parallel. Once we have n partial sums we can use a tree based adder, to affect the multiplication, the way this is done is like this, that we have a similar tree as we had in the log n square multiplier.

(Refer Slide Time: 15:12)



So, in this case, what we do is, that we divide n partial sums into groups of three. We have a carry save adder CSA adder that takes in three inputs, and that produces two outputs. So this adder produces two outputs, and then the next adder; one output is sent here, and one more adder is sent to another carry save adder, which takes two outputs from here, and the two outputs from the next partial sum adder right, from the next CSA carry save adder. So, in this case if you have.

So, basically each carry save adder here is decreasing the number of inputs by 3 by 2. So, if there are three x inputs here, there are two x inputs in the next level. So, what we do is that we create a tree of adders, where we start with the partial sums and keep on adding them. So, recall that the carry save adder takes three inputs and produces two outputs, where the sum of the outputs is the same as the sum of the inputs. Then what we do is that we keep on adding levels till we finally have a case where we have only two outputs, and a sum of these two if the sum of all the partial sums.

So the approximate number of levels that we will require is order of, \log of n to the base 3 by 2. Why three by two. It is three by two, because that every level where decreasing the number of partial sums by a factor of 3 by 2. So, we are just adding, so ultimately we will reach a point for the sum of all the partial sums is just some of these two numbers. Once we have two numbers we can use a regular carry look ahead adder, which again takes in order $\log n$ time to add these two numbers. So, these two numbers will be

much larger, and they will have roughly $2n$ bits. So, that is the complexity relation still holds, and then will get the final product.

So let us now compute the total time of this tree of CSA adders, which is also called a Wallace Tree Multiplier. So, what we have is, since we have the tree over here, the tree has $\log_3 n$ levels, and each level takes order one time, because one CSA adder runs in order one time. So, the total time that is required is order of $\log_3 n$, which is the same as orders of $\log n$, and the reason being at the base of log does not matter, because in a log to any base and log to any other base, the related via constant factors. So, those constant can be ignored. So, once we have order of $\log n$ levels, in the time per level is order one. The total time required to compute these two numbers; let us call them x and y .

So here what we have is x plus y , is the sum of all the partial sums. And the sum of all the partial sums is essentially the product the result of the multiplication, which essentially we are producing two numbers, whose sum is the final product p . So, first thing that we do is, we add this n partial sums, to come up with two numbers x and y say that x plus y is equal to p_1 till p_n , via a tree of carry save adders, and each carry save adder takes order one time. And since we have $\log n$ such levels the total time is $\log n$, and finally, we need to add x and y with a carry look ahead adder, which again takes $\log n$ time. So, as we can see the final result, we can get in order of $\log n$ time.

So this is the fastest possible multiplier, and this is very commonly and frequently used in commercial circuits. So, as you can see this is very fast. And the reason it is fast, is basically, because of this carry save adder right. and you know had we, not had the carry save adder it would have been difficult, but because we have this carry save adder which is a very fast unit, which can take three inputs and produce two outputs for the sum of the two outputs, is the same as the sum of three inputs.

We are quickly. we know we can quickly reduce the sum of n numbers to sum of only two numbers within $\log n$ time, and then we add the two numbers we get the final product, which is the result of the multiplication. So, we have achieved our goal of having a $\log n$ time multiplier.

(Refer Slide Time: 20:19)

Tree of CSA Adders

- * Group the **partial sums** into sets of 3
 - * Use an **array of CSA adders** to add 3 numbers (A,B,C) to produce two numbers (D,E)
 - * Hence, **reduce the set of numbers by 2/3 in each level**
- * After $\log_{3/2}(n)$ **levels**, we are left with **only two numbers**
- * Use a **CLA adder** to add them

log(n)

Mc Graw Hill Education

81

So, this slide only summarizes what we have discussed, just a quick recap use an array of CSA adders to add three numbers to produce two numbers. So, we reduce the set of numbers by two third every level or divided by 3 by 2. After of $\log n$ to the base 3 by 2 level so left with the only two numbers. We use a carry look ahead adders to add them. So, this takes $\log n$ time. So, the total time as a result is $\log n$.

(Refer Slide Time: 20:49)

Time Complexity

- * Time to generate all the **partials sums** $\rightarrow O(1)$
- * Time to reduce **n partial sums** to sum of **two numbers**
 - * Number of levels $\rightarrow O(\log(n))$
 - * Time per level $\rightarrow O(1)$
 - * Total time for this stage $\rightarrow O(\log(n))$
- * Last step
 - * Size of the inputs to the CLA adder $\rightarrow (2n-1)$ bits
 - * Time taken $\rightarrow O(\log(n))$
- * Total Time : $O(\log(n))$

Mc Graw Hill Education

82

(Refer Slide Time: 20:58)

Outline

- * Addition
- * Multiplication
- * Division
- * Floating Point Addition
- * Floating Point Multiplication
- * Floating Point Division

7 part 1

7 part 2

McGraw Hill Education

83

So against the time complexity is $\log n$ this. So, I am skipping this slide, because you have already discussed the content. So what have we done? What we have done is that we have focused on addition and multiplication in this, you know in the last three lectures. So, now, will move to other things, we will move to division, floating point addition, floating point multiplication, and floating point division. So, because this was a long chapter I divided the slides into two sub slides, like two set up sub slides.

So, you will find one slide as chapter 7, part 1 and other is chapter seven part 2. So, what will do is when move to the next slide set which is part 2. So, we will discuss these four topics in part 2. So, the current part which is part 1 is over.