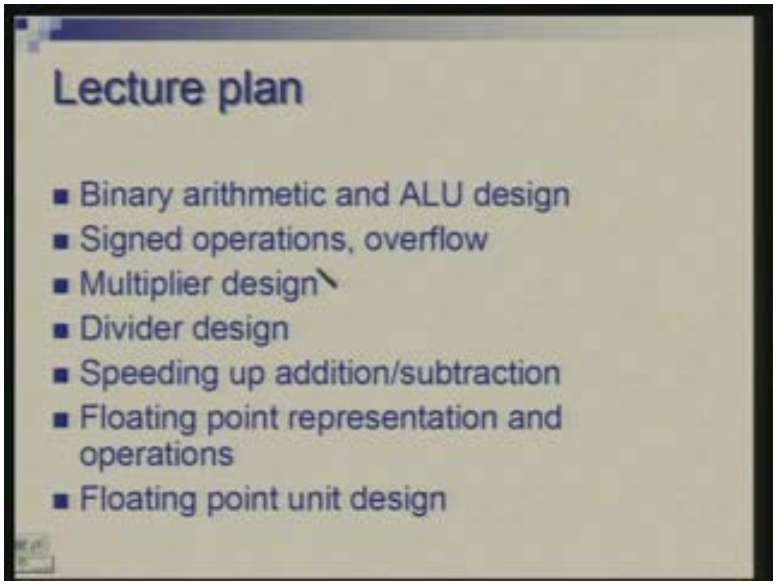
Computer Architecture
Prof. Anshul Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture - 13
Multiplier Design

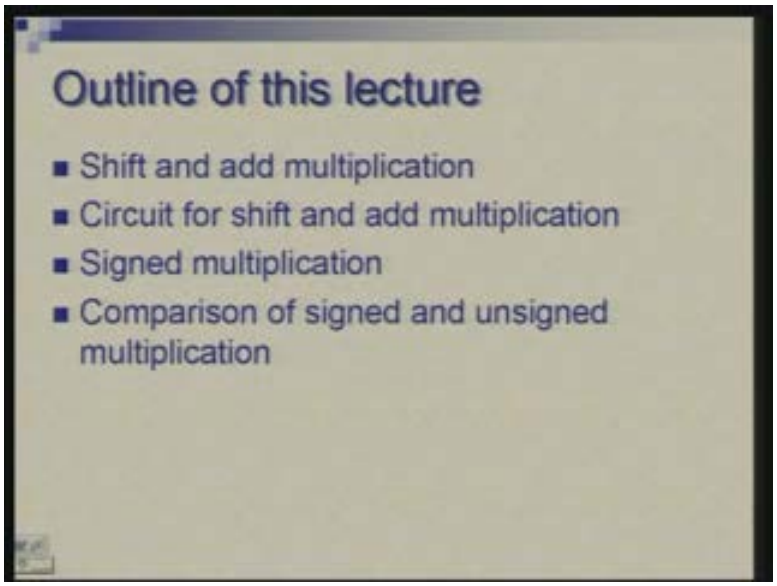
We have discussed the design of ALU with respect to addition, subtraction, comparison and logical operation. We also discussed other operations like shift. We will continue on this and take a slightly more complex operation namely multiplier multiplication. So we will see how multiplier can be designed both for signed case and unsigned case. And in the next lecture we will move to another operation which is complex that is division. In the plan of lecture we have reached this point, talk of multiplier design.

(Refer Slide Time: 01:31 min)



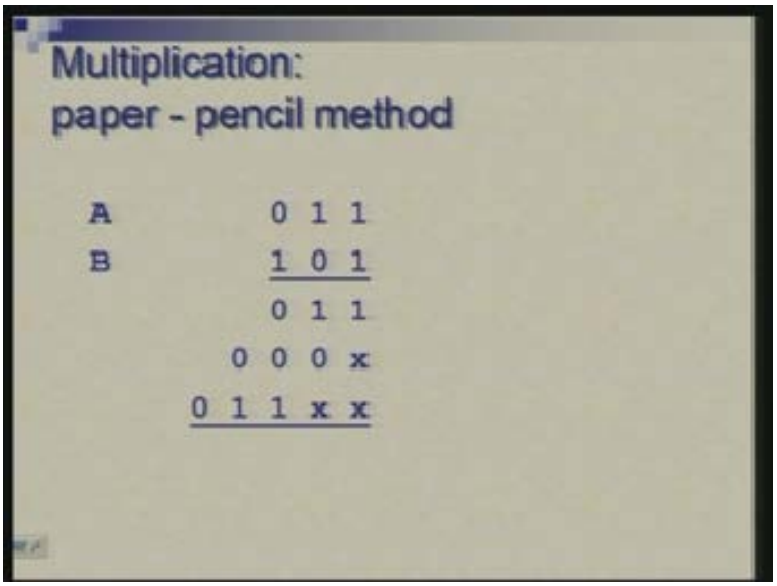
We will begin with a very simple design shift and add multiplication. We will see how you could build the circuit following this shift and add approach. We will see multiple ways which differ in terms of cost of hardware or simplicity of hardware. Then we will talk about signed multiplication where you can either, do unsigned multiplication then take care of the sign separately or do directly signed multiplication. We will see what are the essential differences in the circuits which are required for signed and unsigned multiplication.

(Refer Slide Time: 02:16 min)



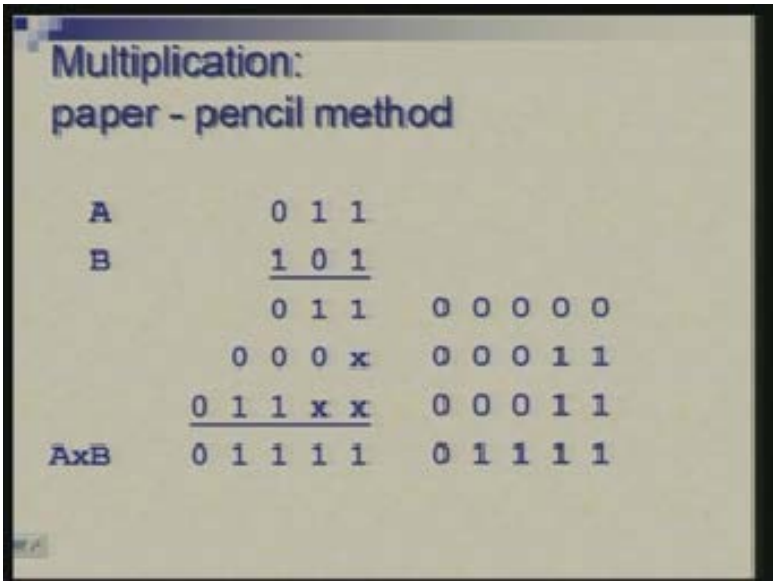
So let us begin with multiplication as we do with paper and pencil. It is a simple multiplication method as we are all used to…. the only thing is that we do in decimal system and the same thing translated to binary in fact is even simpler as you would notice. Suppose you have a 3 bit number A which has to be multiplied by another 3 bit number B so as usual we will multiply A by each bit of B with suitable weightage and simply add them. So multiplied by the LSB of B we get 011, multiply A with middle bit of B you get all 0s, multiply with left most bit of B you get 011so basically individual multiplication is multiplication with either 0 or 1 which is very straightforward.

(Refer Slide Time: 3:24)



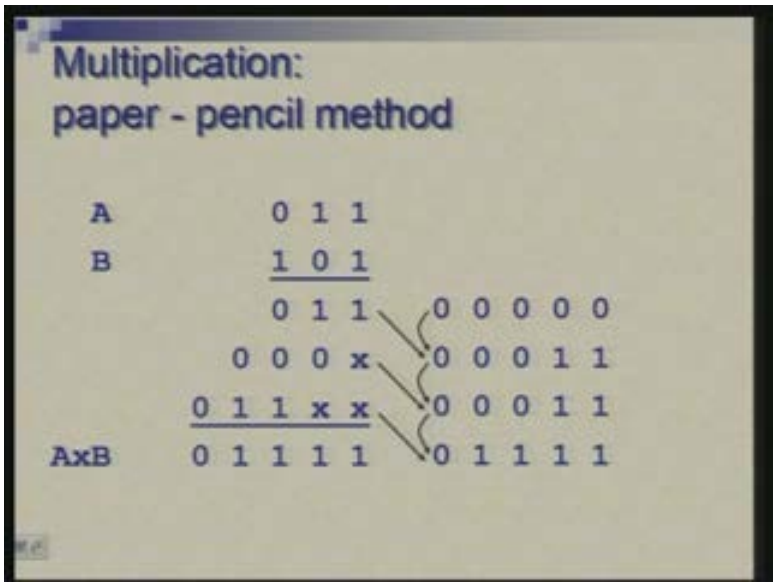
Multiplication with 0 results in 0 and multiplication with 1 results in the same multiplicand itself and these partial products; each line represented here is a partial product they have to be weighted appropriately and the weightage here is power of 2 which means shifting it appropriately towards left and simply add all these. That is a simple way which we will try to capture hardware and the way hardware work is that you will start with zero value in some register; you can add first partial product to that you get this, add second partial product in this case of course there is no change, add the next partial product and you get the final result.

(Refer Slide Time: 4:17)



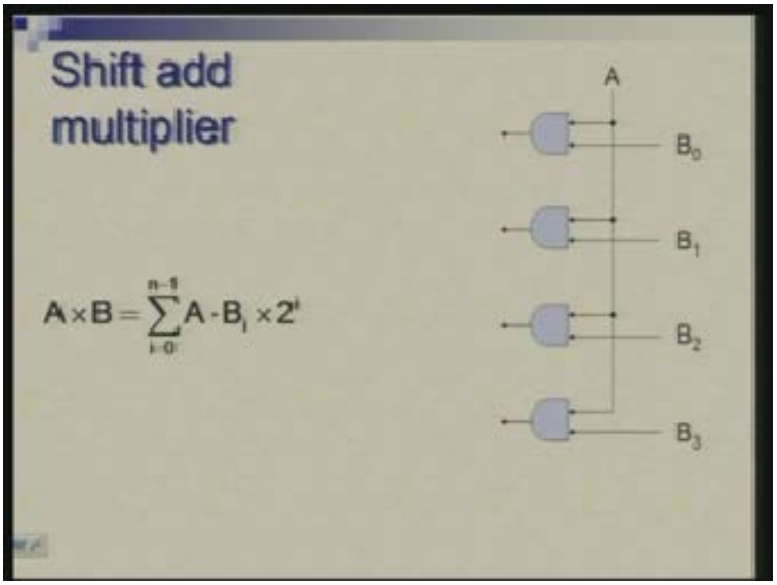
So basically you are carrying out addition of initial zero value and this partial product to get this. Then next one is added and the next one is added. So, as you can see, the whole thing breaks up into multiplication by 0 and 1 shifting by successively 1 2 3 4 so many positions and adding all these up. So we have already seen each one of these individually. we know how to add, we know how to shift and of course all we need is to multiply with 0, 1 which means performing essentially an AND operation. AND operation actually is equivalent to multiplication by 1 bit.

(Refer Slide Time: 05:05 min)



So we can express this as a summation where A is multiplied by bits of B that is B i with a weightage of 2 raised to the power i and we sum it over all values of i going from 0 to n minus 1; this is an unsigned multiplication.

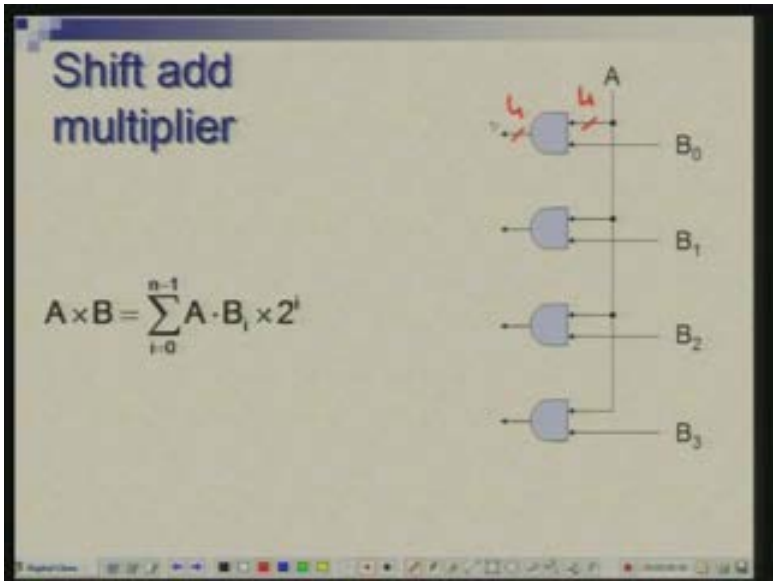
(Refer Slide Time: 5:32)



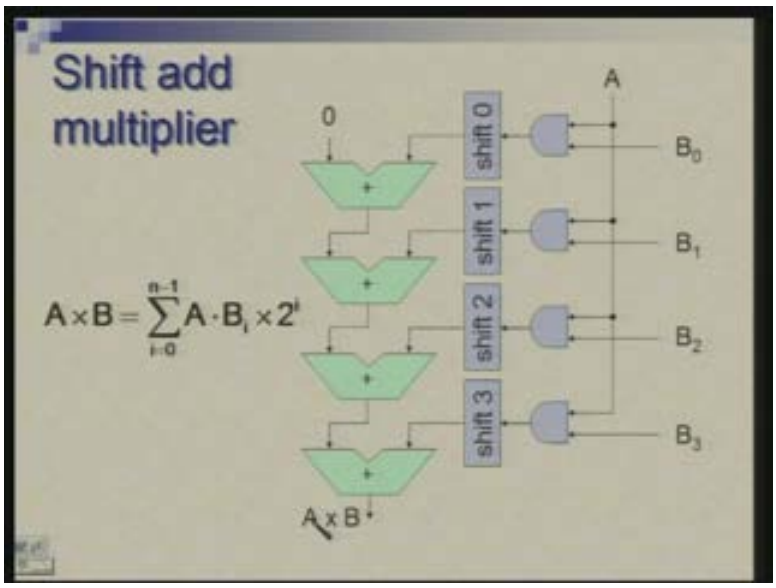
And the circuit for doing this can be easily built. First thing we are trying to do is multiplication with a 0 or 1 so it is effectively A is multiplied by one of the bits of B so here I am illustrating with a four bit situation. This AND gate actually represents an array of AND gates where each bit of A is ANDed with B 0. So, in […..6:09] I am showing a single AND gate which is where we are taking a vector of bits A and a single bit B and output is a vector. So, strictly speaking I

should have marked the size of the vector going into or out of it. So each of these AND gates is effectively an array of four AND gates. So what I am getting here add these points is the partial product A is multiplied by one bit of B. And then I require shifters so each of this shifter is doing shifting by a fixed amount so it is only a there is no active logic here it is only appropriately wiring it; these shifters do not have to select between shift and no shift; it is a fixed shift with each is doing so it is simply a matter of wiring it appropriately so that effectively shift takes place.

(Refer Slide Time: 06:38 min)



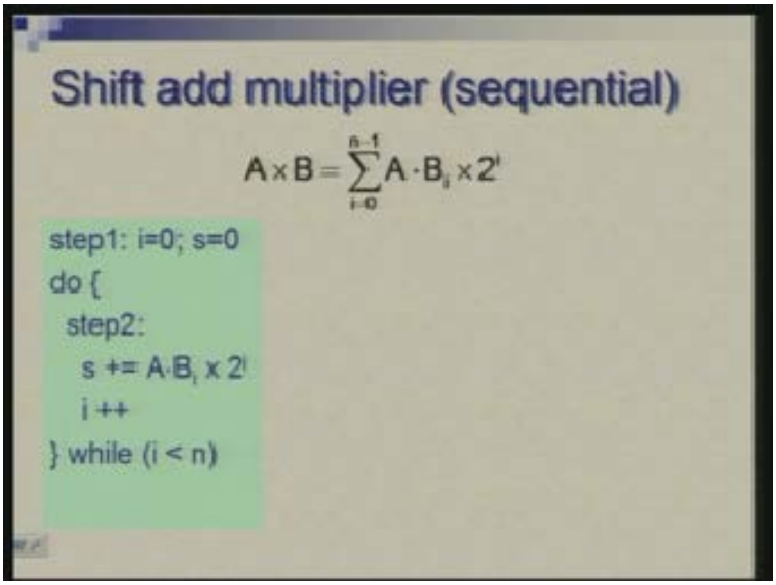
(Refer Slide Time: 07:50 min)



The next stage is addition. So, starting with a 0 here we add first partial product, then second partial product, third and fourth and finally we have a product of A and B. So when I say that these shifters are simply wiring it means that these four bits are connected at appropriate points at the input of adder. You connect directly or shift by one position or shift by two positions or shift by three positions as the case is. This is one very simple way of capturing this idea of summing the partial products to get an unsigned multiplication. Is any question about this?

Now this requires n adder. If you have n by n multiplication to be carried out this will require n adder each is adding one partial product. We can simplify the circuit by using same adder several times. So we can capture this in the form of an iterative algorithm so I am keeping a count i a sum S is initialized to 0 and then I do something repeatedly, there is a loop where A into B i multiplied by 2 raised to the power i is accumulated i is updated and I repeat it for all values of i.

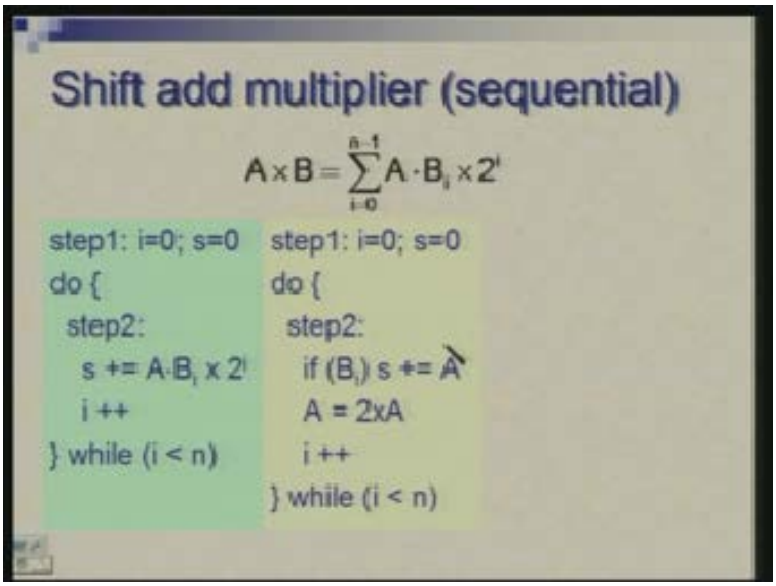
(Refer Slide Time: 9:37)



So what I am trying to indicate here is that there is an initialization step which is step one and step two which is repeated n times. Step two is both these assignments so actually what I imply is that all these both these are done together in the same clock cycle. So, for n-bit multiplication this would be done this loop will be done in n clock cycles; although I have written as two separate assignments but my intention is to do these two in hardware in a single cycle.

Now there could be some improvements made in this. Instead of adding A multiplied by 2 raised to the power i what I could do is I could actually keep on shifting A itself, keep on modifying A so for every cycle A will get doubled and then I need to add A only.

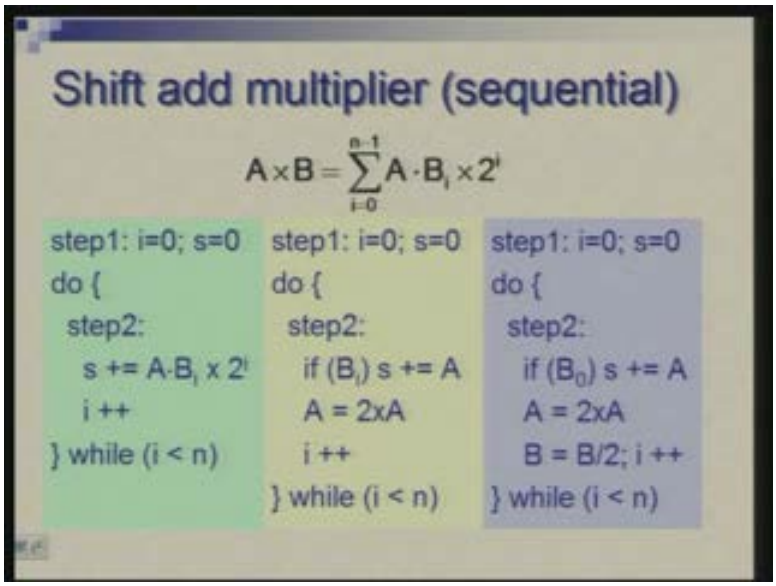
(Refer Slide Time: 10:49)



Let us see the algorithm with this modification. Instead of saying s accumulates A into B i into 2 raised to the power i…. this I am putting as a condition (Refer Slide Time: 11:04) that if B i is 1 then I do this accumulation otherwise I do not do this addition and in any case whether si whether B i is 1 or 0 A is doubled every time. So I could say A is doubled or shifted left it is the same thing. And all these three activities are done in a single step.

As I add A to s, I prepare the next value for A, for the next cycle in the same step. So this whole thing is one clock cycle as far as hardware is concerned: updating i, doubling of A and conditionally accumulating of A. So basically shifting of A here is taking care of taking care of this multiplication by 2 raised to the power i. Since I am doing it sequentially I can keep on incrementally shifting it every time.

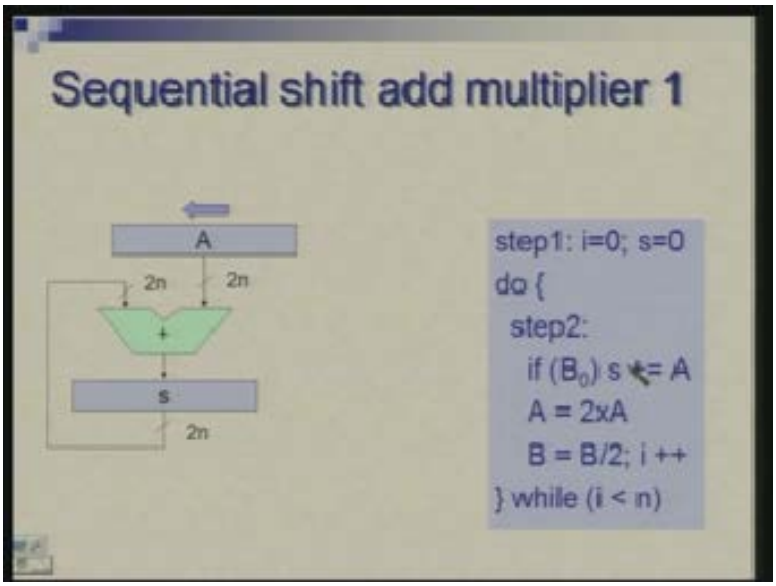
(Refer Slide Time: 12:07 min)



The next modification here is that instead of looking at different bits of B in a successive iteration I can keep on shifting B in a register so that I always look at B 0 that will also simplify the hardware to some extent. Rather than trying to look different bits in different cycles I look at the same bit but move the bits in such a manner that I need to focus at the same point always. so here (Refer Slide Time: 12:44) I am checking B 0 always but to compensate for that I am making B as B by 2 which means B is right shifted every time to get the same effect.

Now let us take this and look at its hardware equivalent. It is the same algorithm. What I require is essentially a mechanism to add A to s. So here is an adder, A is one input, s is another input; I am not showing how I am making s as 0 initially so that detail is limited but this will take care of performing this type of S gates S plus…….

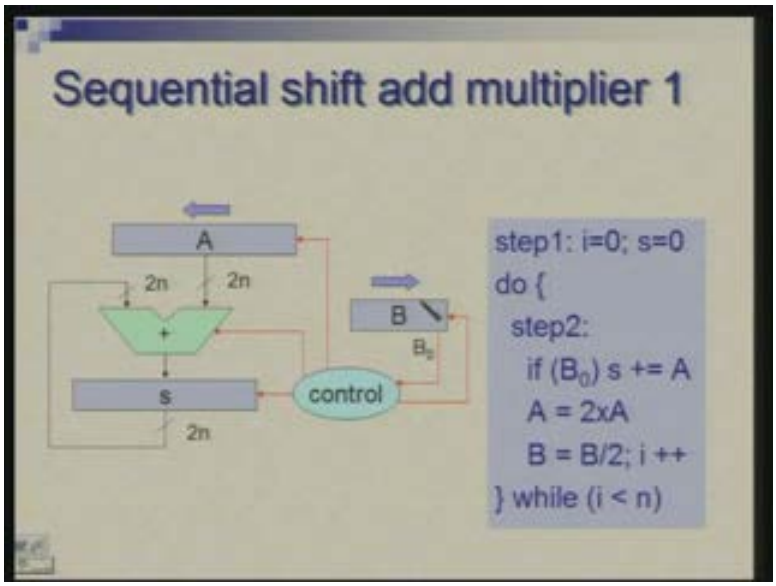
(Refer Slide Time: 13:33)



And to take care of this, A will be shifted left after A supplies, while A forms an input to this adder A will also make itself ready for the next cycle. So at the edge of the clock S plus A is a stored into this and 2A is stored into A. So these two events will happen concurrently with the edge of the clock.

Now one thing you should notice here is that this adder has to add two n-bit numbers. When you are talking of two n-bit A and B both are n bits but as the partial products are effectively shifted to left we need adder of double the size. First addition would be done at one extreme position and the last addition would be done at the other extreme position so adder has to be wide enough or essentially double the width to accumulate all these values. And this register which will hold A also is of double the length so the operand initially is placed in the right half in the right half the value is placed and it keeps on shifting to the left and by the end it has reached the other end. So this is the key part of the circuit and to control this we require another register holding B which will experience a right shift every in every iteration we shift B so that we are always looking at B 0 position which is the LSB of this.

(Refer Slide Time: 15:30)

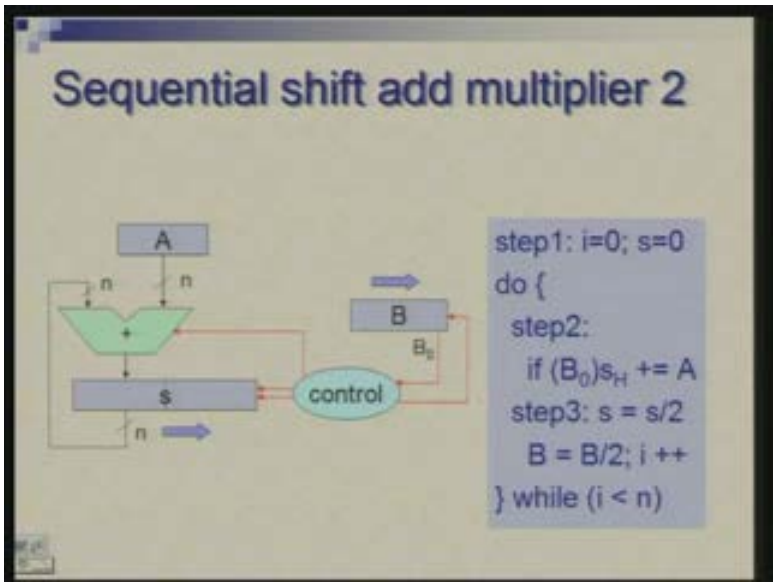


This has to work in conjunction with control which firstly takes care of this iteration count. The process has to be repeated several times so there it does the counting and it also times the operation of all other units. I am showing these red signals which are controlled signals; this is working under control of B 0.

Depending upon the value B 0 it will actually instruct addition to be done or either this circuit will pass on the value of A here sorry A plus S or just S itself. There is some logic here which either performs the S plus A or S plus 0 so that control is here and this is…… this, this and this (Refer Slide Time: 16:24) are basically to time the operation of these three registers when they shift and when they store in any value. Therefore, this is an essence of the circuit. There are some details which are omitted here. But what I would like you to notice here is how we are going from step to step and what is each crucial step here. Each step basically involves these things checking B 0, performing addition conditionally, left shift of A and right shift of B and update of a counter so that counter I am not showing it explicitly but it is part of the control.

Now let us do something so that the requirement of two n-bit addition can be cut down. We can mange with n-bit addition then something here will be simplified. And the idea of this comes from….. let me get back to this diagram, this one (Refer Slide Time: 17:33). So you would notice that at any time one of the value which you are adding is only n-bit and it is only when you look at the whole thing it is 2 n-bit wide but if you look at each addition you are adding one n-bit value in some position. So if you focus on that you need to have only n-bit adder; there is nothing changing on the right of it and there is nothing changing on the left of it. So, if you arrange your information such that you are taking care of those n bits where the new value is positioned you can work with n-bit adder.

(Refer Slide Time: 18:35)



Here is the modified circuit where what we are trying to do basically is that we are adding A to the left half of S. SH is denoting the high end of S or the upper half of S. So A is always added in that position and you are making sure that the two are correctly aligned so what we will do is we will be shifting S to right every time. initially S contains 0, you add that to the left half of S then it is shifted as I shift it to right then you add A to…… again it is the same position so it is the partial product which we have accumulated that keeps on shifting to the right. instead of shifting A to the left in every cycle we are shifting the partial product obtained so far to right which actually maintains the same reality position and achieves the same effect and the consequence of that is the adder needs to be only an n-bit adder. So A is now n-bit register A does not shift at all; the value A gets added to the left half of S and after addition you can shift S to right.

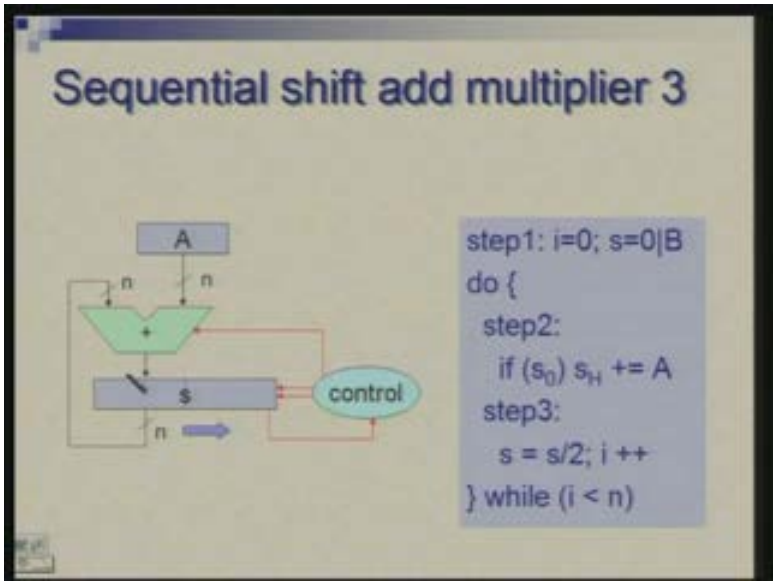
Therefore, I have introduced another step here that after this addition S becomes S by 2 which means you are shifting it to the right and the rest of it is same. B has been shifting right the counter increments and so on. So the crucial change is here that the addition is only to the left half of S and S shifts right. Although I have introduced this as another step but it is actually possible in principle to do all this in a single clock; I will not going into the details; you can just take my word for it for the moment. But it is possible that the final value which has to be there in S as a result of summation and shifting. You can actually look at that and place that directly in a single step. That is not very difficult. But just for conceptual clarity we assume that first A is added and then shift is done.

There is one more subtle point here is that when you are shifting S right there is a carry which will come out of this which might come out of this that has to enter so it is not a pure simple right shift with the zero getting stuffed into the vacant position; it is this carry which you do not want to lose it is an intermediate carry which has to be accommodated here. So apart from that subtle consideration it is very straightforward. What we have achieved is we have reduced the size of this register by a factor of 2, we have reduced the size of adder by a factor of 2. And having done this there is another interesting observation which can be made. That is, we have

now two registers S and B which are shifting right so initially S has all 0s and as you perform addition and shift right you keep the bits keep trickling into the right half of S but initially everything is zero so as partial product gets added, one by one, the bits keep on entering the right half of S. At the same time B is this register is filled with the value B and as it shifts right it keeps on making spaces on the left side. So this observation leads to the possibility of combining S and B in the same register.

What I am saying is that initially right half of S is vacant and it is getting filled up gradually bit by bit from the middle whereas B initially is full and getting vacated bit by bit on the left side so the two actually match so the two together never require more than 2 n bits. So what we can do is we can actually initialize as with 0 in the left half and B in the right half and then look at the right most bit of S instead of right most of bit B and the rest remain the same.

(Refer Slide Time: 23:19)



The initialization of S is different. We have 0 and B put together performing to n-bit word which is put in S
and, yeah, here shifting of B is avoided there is no B register now so s become s by 2 which is a right shift and rest of it remains same. This is the final circuit which we will I will leave at. It requires one 2 n-bit registers one 1-bit register and a single n-bit adder.

Any questions about this?
In MIPS we have multiplying instruction which actually takes two operand in two registers and it produces a 2 n-bit results which are kept in two special registers: one is called high and one is called low; Hi and Lo. So there are special instructions which can be used to move data between these registers and one of those thirty two general purpose registers. But the output of multiplication goes to those specific registers high and low.

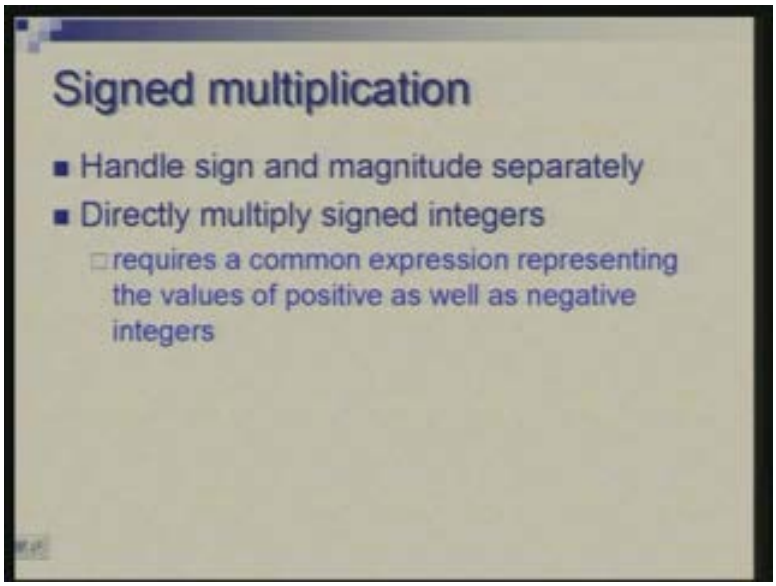
Actually there is also pseudo instruction for multiplication which takes three registers so two for operand and one for result and that actually expects small numbers so that the product is within 6

within 32 bits and it ignores Hi and only Lo is looked at and automatically brought into one of the registers.

Now let us move towards sign multiplication. There are two approaches to this: one is we handle sign in magnitude separately. That means if there two signed numbers we find their magnitudes, multiply them, get magnitude of the product and determine the sign. So sign, you know, if the two numbers are of the opposite sign then the sign is negative they are of same sign, sign is positive and must we know the sign of the product you can put it back in the appropriate form the 2's compliment or whatever approach you have.

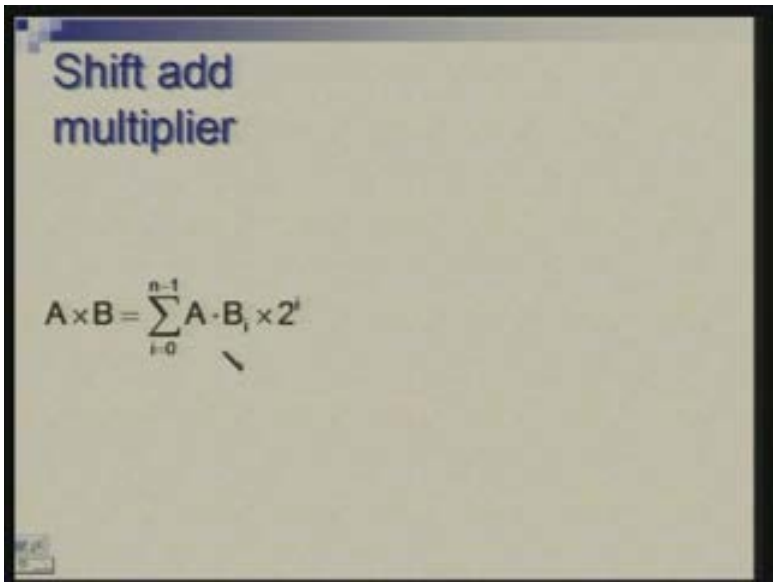
The second approach is to directly multiply sign integers. So, unlike addition and subtraction where this 2's compliment representation made it possible to look at signed and unsigned addition subtraction identically except for overflow detections but the addition subtraction process was oblivious of whether there is a sign or not. For multiplication that does not exist. So we have to device a method which can directly multiply signed numbers and what that requires is a common expression representing the values of positive as well as negative integers.

(Refer Slide Time: 26:32 min)



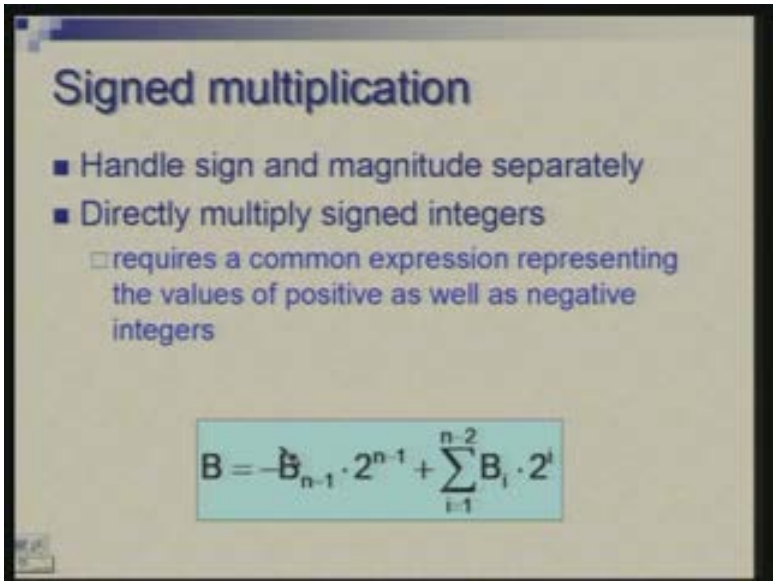
We expressed the number in terms of its bits in this form:

(Refer Slide Time: 26:48 min)



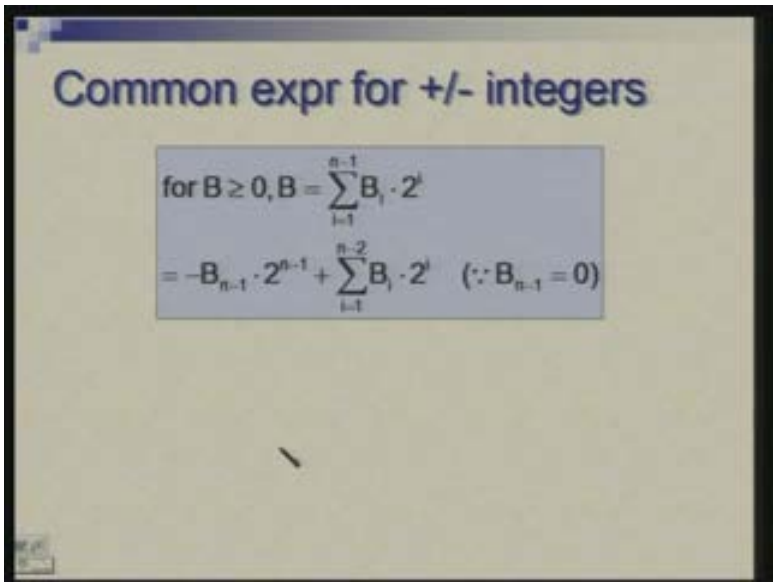
Here you see, basically we have expressed B as a summation of B i into 2 i for different values of i. We need to find something which handles positive as well as negative numbers. Again representation we will use 2's compliment and this representation is shown here. So you would notice that there is similar kind of summation but it goes to bit n minus 2 only. The last bit which is a sign bit is handled separately so all that we have done is put a negative sign with this. So again the weightage is 2 raised to the power n minus 1 here but only difference is that this comes with the negative sign.

(Refer Slide Time: 27:16 min)



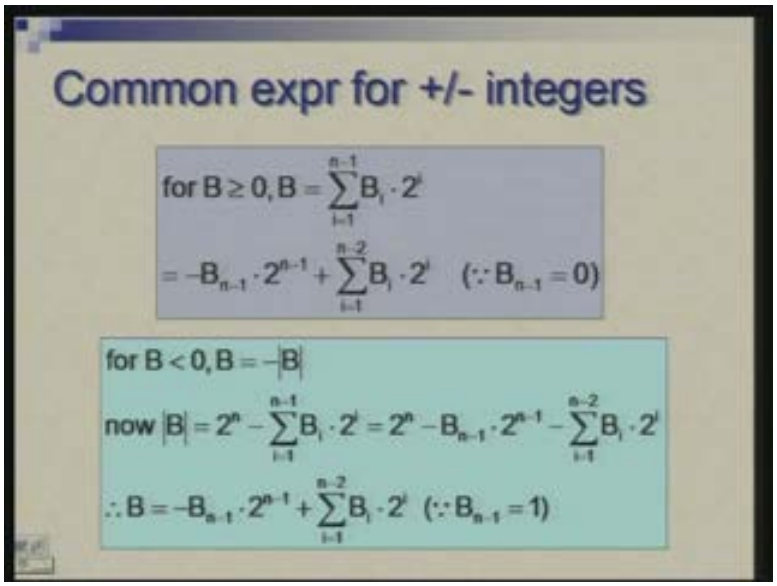
Why is it so; we can see it here.

(Refer Slide Time: 27:53)



So let us look at two cases. What happens when the number is non-negative and what happens when the number is negative. So, if B is non-negative we could have expressed this in this form that is the usual thing for unsigned number. But since B n minus 1 is 0 because this number is non-negative the sign bit is 0 therefore if we pull out the last term and put a different sign it does not matter this term is 0 in any case. With B n minus 1 is 0 whether you put plus or minus it does not matter because this part is 0 (Refer Slide Time: 28:39) so a positive a non-negative integer B can be expressed in this form so this part is…. this case is very straightforward. The other cases are that of negative numbers which needs a little bit of analyses.

(Refer Slide Time: 28:54)



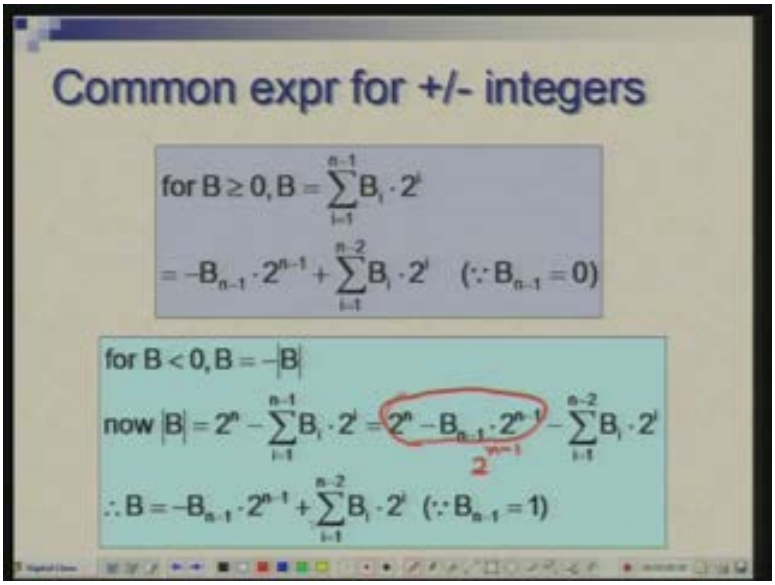
When B is negative its value is basically minus magnitude of B. So this represents; this is the magnitude of B and the prefix with minus sign is what B is all about.

Now let us see what the magnitude of B would be. Since it is a negative number we can express we can find the magnitude by knowing that it is a 2's compliment representation. So 2's compliment representation means 2 raised to the power n minus weighted sum of the bits. So, this is the interpretation of that number had it been an unsigned integer. So, to find its equivalent magnitude, retain this as a negative number, we take that value and subtract it out of 2 raised to the power n. So this is by definition of 2's compliment representation that you subtract this from 2 raised to the power n.

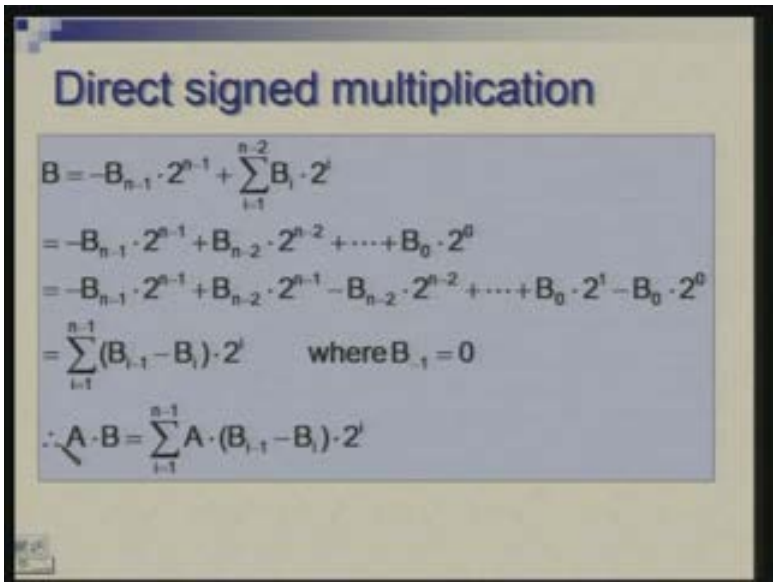
Now this summation (Refer Slide Time: 30:08) can be broken up, we take n minus 2 terms keep them inside summation and bring out the one which corresponds to sign bits. Now, B n minus 1 is 1 in this case so this thing is nothing but 2 raised to the power n minus 2 raised to the power n minus 1. Since 2 raised to the power n is nothing but two times 2 raised to the power n minus 1 this difference will correspond to…… let me write it here ….this whole thing outside the summation is equivalent to 2 raised to the power n minus 1 I am sorry 1 in the subseq[uent]…… so this whole thing (Refer Slide Time: 31:19) is equivalent to this which I have rewritten bringing B n minus 1 back because that is this is equal to 1 so I can write this multiplied by B n minus 1 which is 1 to get in this form. So this thing substituted here I get minus B n minus 1 2 raised to the power n minus 1 and that becomes positive so I get this.

I have shown that both positive or negative numbers when they are expressed in 2's compliment form can be captured by a single expression. Now we have an expression which ignores whether it is a positive or negative number and we can multiply the two together.

(Refer Slide Time: 31:15 min)



(Refer Slide Time: 32:22)



So, direct signed multiplication is basically A multiplied by this expression. But I work on this little bit more, on this expression to get to a convenient form. First of all let me explain this. What I get is the first term is a negative sign and all other terms are in positive sign.

Now all other terms are broken into two parts. For example, this B n minus 2 into 2 raised to the power n minus 2 is written as this (Refer Slide Time: 33:03) sum of these two terms whereas effectively I have put a factor which is double of that. If you take B n minus you can actually verified it by working by yourself. Take B n minus 2 common out of this you get 2 raised to the power n minus 1 minus 2 raised to the power n minus 2 this is this actually is twice this so what you will get is….. so if I break each of these terms like this I can then combine the term with same power of 2.

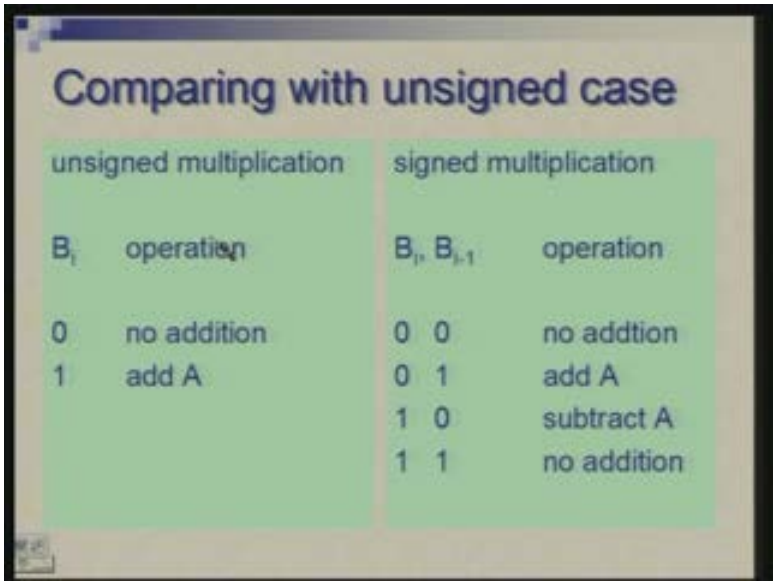
So, for example, here these two (Refer Slide Time: 33:42) have the same multiplying factor 2 raised to the power n minus 1 what I get is B n minus 2 minus n minus 1. Similarly, this term will combine with the next term involving B n minus 3 and so on so what i get is this summation of B i minus 1 minus B i weighted with 2 raised to the power i and summed over i goes from……. I think this should have been 0, yeah. So this is 0 but to facilitate writing in this particular manner I have introduced a dummy B minus 1; this goes up to B minus 2 but otherwise this term will remain unpaired; to combine it with another term I have introduced a zero term which is actually 0 by definition. Assuming that B minus 1 is 0 by definition I can reduce the last term here so that every term gets paired and then this formula can be written uniformly.

Therefore, given this representation of B you can write A multiplied by B as this summation; it is just that in this I bring in A within summation. So now what does it require; it requires partial products to be formed according to this (Refer Slide Time: 35:21). Instead of multiplying A with B i which is 0, 1 I am multiplying A with this term and this can have what are the values it can have it can have a 0 value, plus 1 or minus 1 so still things are not too difficult. Minus 1 would

mean that I take actually minus A. So this partial product would be A, 0 or minus A and adding minus A means basically subtracting A. So i have something which is very similar to what we have done; there are just a few differences which I will enumerate. Instantly this is called Booth's algorithm.

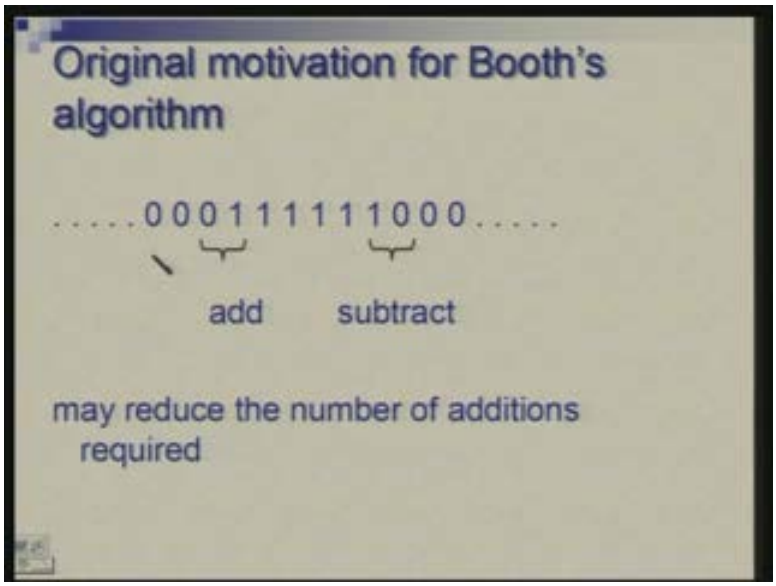
So let us compare unsigned multiplication, signed multiplication here. In unsigned multiplication what we saw earlier we looked at B i which can be either 0 or 1 and accordingly we either perform no addition of perform addition of A whereas in signed multiplication we looked at two bits together B i and B i minus 1. Actually we are looking at this minus that which can have values 0, 1 or minus 1. So, when both are 0 we have no addition nothing to be done no addition no subtraction; when this is 0 and this is 1 (Refer Slide Time: 36:52) since we are doing B i minus 1 minus B i we need addition of A. When this is 1 this is 0 we subtract A and when both are 1 again we require no addition. So, with this small change introduced all the circuits which we had could be made to work for signed multiplication. The only thing we need to remember is that what we had shown as adder needs to be capable of performing addition and subtraction. And there has to be little logic which looks at the pattern of B i and B m it looks at two bits and decides instructs that circuit to perform either no addition or addition or subtraction.

(Refer Slide Time: 36:10 min)



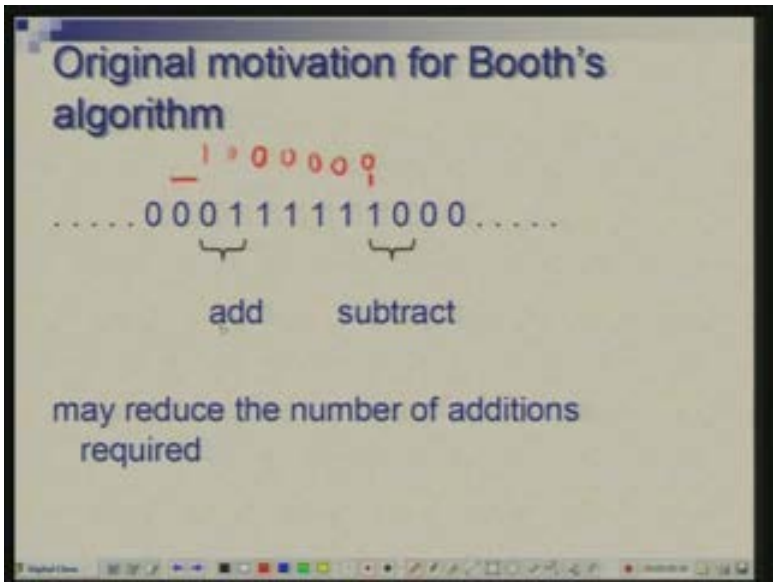
Well, this algorithm which I mentioned as Booth's algorithm was originally devised not for signed multiplication; its motivation originally was something different which is as follows that if you have…….. let us look at the multiplier B. If you have a row of 1s as per the logic we are doing something when there is change from 0 to 1 or 1 to 0. Let me go back to this one. See, when both are 0 both are 1 there is nothing required; when you are going from 1 to 0 there is an addition, when you are going from 0 to 1 there is a subtraction. So let us say this is part of the word comprising B and you are scanning the bits from LSB to MSB. So when you have a run of 0s there is nothing required; when you have a change from 0 to 1 then you perform subtraction and when you have a change from 1 to 0 you perform addition.

(Refer Slide Time: 38:43)



Another way of understanding this is like this that you can write a number like this as 1 0 0 0 0 0 0 and subtract a 1 from this position (Refer Slide Time: 39:09). So actually it is this one which corresponds to one subtraction and this corresponds to an addition. So, when this algorithm was devised by Booth the attempt was made to minimize the number of addition, subtraction which are required and that was in a context where a step involving addition will take longer and a step not involving addition will take shorter time. So, attempt was to see if there if there is chain of 1s if there is a string of 1s instead of doing 1 2 3 4 5 6 addition you perform one addition and one subtraction so there is a speed up there.

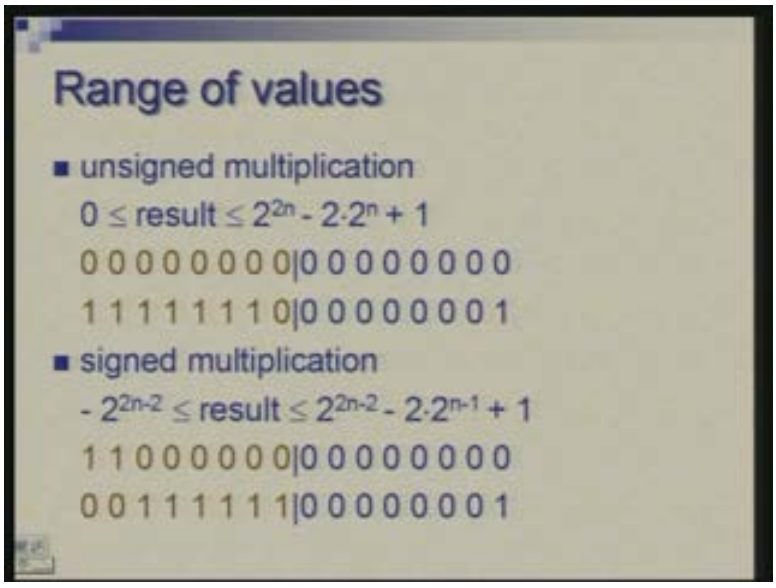
(Refer Slide Time: 39:10 min)



But of course in modern hardware each iteration takes one cycle whether you are doing addition or you are not doing addition so it is not considered as a mechanism to speed up things. But it gives us a mechanism to look at positive and negative numbers in a uniform manner and carry out signed multiplication directly.

Now finally let us look at what is the range of values which a multiplication would produce.

(Refer Slide Time: 40:41)

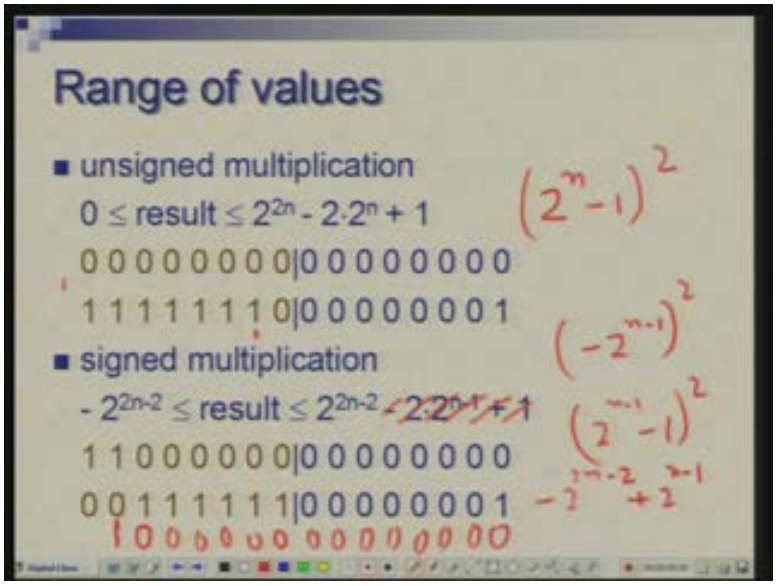


When you are talking of unsigned numbers the result would vary between 0 and this term (Refer Slide Time: 40:55) which is nothing but square of 2 raised to the power n minus 1 so this is the largest unsigned value you have and square of that is this. So this is the largest sum you can get. So this number could be expressed; I have illustrated with an 8-bit example and equal to 8 this is what you will have… this 2 raised to the power 2 n basically corresponds to a 1 here and all 0s. So, from that you subtract 2 into 2 raised to the power 2 n which means 2 raised to the power n plus 1 which means you subtract a 1 in this position so you get this number and this add 1 which is here. So this is the largest number as seen in binary you will get. That means there will be first n minus 1s, there will be n 0s and there is a 1 so this is the common pattern you will find irrespective of what n is. And obviously it requires 2 n-bit registers to hold the result.

On the other hand, on the other hand, when you take sign multiplication the range would be….. so the largest most negative number is this and most positive number is this (Refer Slide Time: 42:44) so if you consider square of this and square of this this will give you the range. So this is 2 raised to the power 2 n minus 2 within a I am sorry this should not be a negative sign. The most negative value will come…… no, this is not correct, this will be, yeah, this will also be positive actually so I should, no, this is the most positive value and the most negative value will come when you multiply most negative with most positive. So let us work it out. What you get is minus 2 2 n minus 2 plus 2 raised to the power n minus 1 so most pos[itive] let us look at each of these first. Most positive value will be 2 raised to the power

2 n minus 2 which should be this followed by all these 0s. So, that is the largest positive value you get.

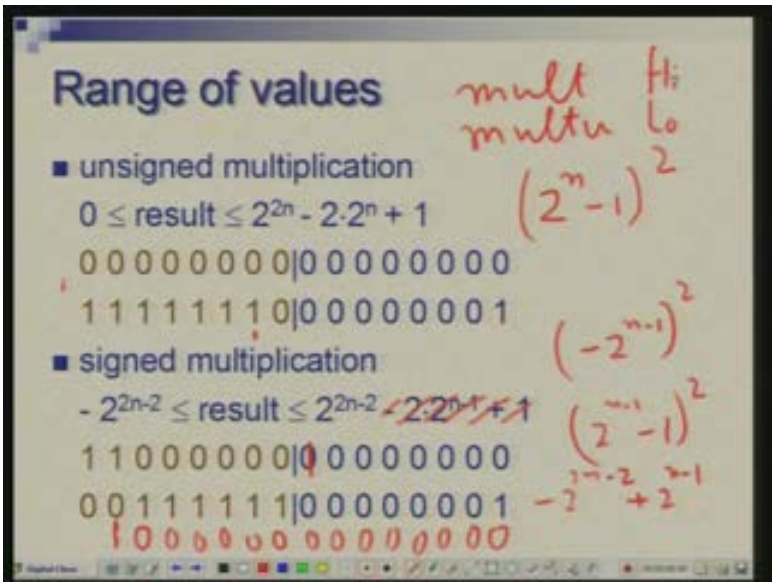
(Refer Slide Time: 44:23)



And the largest negative value you are getting is you have minus 2 n minus 2 is actually this, so, to that you add 2 n minus 1 which is a 1 here. So this is the most negative value, this is the most positive value you get (Refer Slide Time: 44:57).

Roughly speaking you are not you are getting about half the value. If you take approximately you are getting roughly half the value half the values in magnitude but still you have to use 2 n-bit registers you are not utilizing in the last bit fully. So MIPS has a mult and multu two instructions are there; mult and multu; I think in the last lecture I placed it wrongly in the group where overflow is detected and there is no difference in signed and unsigned operation. Actually there is a difference between these two; multu will interpret that two operands as unsigned integers so the ranges are different whereas mult will treat them as signed integers and perform multiplication, get results in this range. Since you are accommodating all the values in 2 n bits you are providing for 2 n bits the two registers are Hi and Lo. So, in both cases irrespective what the result is it can be contained within 2 n bits it never goes out so there is no problem of overflow.

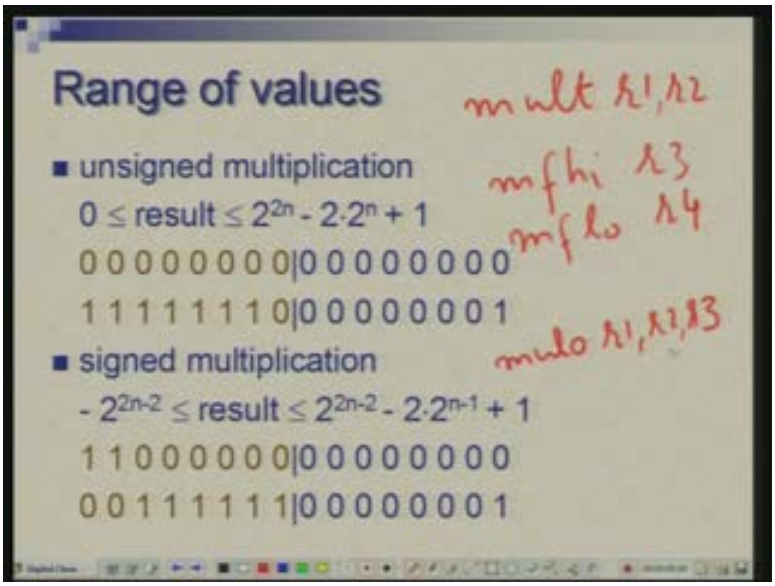
(Refer Slide Time: 46:40 min)



On the other hand, the pseudo instructions which tries to look at only n bits of the results there is a pseudo instruction, these instructions let me let me…….. so let me write this instruction. If you have an instruction with these two operands each is n bits the result is in Hi and Lo and you need instructions move from high and you say r3 or move from low let me put a different register so the result which is in two registers can move to one of the gprs; the high part, another gprs contain the low part.

On the other hand, the pseudo instruction which works with, I think multiply with overflow, it will take three registers so the programmer is not bothering about the fact that the result first comes to high low and then it is transferred to r3…… sorry in this case the result…. r2 r3 are operands so the result is going in to r1. So we are looking at only n bits of the results and ignoring the higher n bits. So, if the product is bigger than that if it requires more than n bits then it is a case of overflow. So this is the exact description of these instructions.

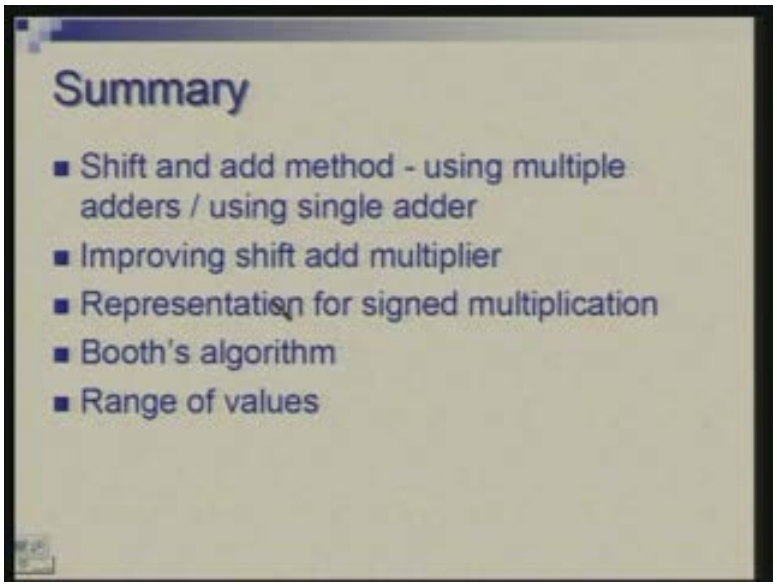
(Refer Slide Time: 48:40 min)



So, to summarize, we began with our paper pencil method as we understood from early days of school that you multiply the multiplicand digit by digit so same thing translates to multiplying the multiplicand by bits of the multiplier and then adding these partial products with appropriate weightage. So we could translate that into a circuit which had one adder from each partial product and then we sort of wrapped it around, put that in a sequential iterative process where the same adder performs additions of various partial products. Then gradually we made certain observations and tried to improve the circuit.

The first improvement was that we reduced the requirement of addition from 2 n bits to 2 n bits and made one of the registers which was holding the multiplicand shorter. Then, after having done that we also noticed that the register requirement can be reduced by sharing in the same register between the accumulated sum and the multiplier. Then we moved on to the case of signed multiplication which was basically derived from a common representation for positive and negative integers and the resulting algorithm was Booth's algorithm.

(Refer Slide Time: 50:28 min)



Although it is the original motivation was different which was to save the time it is a mechanism which allows signed numbers to multiply directly. And then we have also discussed range of the values and different MIPS instruction which look at the numbers differently as whether they detect the overflow or do not detect the overflow. I will stop with that.