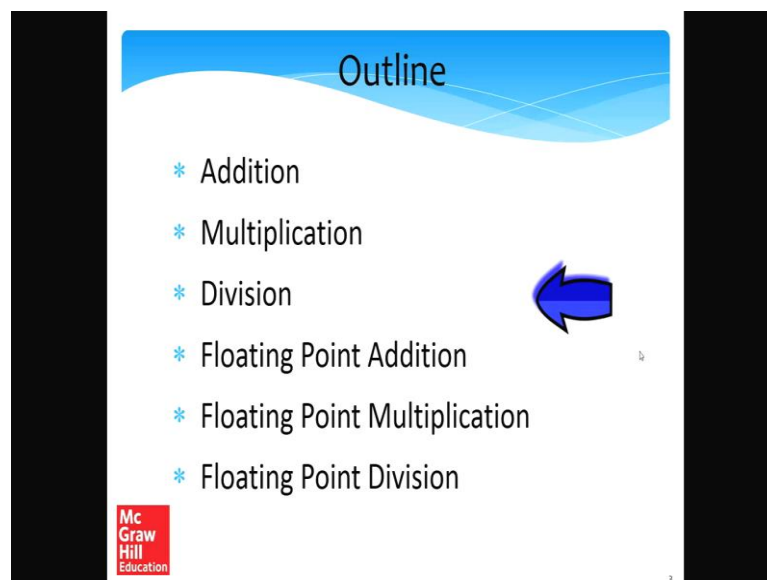


Computer Architecture
Prof. Smruti Ranjan Sarangi
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Lecture - 20
Computer Arithmetic Part-IV

Welcome back. This is the second slides set in the Computer Arithmetic chapter. So, since the chapter was long, we divided it into two slide sets. This is the otherwise 4th lecture in this series.

(Refer Slide Time: 00:44)



So, we had promised to discuss a couple of things. We were promised to discuss division and floating point operations; namely addition, multiplication, and floating point division.

(Refer Slide Time: 00:51)

Integer Division

- * Let us only consider **positive numbers**
- * $N = DQ + R$ (1) $\frac{N}{D} = (N \div D)$
- * N → Dividend
- * D → Divisor
- * Q → Quotient
- * R → Remainder

Properties

- * [Property 1:] $R < D, R \geq 0$
- * [Property 2:] Q is the largest positive integer satisfying Equation 1 and Property 1

Handwritten notes: $\frac{N}{D}$, $5 \div 2$, $Q=2$, $R=1$, $101/20$

McGraw Hill Education

So, let us now divide; so let us only considered positive numbers in our discursion because dividing negative numbers this is kind of tricky. Well, it is tricky in the sense that we need to. First understand that when we are doing integer division, and there is a negative number involved, there are different conventions for the remainder in the, some convention says that reminder is always positive. Some convention says that remainder has the same sign as the divisor. So, actually what most library's or processor would do, is that they would first convert the numbers to positive numbers, do the division with positive numbers, and then adjust the remainder according to the convention.

So, the again there are different ways are doing it, but let us just take to positive numbers in our discussion, because number 1 is simpler. Secondly, since division is a fairly long operation, and is not possible to have log N time algorithms very easily. It is ok to convert negative numbers to positive do the division, and then adjust the signs.

So, let us now introduce some basic terminology. The terminology is like this, that when we are dividing, sorry. Let us say that we are dividing N by D right. So, this is in a slash operation same as integer division, N divided by D. So, in this case we will have a quotient and a remainder. For example, if I divide 5 by 2, the quotient is 2 and the remainder is 1. So, this can alternatively we said, that N which is a dividend. So, basically the top number is called the divided, and this is the divisor, N is equal to DQ plus R, where N is the dividend D is the divisor. So, we are essentially dividing the

dividend by the divisor, we are computing N by D then we will have a quotient. Quotient is a result of the division for example, if we divide 100 by 20 the quotient is 5. If we divide 101 by 20 also an integer division the quotient is 5, but in this case the remainder will be 1. So, there are two things the division has two outputs; a quotient and the remainder, both of them need to be computed.

Now, let us take a look at some properties right, in this equation N equals DQ plus R . So, 1 property is that the remainder has to always be less than the divisor right that has to be the case. The reason that that has to be the cases; otherwise you know the division is wrong for example, if I divide 101 by 20 right, if I divide 101 0 1 by 20, the quotient cannot be 4 in the remainder cannot be 21. So, it basically means of the division is not been done correctly. So, the quotient has to be 5 and the remainder has to be 1 in this case.

So, R is less than D , further more R is greater than equal to 0. So, we all always have a positive remainder, if we are considering only positive numbers. We can then. So, let us remember these properties, will refer to this is property 1; the property that the remainder is lesser than divisor. The other one will refer to as property 2, which says that the quotient q , is the largest positive integer, that satisfies equation 1. I should probably call this is equation 1 sorry right. That satisfies the equation and this is equation 1.

So, the quotient is the largest positive integers. So, this is also obvious. So, again if I am dividing 101 by 20; so 5 is the largest number that satisfies the equation 1, and essentially the equation and the property 1. So, why is this the case, because 5 times 20s 100 plus 1, 6 will not do, because if I have 6 times 20 then DQ will become 100 20s R has to be minus 19, but what we understand from here, from property 1 is said R , is always greater than equal to 0. So, as a result it cannot be negative.

So, both of these properties are otherwise obvious, but you know it is nice to write them down, because the any division algorithm has to obey both of these properties and. So, these are also sufficient properties, not necessary, but sufficient properties we need to find. So, Q is the largest positive integer, which satisfies the equation, and also property 1 needs to hold, where R is less than D and R itself is greater than equal to 0. So, we need to ensure these two property is 1 and 2 always hold.

So, let us. So, maybe in other way that we should modify the slide is that again a satisfying equation 1 and property 1 rights that will fix this slide all right. So, let us do one thing, let us do a little bit a math.

(Refer Slide Time: 06:41)

Reduction of the Division Problem

$$\begin{aligned}
 N &= DQ + R \\
 &= DQ_{1\dots n-1} + DQ_n 2^{n-1} + R
 \end{aligned}$$

$N' = N - DQ_n 2^{n-1}$
 Q_{n-1}
 $N' = N - D2^{n-1}$

$$\frac{N - DQ_n 2^{n-1}}{N'} = \frac{DQ_{1\dots n-1}}{Q'} + R$$

$$N' = DQ' + R$$

We have reduced the original problem to a smaller problem

McGraw Hill Education

So, let us start with our division equation which is N equals DQ plus R, and let us assume that the quotient q, which is of course, represented in binary the bits are Q n to Q 1. So, what we can do is that we can divide the Q the quotient sorry. So, we can sort of break in any 2 parts 1 is Q n times 2 raise to the power n minus 1 plus 1 number which consist of the remaining n minus 1 bits right. So, we have also done this before in chapter 2.

So, 1 good idea would be that somebody you know who was forgotten what we learnt in chapter 2, all about bits and floating point numbers and so on, can go back to chapter 2 and refresh the knowledge, because we will be using many results here. So, what did I do again? I considered the quotient to be N bit number, and I take the MSB outs. So, is a Q n times 2 raise to the power n minus 1 plus the number that is left out of the rest n minus 1 bits. So, if I bake the quotient in this way I have one component over here, which is the first n minus 1 bits of the quotient. I have one more component over here which is divisor times this thing, and this thing is the Q n is the MSB of the quotient more significant bit times 2 raise to the power n minus 1 plus the remainder r.

Now, what I can do, is that I can take this term to the left side, and have N minus D times $Q_n 2^{n-1}$ is equal to D times. So, let the number of the quotient. So, so let the number form by the first $n-1$ bits of the quotient let us call it Q_{dash} . So, what we will have is, D times Q_{dash} plus R right, and let me call this term over here the N_{dash} .

So, what we sees that this equation looks very similar to this equation right. There is no difference at all in the structure, but there is a little bit a difference in the term. So, the remainder is the same mind you, and the divisor is the same, but the quotient in this case has this is, the N bit quotient in this case is the same quotient, but the first $n-1$ bits, in this case is the original dividend, in this case is the dividend the dividend is. So, what is the dividend in the upper case, it is the original dividend N minus the divisor times the n th bit of the quotient times 2^{n-1} right. So, this is the dividend that is being used.

So, what we have done, is that if we can somehow figure out, the n th bit, the most significant bit of the quotient, if there is a some way of figuring that out, then what we can do, is that we can compute a new dividend which is N_{dash} , and solve a smaller division problem with a quotient has $n-1$ bits. Similarly, I can then have one more, then you know I can carry on in this same manner, same fashion, I can then have one more equation which is Q_{dash} plus R , where this contains of the last $N-2$ bits of the quotient. If I have some way of again figuring out, the $n-1$ th bit of the quotient I can come to this equation.

Similarly, I can go down, down, down, down till I sort have reach 1 point. So, let me maybe you put a star, sorry this is the bad star, let me erase that. Yes, the better star till I reach one point. So, this is the divisor times the first bit of the quotient Q_1 plus R . Now at this point if I have some magical way of figuring out what is the least significant bit of the quotient, then what will ultimately remain is, we will ultimately what will remain is at the end, we will only have the remainder left. So, you know what is left will be the reminder, and it every point we will get 1 bit each in the quotients starting with the n th bit, till the first bit.

So, what we are doing is, that we are divide we are. So, gradually reducing the division problem from a problem of dividing you know, I am sorry from a problem of computing

and N bit quotient, to a problem of computing and n minus 1 bit quotient, to a problem of computing a 1 bit quotient right. So, we at every stage we are sort of making the problems slightly smaller, and slightly simpler, till at the end we need to compute a single bit, and finally will be left with the remainder.

So, this sounds like a good plan it is just that the secret sauce is missing, and the secret sauce here is how on earth do we compute the nth bit of the quotient, because if we can do that then we will be able to solve the entire problem, because at every stage what we will do is, that we will keep on reducing the problem to a smaller version, till we reach the end we get all the bits of the quotient, and what is left will be the remainder.

(Refer Slide Time: 13:26)

How to Reduce the Problem

- * We need to find Q_n (0, 1)
- * We can try both values - 0 and 1
- * First try 1
- * If $(N - D \cdot 2^{n-1}) \geq 0$, $Q_n = 1$ (maximize the quotient) $Q_n = 1$
- * Otherwise it is 0 $Q_n = 0$
- * Once we have reduced the problem $(N' = N - D \cdot 2^n)$ n-bit \downarrow (n-1)-bit \vdots (1)-bit
- * We can proceed recursively

McGraw Hill Education

So, let us see right, how do we reduce the original problem to a slightly smaller problem. So, what is the main issue? The main issue here that is bothering us is how to find Q_n , the nth bit. The most significant bit of the quotient how is that to be found. So, let us think of, it is so simple, but let us sort of locate at from a distance. The nth bit of a quotient says after all the single bit right. So, if it is a single bit how many values can it have? It can have only 2 values 0 and 1.

So, since there is a two values what we can do, is that we need to. Well we can try a brute force approach, so you can try both the values; both 0 and 1. So, let us first try one, so if N minus. So, what do we need to do, we basically need to compute N dash is equal to N minus D times Q right. If Q_n is equal to 1 then N dash becomes equal to N minus D

times. So, what we can do is, we need to compute this, and since we have to maximize our quotient. So, why do we need to do maximize the quotient, it is property number 2 that we want to maximize the largest positive integer.

So, since we want to maximize our quotient, what we would like to do, is that we would try to set Q_n is equal to 1, if the new dividend is still positive right. So, $N - D \times 2^{n-1}$ if that is still greater than equal to 0, then it means that the n th bit of the quotient can be set to 1, and we in a sense will maximize the quotient. So, then find if this is the case we set Q_n is equal to 1, because the original N was to have a quotient as large as possible. And if we can get away with setting the n th bit of the quotient is 1, this means that it is a right guess, it is the correct guess, and we will set the new dividend to be this, and solve the smaller division problem right, where will get the rest $n - 1$ bits of the quotient.

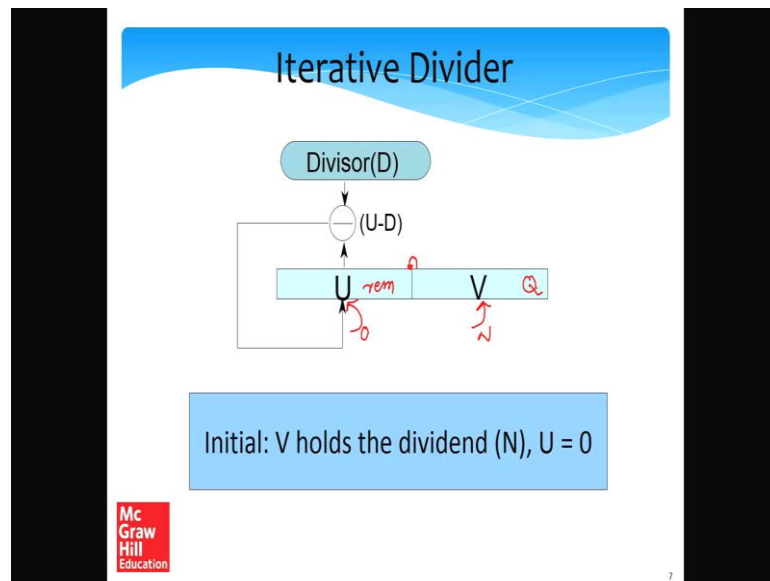
However if this term is less than 0, then it means of the largest bit of the quotient cannot be 1; hence we set it to 0. We set what to 0. We set Q_n to 0 all right. So, what we have done is, that we have does computed Q_n , and we have also reduced the problem. Problem was initially to compute an N bit quotient. Now we are computing an $n - 1$ bit quotient right. So, how are you doing that, we consider the quotient to be an N bit number we make 2 guesses for Q_n ; 1 guess was 1, and the other guess was 0 with 1.

We tried to see that whether the problem still remains valid if you make a guess of 1, and if it remains valid we agree that in a Q_n should be 1. So, we set it to 1 the n th bit of q ; otherwise we set it to 0. And then we compute the new dividend, which is $N - D \times 2^{n-1} \times Q_n$ right, and we proceed recursively. Which means that we call the same process over and over again I will bit with reduced or smaller arguments, till we are left with a single bit.

So, the long and short of this slide, the summary of this slide, is that we now have a method of computing a single bit in the quotient, which is the most significant bit we have a method. What we do is, that we first make a guess we see if you know whether if it is 1 little work or not. The reason for this is that we want to maximize the quotient. If $N - D \times 2^{n-1}$, if this number is greater than equal to 0 then it means that the quotient can be set to 1; otherwise it means to the quotient cannot be set to 1, and in this manner we proceed and we gradually reduce the problem.

Alright, let us proceed and then I will show an example.

(Refer Slide Time: 18:14)



So, before I actually show the mechanism, let me make it quick trip to the previous slide, and let me in a delete all of this stuff and explain within very quick and small example.

(Refer Slide Time: 18:28)

How to Reduce the Problem

- * We need to find Q_n
- * We can try both values – 0 and 1
 - * First try 1
 - * If : $N - D2^{n-1} \geq 0$, $Q_n = 1$ (maximize the quotient)
 - * Otherwise it is 0
- * Once we have reduced the problem
- * We can **proceed recursively**

4-bit number system
 $n=4$
 $13/3 = (Q=4, R=1)$
 $N=13, D=3$
 $N - D2^{n-1} = 13 - 3 \times 8 < 0$
 $Q_4 = 0$
 $N' = N = 13$
 $13 - 3 \times 2^2 = 7$
 $7 - 3 \times 2 = 1$
 $1 - 3 \times 1 < 0$
 $Q_3 = 1, N' = 1$
 $1 - 3 \times 2^1 < 0$
 $Q_2 = 0$

$Q_1 = 1$
 $1 - 3 \times 2^0 < 0$
 $Q_1 = 0$

Mc
Graw
Hill
Education

So, what I do, sorry. So, let me try to divides. So, let me cons considered a 4 bit number system all right. .

So, in this case N which is the number of bits in the number system; that is equal to 4. Furthermore let me try to divide 13 by 4. No let me try dividing 13 by 3. So, what is the quotient, well the quotient in this case is 4 and the remainder is 1. What is the binary representation of quotient, it is 0 1 0 0. So, let us try to do something. So, in this case N is 13, the divisor is 3. So, let me first, in the first iteration number 1, let me try to compute N minus D times 2 raise to the power n minus 1, which is equal to 13 minus 3 times 2 to the power 4 minus 1 which is 2 cube which is 8. So, 13 minus 24 is less than 0. So, as a result we will set the most significant bit of the quotient to be 0. As you can see this is the correct answer. So, in this case N dash which is the new remainder dividend will remain the same as a old dividend, which is 13 fine.

After that let us go to the second next iteration; in this case let me do the same thing 13 times. So, say in this case N was 4, now in this case N will become equal to 3, because we are moving to a smaller number system right. So, say is the next iteration pretty much. So, in this case what we do, is that we do 13 minus 3 times 2 raise to the power 3 minus 1 which is 2 to the power 2, which is equal to 1, which is greater than equal to 0. So, as a result we set Q_n is equal to 1, and as you can see this is exactly, sorry not given, but this is Q_n minus 1. So, let us call it Q_4 and Q_3 you know; that is slightly easier the 4th bit of the quotient and the third bit of the quotient. Just quickly erase this part erase this part.

So, let us say that this is Q_4 and this is Q_3 . Fine let us go to the next iteration which is iteration number 3. So, so in this case what is N dash, N dash is equal to 13 minus 3 times 2 square, which is 1. So, then I will again compute 1 minus 3 times. So, in this case N again becomes 2. So, 2 to the power 2 minus 1 is 1, which is less than 0 so basically Q_2 is equal to 0. Well, say again Q_2 here is equal to 0. So, we have computed this properly. So, go to we go to the last iteration, which is the 4th iteration. So, here again the new N dash is equal to 1, because no change was made in the last time. So, what we need to compute is, 1 minus 3 times 2 to the power. In this case is 1 minus 1 which is 2 to the power 0 is 1, this is again less than 0. So, we can say that Q_1 is equal to 0.

So, as we can see Q_1 is equal to 0 over here. So, what we basically get to see, is that we have computed the quotient which is 0 1 0 0 correctly, using this method. So, we will get to see some more examples of this method, but this is at least a quick way of explaining how this method works. So, how will we actually implemented in hardware. Well, very

similar to the iterative multiplied, so we will have the divisor here in D initially. So, we will have 2 registers here, and the same way we had for the iterative multiplied in the purpose for the, purposes of shifting, they act as the same registered. So, initially we will hold the dividend n, and you will contain the number 0 and D will contain the divisor.

(Refer Slide Time: 23:55)

Algorithm 3: Restoring algorithm to divide two 32 bit numbers

Data: Divisor in D, Dividend in V, U = 0
 Result: U contains the remainder (lower 32 bits), and V contains the quotient.

```

i ← 0
for i < 32 do
  i ← i + 1
  /* Left shift UV by 1 position
  UV ← UV << 1
  U ← U - D
  if U ≥ 0 then
    q ← 1
  end
  else
    U ← U + D
    q ← 0
  end
  /* Set the quotient bit
  LSB of V ← q
end
  
```

The diagram shows two registers, U and V, each 32 bits wide. Register U contains the remainder, and register V contains the quotient. The dividend is loaded into register V. The divisor is in register D. The algorithm involves left-shifting the UV register by 1 position, subtracting D from U, and setting the quotient bit q based on the sign of U. Handwritten annotations include a red box around 'q ← 1', a red arrow pointing to 'U ← U + D', and a red box around 'q ← 0'. A red arrow also points to 'LSB of V ← q'. The diagram also shows the mathematical representation of the restoring division process: $N = D \cdot 2^{i-1} \cdot q_{i-1}$.

So, here instead of addition every since division is repeated subtraction; every iteration will subtract, in a possibly subtract D from U and write the value back to the U register all right. So, let us take a look at the algorithm which is very simple, its call the restoring algorithm. So, the restoring algorithm what it does is, we divide 232 bit numbers, 32 64 does not matter. So, the input is a data the divisor is in registered D. So, as we can see the divisor is here in registered D, the dividend is an registered V. So, which is this register, the dividend is loaded over here and. So, if you left shift then the essentially this bit will come here, and at the end we want u, the registered U to contain the remainder, and registered v to contain the quotient. So, U will at the end contain the remainder; that is what we want and register v we wanted to contain the quotient at the end.

So, what we do, is that we run in the iteration of for loop for 32 times for I equals 1 to 32; that is all that you know these lines 2 is that you run a for loop for 32 times. So, then we left shift the UV register, the same way we did with the case the multiplier. So, in that case we were right shifting in this case we left shift. So, we do a left shift by 1 position.

Then what we do is, we subtract D from the register U . So, so we subtract the divisor from the registered U . If U is greater than equal to 0 right, then we set Q to 1, Q is a temporary variable we set it to 1. Otherwise what we do is we restore the value of U . So, U becomes its old value; that is the reason is called a restoring division, because initially we subtract it, but if you find that the result has become negative. Then what we do is that we add the value of the divisor back to u , so U becomes its old value, and we set Q to 0, and then we set the quotient bit least significant bit of v to Q . So, that is what we do.

So, before we look at an example, it is very very important for me to give a certain feel of why this algorithm is working, because to get a feel is very important. So, let me use some real estate here to draw. So, let us consider the 2 registers. The 2 registers that I have R registers u ; let me just write them at the corner I need some space, and registers v . Say initially the dividend is completely loaded in registered v till we come to this line, in which the dividend is shifted to the left right. So, the dividend is shifted to the left by 1 position, so basically the dividend occupies. Let us consider this much to be 1 position. So, pretty much occupies this much.

So, let me call this the dividend version. So, I am just considering the first iteration, but all iterations will be give in the same fashion. So, this is the dividend N . of course, the sum amount of space created over here, but let us ignore this. So, so this will not coming to our calculation, so it can ignore.

Now, what I do is that I subtract the divisor from registered U . So, if I consider this part as the divisor right, from register U . and if I consider this as an entire number starting from here to the right of this I ignored, but let us it starting from here if I consider this much to be a full number. What I am essentially computing is. I am computing and N dash which is equal to N minus the divisors. So, mind you the divisor as compared to the beginning of this number is shifted to the left. How many positions are it shifted. So, basically this entire distance is N bits.

And this distance here is 1 bit. So, as compared to the beginning of the place where is dividend starts right. So, the as these bits we have ignored, because we have left shifted anyway in this line. So, as compared to the beginning of where the dividend starts, the divisor is n minus 1 position to the left all right. So, if you are looking at the screen, the divisor is n minus 1 position to the left. So, when I am doing the subtraction U is U

minus D . What I am essentially doing is that for this number, which begins from the beginning of the dividend, till the end of register U . So, this number at the beginning is all 0s.

So, let it remain in that fashion. So, this number that I have, I am essentially subtracting the divisor multiplied by 2 raised to the power $n - 1$. Well, I can maybe clean up the minus 1 a little bit. And since I initially assume that I am multiplying this, you know the quotient bit is 1 I can add Q_n , where Q_n is 1 well. So, does this particular equation look familiar? It definitely should because this equation is exactly this equation over here. So, let me maybe be a good idea to clean the slide, this is exactly this equation right. This is the same term $N - D \cdot 2^{n-1} \cdot Q_n$.

So, this is exactly what we are doing, that we have computed the new dividend, and the way we have done that is via this particular hardware mechanism, but in a sense what is being computed inside, is the old dividend minus the divisor multiplied by 2 raised to power $n - 1$ times Q_n , where we are assuming that the quotient bit is 1. So, this is, you know this is just a hardware implementation and the algorithm that we have discarded.

Now, if the new value of U is less than 0, which means that you know this new dividend is not valid, then we will set the quotient bit to 0, and restore the value of U right. So, we do not want a negative dividend, so we will restore it that is what we have been doing; otherwise it is greater than equal to 0. Well fair enough we have completed the quotient bit to be 1. So, once we have done that the quotient bit that we compute that is set to this a new space over here which is Q_n , because this space is anyway not used we are not factoring it into the dividend. So, this will become Q_n . So, gradually what will? Now, let us consider the next iteration right. So, this is the first iteration. If I consider the next iteration the situation will be very similar. I am just trying a smaller version of the U and v registers.

The situation will be very similar, we will instead of. We will start a dividend from a newer position and it will be 2 bits here that instead of 1 bit, and we can consider the rest as a new dividend $N - D$ that was computed in the previous iteration. 1 bit over here will be the n th bit of the quotient Q_n which we have computed, and this bit will be Q_{n-1} that needs to be computed and the current iteration, which is the second

iteration. Then we can do exactly the same right in the next iteration, and compute Q_n minus Q_{n-1} . Exactly the same logic and the same logic will work. The reason that this will work, is basically because you know this equation has a recursive structure. So, initially you know N is 32, and then N is 31 30 29 and so on.

So, in consonance with these equations, this hardware will perform appropriately. So, what will happen as I keep on going towards the end? What will happen is it at 1 point the dividend will start from here. So, this entire register here, the v register will contain the quotient, and what will be left here will be the remainder, after we have computed all the quotient bits. So after all, the quotient bits have been computed, what will essentially remain is the remainder in registered u , which is what we initially started out to find.

(Refer Slide Time: 33:50)

Example

Dividend (N) 00111 ⁷
 Divisor (D) 0011 ³

beginning: U 0000 V 0111

1	after shift:	00000	1110
	end of iteration:	00000	1110
2	after shift:	00001	110X
	end of iteration:	00001	1100
3	after shift:	00011	100X
	end of iteration:	00000	1001
4	after shift:	00001	001X
	end of iteration:	00001	0010

Quotient(Q) 0010
 Remainder(R) 0001

Handwritten notes:
 $7 \div 3 = 2 \text{ R } 1$
 $O(n \log m)$
 $Q=2$
 $R=1$
 $K=1$

Mc Graw Hill Education

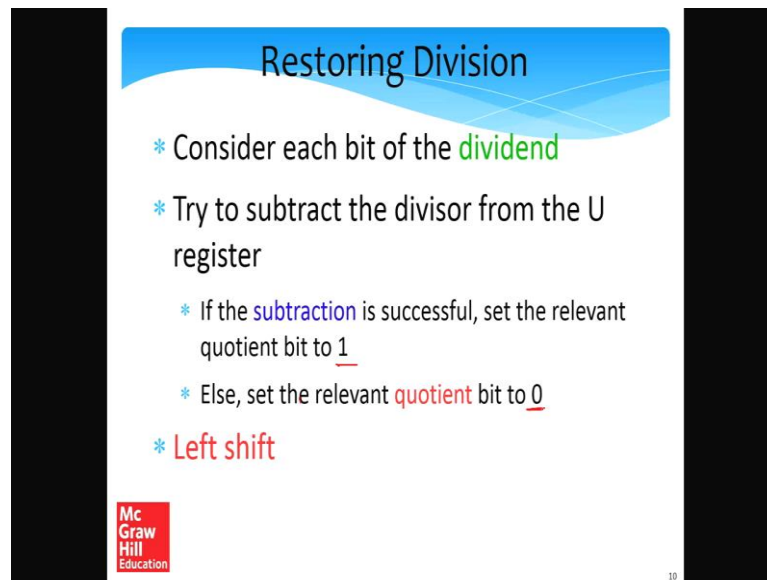
So, when an example is worth a thousand words. So, let us do it in a proper way. So, let us assume that we are trying to divide 7 by 3. So, 7 is a dividend. So, similar to multiplication to avoid overflow issues, we considered a 5 bit dividend. So, in this case N dividend is 0 0 1 1 1 and the divisor is D . So, how do we start algorithm. We started algorithm, by loading the dividend N register v , which is 0 1 1 1, and we load the divisor into the D register 0 0 1 1. Subsequently we will compute our results at 2 points; 1 is after the shift which is this point, and 1 is at the end of the iteration which is this point. So, after the shift 1 1 1 comes here, and x means that this in a bit we do not care.

Then what we do is we try to subtract the divisor from this number, and since we do not have anything, we write a 0 over here. So, since the result is negative, will be negative right, 0 minus any number is negative, we write a 0 here. Fine then after the shift we do a 1 bit shift so this numbers come here and the quotient bit that we have computed comes here. Again we try to subtract 3 from 1 we are not successful. So, the next quotient bit we compute to be 0.

Again we do a shift. So, this one comes here, this one comes here, this one comes here, and these 2 bits come over here, and one extra space is created where we need to write the next quotient bit. So, we see here the 3, the divisor D can be subtracted from 3. So, we go ahead and subtract, so the end of the iteration will have all 0s, and here since you able to successfully subtract, we will set the quotient bit to 1. So, what will have here is 1 0 0 1. Fine after that we do one more shift so this one comes here this one comes here. And since we cannot subtract 3 from 1; 1 will remain and 0 will come here, because we can do the subtraction.

So, the quotient becomes 2 and the remainder becomes 1 right, which is exactly what we needed to do, that 7 divided by 3 the quotient is 2, and the remainder is 1, and this is exactly what we compute right, the quotient 2 remainder 1. And this is a very simple algorithm. So, we just run N iteration, every iteration involves addition and subtraction, which are order $\log N$ time operations. So, the total time complexity is $N \log n$, N coming from N iteration, and $\log N$ coming from the fact that we have additions and subtractions in each iteration, and it takes $\log N$ time for doing an addition or subtraction.

(Refer Slide Time: 37:07)



Restoring Division

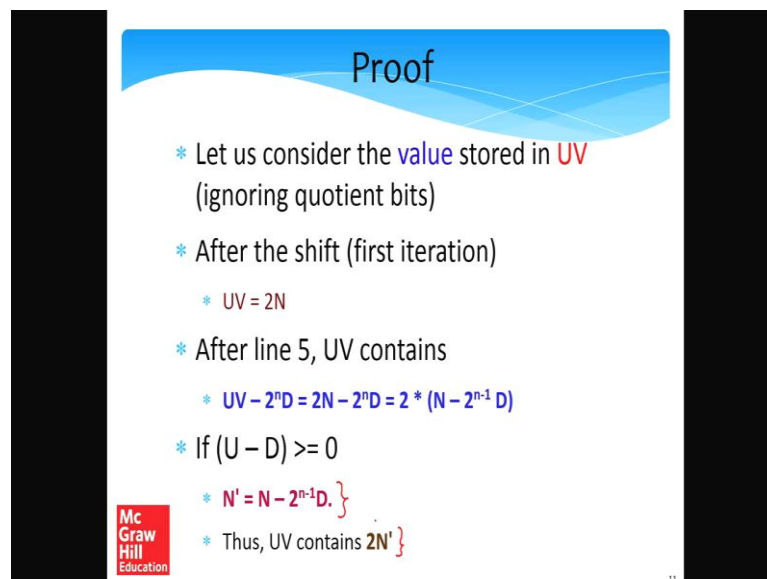
- * Consider each bit of the **dividend**
- * Try to subtract the divisor from the U register
 - * If the **subtraction** is successful, set the relevant quotient bit to 1
 - * Else, set the relevant **quotient** bit to 0
- * **Left shift**

Mc
Graw
Hill
Education

10

So, this is a textual description of our algorithm, we considered each bit of the dividend we try to subtract the divisor from the U register. If the subtraction is successful we set the relevant quotient bit to 1, otherwise we set the relevant quotient bit to 0 and then we left shift.

(Refer Slide Time: 37:28)



Proof

- * Let us consider the **value** stored in **UV** (ignoring quotient bits)
- * After the shift (first iteration)
 - * $UV = 2N$
- * After line 5, UV contains
 - * $UV - 2^n D = 2N - 2^n D = 2 * (N - 2^{n-1} D)$
- * If $(U - D) \geq 0$
 - * $N' = N - 2^{n-1} D$ }
 - * Thus, UV contains $2N'$ }

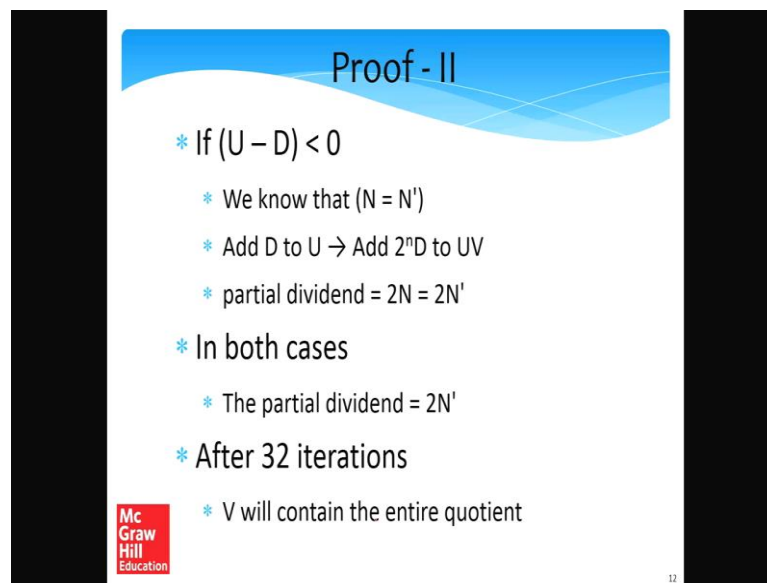
Mc
Graw
Hill
Education

11

So, this is a proof of you know what we discussed, and since you have already given a graphical proof, you know students can take a look at this I will not you know covered this, but in a students can take a look at this part of our description, and they will get an

idea, that you know pretty much what we are talking about and so this is more or like a formal proof, so we will find the formal proof in the book, but I just gave slightly semi formal graphical proof to, you know which can be further elaborated and written in the formal way, but I will not discuss that in the current presentation.

(Refer Slide Time: 38:13)



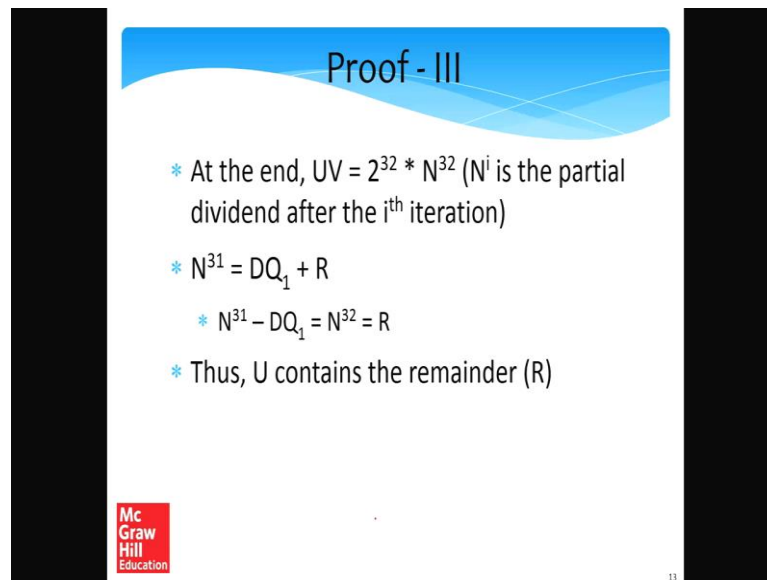
Proof - II

- * If $(U - D) < 0$
 - * We know that $(N = N')$
 - * Add D to U \rightarrow Add $2^n D$ to UV
 - * partial dividend = $2N = 2N'$
- * In both cases
 - * The partial dividend = $2N'$
- * After 32 iterations
 - * V will contain the entire quotient

McGraw Hill Education

12

(Refer Slide Time: 38:17)



Proof - III

- * At the end, $UV = 2^{32} * N^{32}$ (N^i is the partial dividend after the i^{th} iteration)
- * $N^{31} = DQ_1 + R$
 - * $N^{31} - DQ_1 = N^{32} = R$
- * Thus, U contains the remainder (R)

McGraw Hill Education

13

I am skipping the proof parts. So, part of the proof, you know at least graphical part we have covered the rest, you know I think the best strategy would be to read the proof from the book, and try to correlated was written over here.

(Refer Slide Time: 38:33)

Time Complexity

- * n iterations
 - * Each iteration takes $\log(n)$ time
 - * Total time: $(n \log(n))$

McGraw Hill Education

14

So, as I said it takes N iterations and each iteration takes $\log N$ time. So, total time is $N \log N$. So, I am just seen, have taken the o outs. So, its order of $N \log N$.

(Refer Slide Time: 38:48)

Restoring vs Non-Restoring Division

- * We need to **restore** the value of register U
 - * Requires an extra addition or a register move
- * Can we do without this ?
 - * Non Restoring Division

McGraw Hill Education

15

So, let us now introduce a new kind of division algorithm for non-restoring division. So, what did we see, we saw that we subtract D from u , and we need to restore the value of register U right. So, this is a problem, in the sense that we first subtract, you find that the result is negative, and then we need to add the divisor back. So, which means we restoring the value of registered U . So, this requires either an extra addition, or another

approach can be that will copy the registered U to another temporary location. We will then do a subtraction, and then move the values back, in any case with some of the work. So, can we just in a avoid this. To avoid this we do something called non-restoring division.

(Refer Slide Time: 39:44)

Algorithm 4: Non-restoring algorithm to divide two 32 bit numbers

Data: Divisor in D, Dividend in V, U=0

Result: U contains the remainder (lower 32 bits), and V contains the quotient

```

i ← 0
for i < 32 do
  i ← i + 1
  /* Left shift UV by 1 position */
  UV ← UV << 1
  if U ≥ 0 then
    U ← U - D
  else
    U ← U + D
  end
  if U ≥ 0 then
    q ← 1
  else
    q ← 0
  end
  /* Set the quotient bit */
  lsb of V ← q
end
if U < 0 then
  U ← U + D
end

```

McGraw Hill Education

16

So, in this case I will give the algorithm, the entire structure remains the same UV D, and everything remains the same. So, I will just give the algorithms. So, we do the same thing, we run a loop for 32 iterations and. So, this is the beginning of the for loop, and this is the end of the for loop. So, we do the same thing, we first do a left shift on the UV combine registered by one position right. Then what we do is as follows. So, so this is where you know there is a difference. See first check the sign of u, see if U is greater than 0, then we subtract the divisor D from it; otherwise if U is less than 0, which it can be because we are not restoring. So, we you know in the first iteration, even if we cannot subtract and meaning that the value of U will be less than 0, will still go ahead and subtract, and then in a subsequent iterations, the number can remain negative.

So, that is the reason any iteration you find U to be negative, we will add the value of the divisor D right. So, this is 1 if statement 1. So, what is saying is that we take a look at the sign of u, we subtract. If it is positive subtract the divisor, if it is negative we add the divisor. After that if U is positive, you know a positive or 0, we set the quotient bit to 1; otherwise we set the quotient bit to 0. And finally, the least significant bit of v, is set the

adding its 2s complement, and the. So, this is the 2, I can just. So, this is 1 1 1 0 0 plus 1; if this is a 2s complement of 3 so it is minus 3.

So, after this the sign of this number is negative, because the sign bit is 1. So, hence we set the quotient bit right Q_n to 0, Q_4 to 0 all right. So, after this we perform 1 more shift. So, we have 1 1 0 1. So, 1 1 0 1 get shifted to here, 1 get shifted here, and here we have 1 1 0, and we have 1 bit that will fill later. So, this number is negative. Since this number is negative what we do is, that instead of subtracting we actually add the divisor. So, we had 1 1 0 1 1 plus 1 1. So, 1 plus 1 is 0 1 plus 1 plus 1. So, 1 is the carry, 1 plus 1 plus 1 is 1 1 plus 0 is 1 1 1. So, we have 1 1 1 0. So, again this number is negative. So, the quotient bit that we set is 0.

Now, we do one more shift for the third iterations. So, 1 1 1 0 get shifted by 1 position. This one comes here, and 1 0 0 get shifted over here. This number is again negative, so we add 1 1 once again. So, 1 1 1 0 1 plus 1 1 is 1 plus 1 is 0 carries 1 1 plus 1 is 0 carries 1 1 plus 1 is 0 carries 1 1 plus 1 is 0 carries 1 and again 0. So, left it all 5 0s, and this number mind you is 0 is positive or 0. So, we will set the quotient bit to 1.

So, the relevant quotient bit is set to 1, because a variable Q in that algorithm is 1 fine we do one more shifts, so we set. So, one comes here 0 0 1 comes here and. So, since this number is positive, we will subtract the divisors. So, subtracting the divisor means adding its 2s complement which is 1 1 1 0 1 plus just 1, and so then this becomes 1 plus 1 is 0 carries 1 rights. So, this becomes what we need to have, and since this number is negative we set the relevant quotient bit to 0.

Now, the quotient is nicely saved in register v , which is 0 0 1 0, which is 2. So, the quotient is correctly computed, is 7 divided by 3, the quotient is 2 this is correct; however, we find that the register U . If you consider the last line, you have said if U is less than 0 then we add the divisor to U . So, in the last line here registered U is negative, because the sign bit is 1.

So, we need to add the value of U . So, we need to add U as U plus D . So, the way that we do is, we add 1 1 1 0 plus the divisor. So, this is 0 plus 1 is 1 1 plus 1 is again 0 carrier 1, so the rest will be all 0s. So, that is the reason we see 0 0 1 here. So, 0 0 1 is the remainder rights. So, the remainder is 1 and the quotient is 2. So, in this case it is not that we change the, you know ultimate asymptotic complexity of the algorithm, we did not do

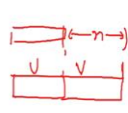
that. So, we still have a N iterations. So, the time required is order n, it still have a N iterations right, and N each iteration at least have a subtraction or an addition, so that is log N time. So, we still required order N log n, but it just that we want restores the value of U. So, we either do an addition or we do a subtraction that is it right, we do not do anything more.


So, that is the reason this is called a non-restoring algorithm, which is again the standard and very popular algorithm. And the non-restoring algorithm is regarded as a standard. So, it is used in many many processors for division, and algorithm is again in just to summarize. We take a look at the register U is positive is subtract the divisors; else if it is negative we add the divisor. We take a look at the sign once again, so if the sign has become, if the sign bit is 0 which means a number is non-negative. We set the relevant quotient bit to 1; otherwise we set it to 0, which perform a small check at the end if the registers U are still less than 0 we add the divisor to u, and U contains the remainder, and register v contains the quotient.

(Refer Slide Time: 48:55)

Idea of the Proof

- * Start from the beginning : If $(U - D) \geq 0$
 - * Both the algorithms (restoring and non-restoring) produce the same result, and have the same state →
- * If $(U - D) < 0$
 - * We have a divergence ↻
 - * In the restoring algorithm
 - * value(UV) = A Ⓢ
 - * In the non-restoring algorithm
 - * value(UV) = $A - 2^n D$ Ⓢ




18

So, now let me give an idea a very very brief idea of the proof, but the detail proof is there in the book. So, is not really you know the proof is slightly involved is also. So, that is the reason it is kind of difficult to explain it while presentation, but let me nevertheless try.

quotient bit is 1, then we will subtract $2^n D$ to the power n times D in the restoring algorithm, to get the value of UV , discounting the quotient bits; of course, as $2^n D$ raise to the power n times d .

In the non-restoring algorithm, we will add $2^n D$ to the power n times D . So, what we will get is $2^n D$. So, this is what we began with, $2^n D$ plus 1 times D . Since we are negative will add $2^n D$ to the power n times D . So, what we will get is, we will get this. So, as we can see this is equal to this, which means that whenever we set the quotient bit to 1, the state of the restoring and non-restoring algorithms becomes the same. So, since the restoring algorithm computes the right results, we also compute the same set of results, in terms of both the quotient as well as the remainder.

(Refer Slide Time: 52:24)

Proof - III

- * If the quotient bit is 0
 - * Restoring
 - * partial dividend = $2^n A$
 - * Non restoring
 - * partial dividend = $2^n A - 2^n D$
 - * Next iteration (if quotient bit = 1) (after shift)
 - * Restoring : partial dividend : $4A$
 - * Non restoring : partial dividend : $4A - 2^{n+1}D$
- * Keep applying the same logic q ← 1
✓

McGraw Hill Education 20

So, mind is my proof is slightly in hand wave in nature. The main reason being that I the book has a more detailed proof, and in a presentation is just my aim is to give a very very high level overview. So, just you know containing in an overview style, whenever we set the quotient bit to 1 we will find that the value in the UV register, discounting the quotient bits between the restoring and non-restoring will be the same. Otherwise the quotient will also keeps it, I am sorry, the restoring algorithm will continue to set the quotient bits is 0, and a non-restoring algorithm will continue to also set the quotient bits is 0, and whenever the quotient bit is set to 1, both the algorithms will have the same state, in the sense all the registers will other the same set of values.

Now let us say at some points, you know quotient bit is 0, then the partial dividend N dash, in the case of restoring is 2 a, and in the case of non-restoring is another value. So, like that we can keep going, but we can easily prove, that whenever the quotient bit is set to 1, the states of both the restoring and the non-restoring algorithms become the same.

(Refer Slide Time: 53:51)

Outline

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

21

So, I explained 2 kinds of algorithms restoring and non-restoring, and with the help of this example, where the slide has to be cleaned up, also with the help of this example.

(Refer Slide Time: 54:08)

Dividend (N) 00111
 Divisor (D) 0011

beginning:	00000	0111
1 after shift:	00000	111X
end of iteration:	11101	1110
2 after shift:	11011	110X
end of iteration:	11110	1100
3 after shift:	11101	100X
end of iteration:	00000	1001
4 after shift:	00001	001X
end of iteration:	11110	0010
end (U=U+D):	0001	0010
Quotient(Q)	0010	
Remainder(R)	0001	

Restoring [- +]
 Non-restoring
 [- or +]
 $\begin{matrix} \leftarrow q_{i-1} \\ \leftarrow q_i \\ \leftarrow q_{i+1} \\ \leftarrow q_{i+2} \end{matrix}$

17

We explain how to divide 7 by 3 and compute the quotient and the remainder. So, in this context 2 algorithms were discussed; restoring and non-restoring. So, in the restoring algorithm 1 subtraction needs to be done in each iteration, and 1 addition. Alternate versions are possible, but still a fair amount of work needs to be done right. So essentially here one subtraction and one addition need to be done, but in a non-restoring version, either you do a subtraction or you do an addition, but not both.

So, in that sense the work is roughly half; that is the reason the non-restoring algorithm is used, and the way that you prove it works is that you start. So, the first state is the same, after that what we need to prove is that every quotient bit is computed correctly, and whenever a quotient bit is computed to be 1, the state of both the algorithms is the same, you know the values of all the registers is the same.

So, pretty much we can say that. Let us see that we compute the quotient bit to be 1 right maybe small q . After this as long as the bits are being computed 0 the algorithms will take to separate paths, in the entire region the quotient bit is being computed to be 0. Finally, when we again compute a quotient bit to be 1 the states merge, then again if the quotient bits is 0 the states diverge. Again the next time that the quotient bit is 1 the states will merge and be the same.

So, that is the way that we can proof a detailed proof is there in the book, but in any case the summary is that the non-restoring algorithm is faster than the restoring variant. So, now we shall take a look at floating point addition.