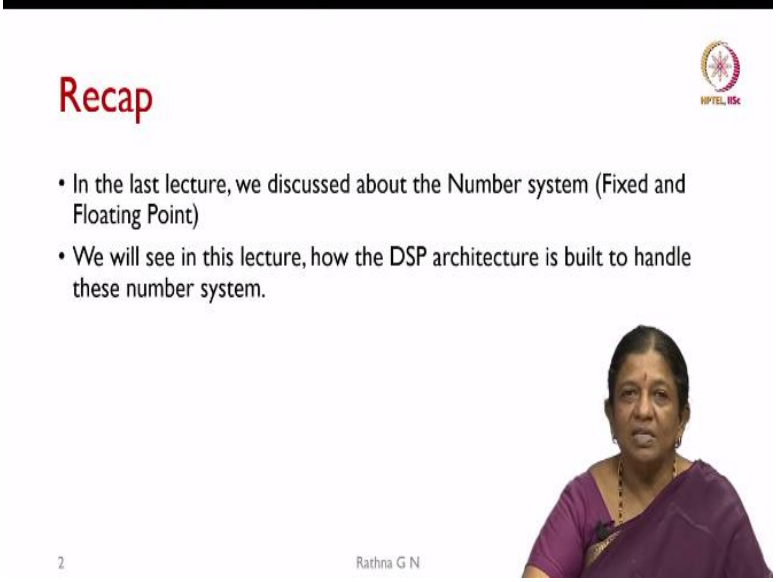
Real - Time Digital Signal Processing
Prof. Rathna G N
Department of Electrical Engineering
Indian Institute of Science - Bengaluru

Lecture – 05
DSP Architecture

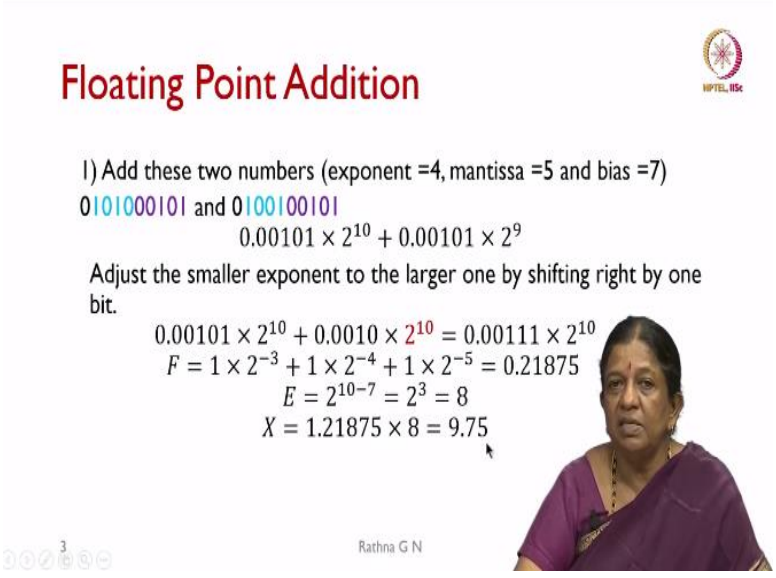
Welcome back to real time digital signal crossing course. So we will discuss today DSP architecture.

(Refer Slide Time: 00:31)



So, just to give a recap what we did in the last class, so we discussed about the number system. So, hopefully, you enjoyed that course, it is fixed and then floating point number system what we discussed in the last class. So, in this class we will be seeing the DSP architecture. So, how this is going to cater to whatever number system we have discussed.

(Refer Slide Time: 00:55)



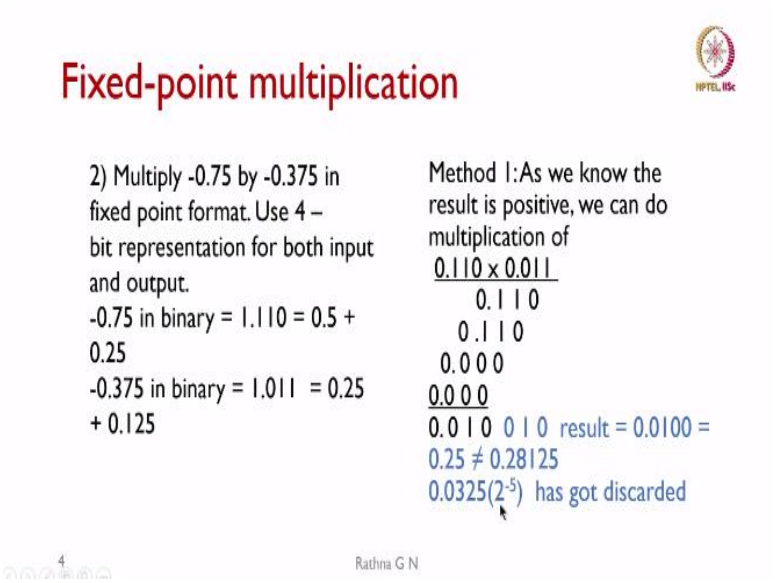
So, coming to the problem what I have posed in the last class, so, you are supposed to add 2 numbers in the floating point format. So the thing was given is exponent was 4, mantissa 5 and bias 7. So, when you are this was the 2 numbers what it was given. So, you know the first bit is sign bit, and 4 bits because you have been given 4 as exponent, 4 bits will be representing exponent in this and rest of them are going to be your mantissa and then same thing with the other number.

Now, we will see that, how we can represent this number in the exponential format. So, if I see the thing, it is going to be 0.00101×2^{10}. And then the next number has a exponent value 9, 1001 is 9, so it is multiplied by 2^9. So, in this case, usually what we do is the smaller exponent, we are going to adjust it to the larger one, so that both exponent become equal, then only we can add the 2 numbers in the floating point number system, we will adjust the exponent to the same exponent.

So we shift this number by 1 right bit, and then increase the exponential to part time as it is seen here. And then we do the addition and keeping the same exponent. So, coming to fraction part it is equivalent as we see the thing it is 1×2^{-3}, what we have it and what it 2^{-4} and 1×2^{-5}, which is going to give me the value is 0.21875. As we discussed in the last class, we will be doing 1 + F. So, the fractional value will be $1 + F = 1.21875$.

And then we said the exponent is the bias to 1. So bias is given as 7 in this case, so it will be 2^{10-7}, which is going to be 2^3, and we do the multiplication by 8. So the resultant is 9.75. So this was the first assignment problem what I had given the thing.

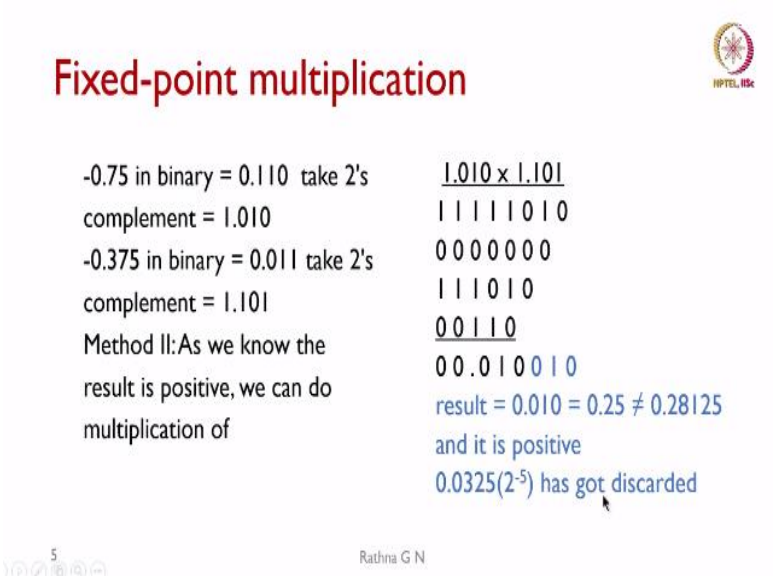
(Refer Slide Time: 03:16)



The second one was fixed point multiplication. So the numbers were given as minus 0.75 and minus 0.375. So, I told you to not to use that the result is going to be positive. So, still we will work out both the methods and see how the result is going to be same in both of them. So, first is we know that minus 0.75 in binary is represented as 1.110, which is nothing but 0.5 + 0.25 is going to give me 0.75 and then -0.375 is this number.

So since I know it is the positive value of what I am going to get it so I can take both as positive numbers and then do the multiplication. So the resultant as you are seeing it here it is 0.010010. So because I said input and output has to be in the 4 bit format, and we said we will be discarding the LSB bits. In this case these 3 bits are going to be discarded. So the result is 0.010. So we will be getting it as 0.25. So originally what we are supposed to get his 0.2815 completely taken into account. Even in the decimal if you multiply 0.375 you should get it as 0. 28125. So what we say is the 2^{-5} LSB bit has got discarded.

(Refer Slide Time: 04:57)

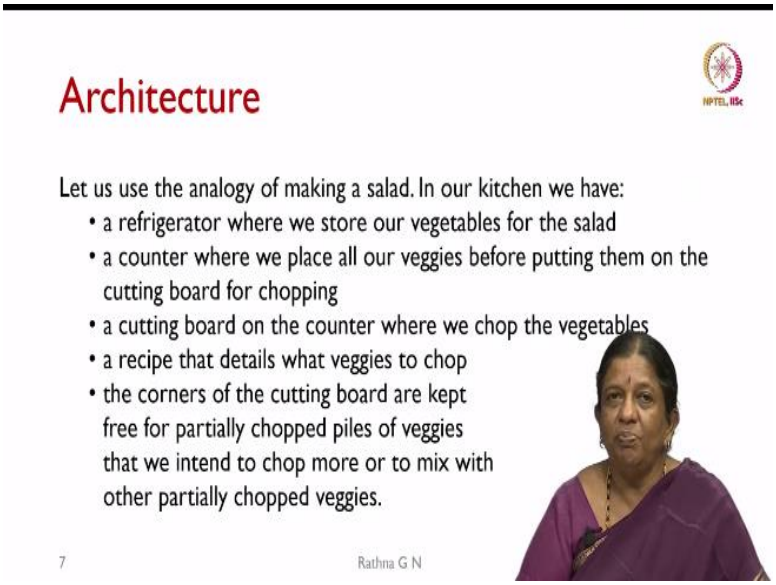


So we will see how to do or 2's complement multiplication. So we said that in 2's complement minus 0.75 is this number. First I take 0.75 and take the 2's complement of that number. And this is the number same with respect to 0.375, 2's complement of that number is this. So in the second method, we will be doing the multiplication in the 2's complement. So, the 1.010 × 1.1101 so, in this case 1 has to consider that when you are multiplying with 1, so you have to extend the digits by 1 we call it a sign bit extension in this case.

So, although it is 1010, later on, we will be putting it 1, when multiplying with 0 we need not have to bother it is going to be 0, again, multiplying 1 so we have done the sign extension with 1 and the last one, because this is the sign bit what we are going to multiply with this number. So, we have to take the 2's complement of this number, and then put it there. So, we know that 2's complement for 1010 is 0.110 so, we will be extending with 000110.

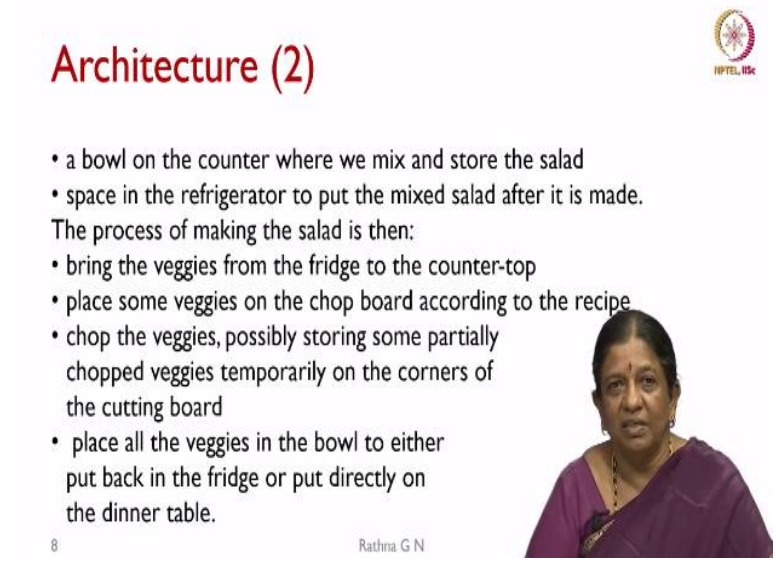
Then we add, we are getting the same result as the previous method. And here also we will be considering it as 0.010 which is equivalent to 0.25 which is not equivalent to 0.28125 what we are supposed to get. So, this is the truncation what we are supposed to have when multiplication of fixed point numbers, here also the 2^{-5} bits whatever 1 has got discarded.

(Refer Slide Time: 06:51)



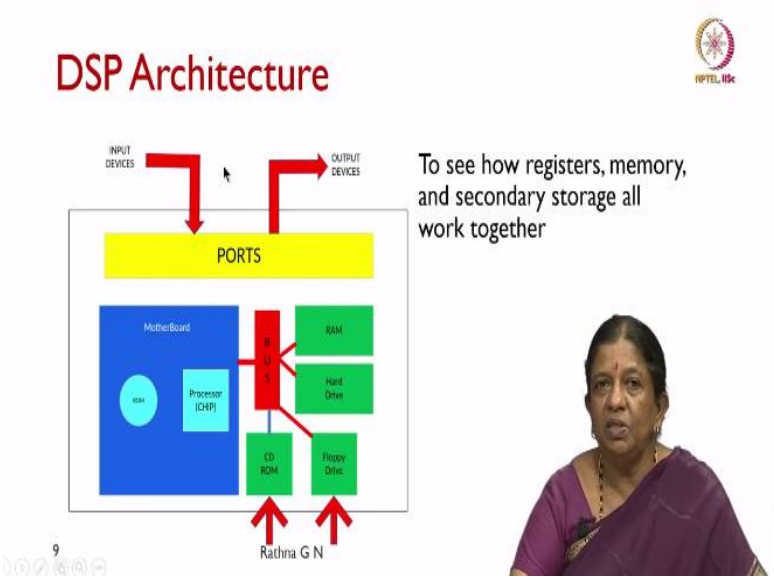
So, with this, we will see how the architecture is going to get fine tuned for these numbers handling in that, as an example, we will consider how one novice wants to prepare a salad in our kitchen. So, what are the things required, we have said the refrigerator, counter cutting board and recipe first of all required what are the corners of the cutting board are kept free for partially chopped once to add rest of the thing.

(Refer Slide Time: 07:25)



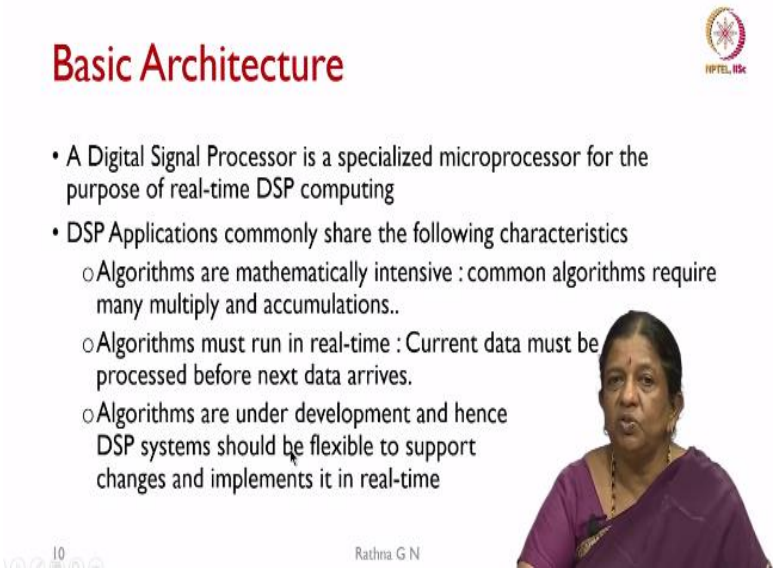
So, in the next lesson, we will see that how the bowl is taken and then mixed everything, once all the veggies have been now considered and then mixed. So, you can keep it back in your fridge or you can use it for consumption, this is how the procedure is going to follow. So, we always think in a simple term, but here you will be seeing each step has been recorded, how long it is going to say it take.

(Refer Slide Time: 07:55)



So, the same way we are considering DSP application also in this manner and see how the architecture has been developed. So, all of us know that regular processor has this architecture basically, we have a motherboard here and then we have the ROM and then we have the processor chip in this then through the bus all the peripheral units have been connected. So what are the peripherals, the registers memory and secondary storage are all going to work together and we have the boards for taking the input and then putting it on the output also.

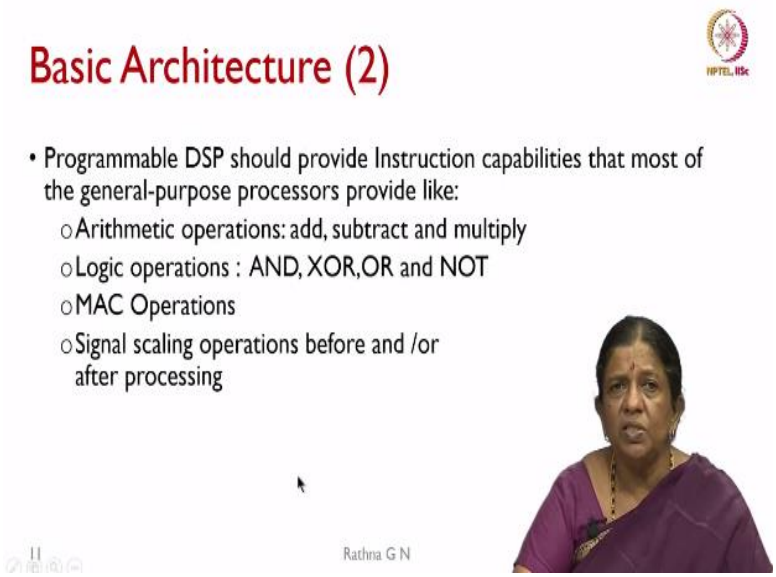
(Refer Slide Time: 08:37)



So coming to the basic architecture of the DSP. So, we say it is a specialized microprocessor, for the purpose of real time DSP computing. So the applications what are they in commonly used in digital signal processing is so they are mathematically intensive, that is common algorithms require many multiply and accumulation. And then we know that algorithm has to run in real time. That is current data must be processed before next data arrives in the clock cycle, what we call it, every clock cycle, I am going to get the data.

So before the next data comes I should have finished my computation and algorithms are under development. And hence DSP system should be flexible to support the changes, if the algorithm is going to be changed or whatever may be the thing and we have to implement them in real time.

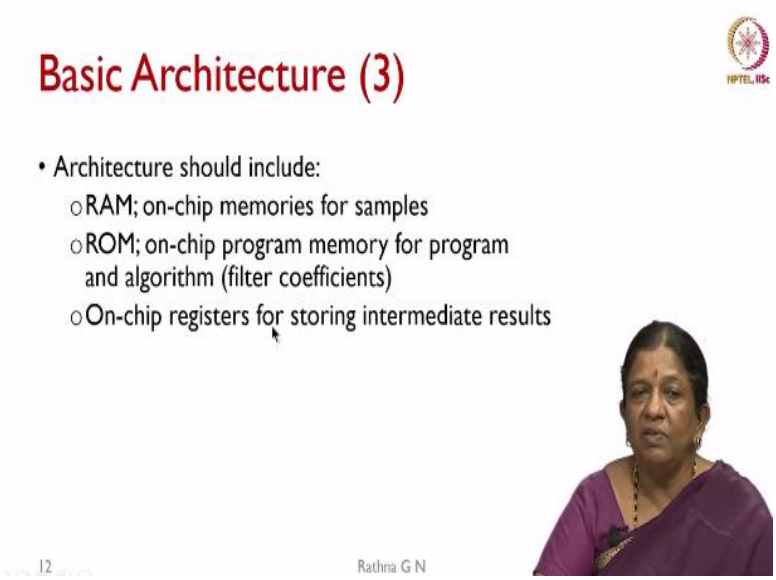
(Refer Slide Time: 09:38)



So coming to the continuing with the architecture, what all the other things what we need it? So one is arithmetic operations like any other general purpose processor, add, subtract and multiply is one extra what we will be adding it and other support has to be logic operations like AND, XOR, OR, NOT. And we need extra is the MAC operations because we need multiply and accumulate. And some of the earlier processors had both multiply and accumulate, nowadays multiply unit is separate and then accumulator is separate.

But we can run them in parallel so that I can get the MAC operations in 1 clock cycle. The next one is we have to do scaling of the signal before and after hold on a while, why we need it we will come to that discussion in a while.

(Refer Slide Time: 10:33)



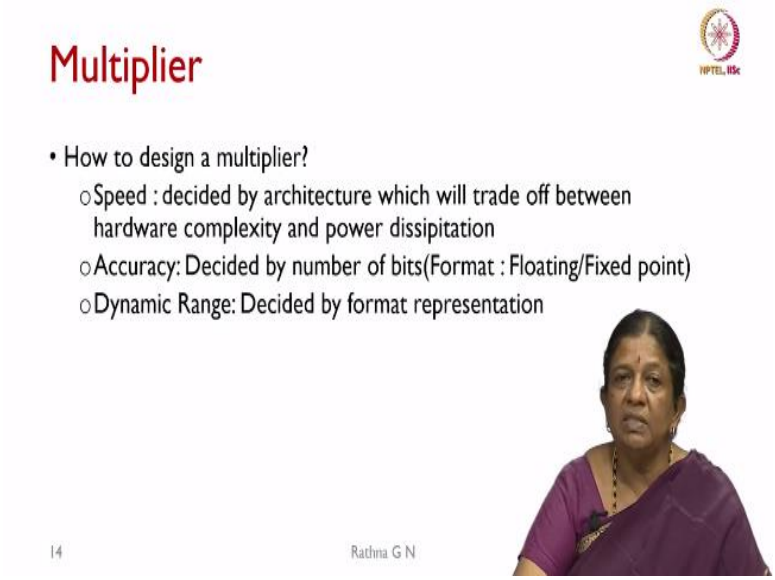
So coming to the next one, what all the other things from the memory point of view, we need a RAM on chip free memories for samples because we are telling real time signals are coming in. So we have to have the RAM to collect it. And next one is the ROM that is on chip program memory for storing our program. And then we will see that with modification, we can store our filter coefficients also in the ROM.

The next one is on chip registers for storing intermediate results because we know that memory access is going to slow down our operations. So we want to have it everything is faster. So we say that we can store the intermediate results in registers which are much closer to our architecture.

(Refer Slide Time: 11:24)

Basically, we will say it as a DSP chip. So the building blocks for digital signal processing computations are multiplier, shifter, and MAC units, their capabilities. And last one not the least one we will say it as ALUs that is Arithmetic Logic units.

(Refer Slide Time: 11:47)



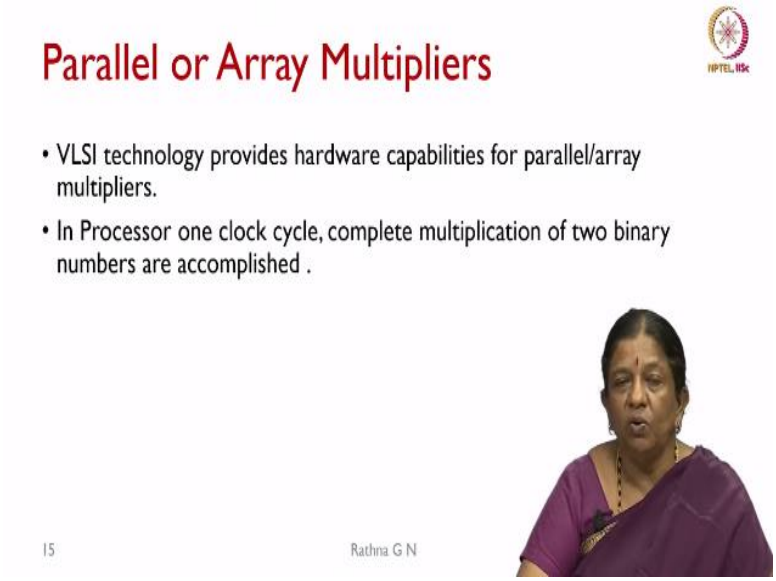
So, coming first, we will take up multiplayer how it can be designed. So one is we are looking at because we have lot of multiplication and accumulation. So, speed is one of the criteria, this is going to be decided by the architecture. So, we have to have a trade off between hardware complexity and power dissipation, as the hardware increases, our power is going to be increased also.

So, we want to keep it lower power consumption and then we want to have more complexity in the hardware. So, we have to match between the 2 of them. The next one is accuracy so, this is going to be decided by number of bits, as we discussed in the last class, it is going to be the format can be floating point or fixed point. So, we know that for the floating point numbers of bits are going to be more so our hardware is going to increase but at the same time accuracy will be more.

So, what is the trade off between the 2 one has to look at it the next one is whether we can creator architecture to large dynamic range. So, this again is going to be decided by format representation. I know floating point numbers have large dynamic range whereas fixed point will be having the low dynamic range but they equal floating point numbers in precision. So it depends on what kind of application one is choosing depending on it one has to match all these

things and then choose the correct a DSP processor or design your own DSP processor using FPGA.

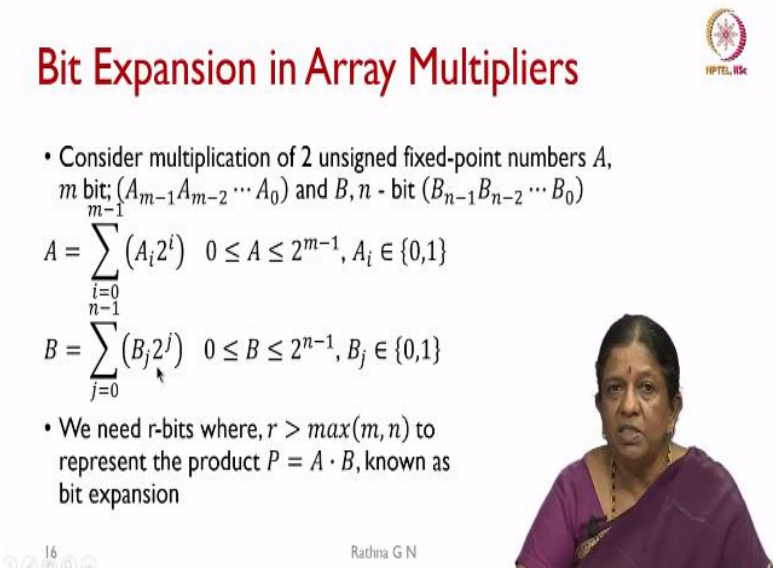
(Refer Slide Time: 13:33)



We will see that how we can increase the parallel or array multipliers what we call it just now we did the fixed point multiplication. So you saw that the multiplication is basically a successive addition. So if we consider n bits we will take n clock cycles, and then $n + 1$ clock cycle will be required to do our successive addition. So whether we can improve on this so for that we will be going with parallel or array multipliers because VLSI technology provides hardware capabilities for accommodating these multipliers.

And we want in processor 1 clock cycle we have to complete the multiplication of 2 binary numbers that is what are aim is or goal is?

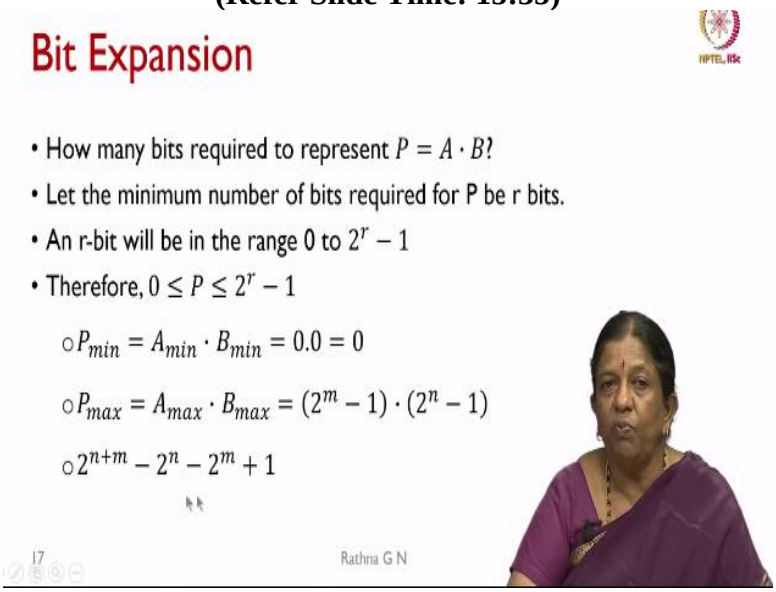
(Refer Slide Time: 14:32)



So how we can do that with a little bit of with expansion we will see it so we will be considering the multiplication and to unsigned the fixed point numbers in this case, A is m bit, and then B will be n bit number what it has been chosen. And then it can be represented in the summation form like this, A will be ranging between 0 to 2^{m-1} and A_i will be belonging to either 0 or 1, same way we will B operand also.

So, we say that we need r bits, where r is going to be greater than maximum of m, n to represent the product, which is P = A into B, we know that we need more than maximum of one of the bits m and n are not equal need not have to be equivalent in this case, that is what, what we are considering it. So we are taking the maximum of it, but it should be greater than that. So we will see how much greater we need it in a while.

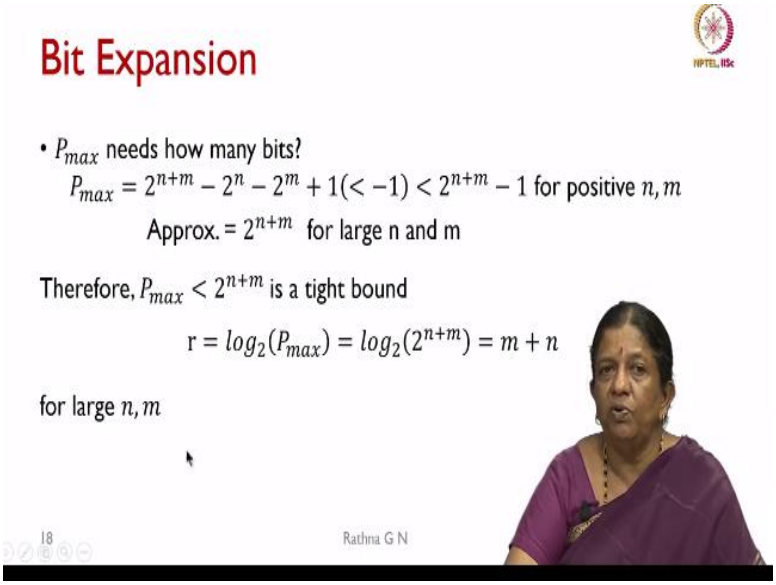
(Refer Slide Time: 15:35)



- So, that is what the question is posed also, that is what should be the number of bits that is r has to be, we will say first consider the minimum number of bits required P be r bits will say, r bit will be in the range 0 to $2^r - 1$. Therefore, our P will be in the range between these 2 P see first P_{min} when A bits are all 0s and b bits are 0s, then the minimum is going to be product will be 0, when will be the maximum when all A bits are 1s and b bits are 1s which is we call it as maximum.

So then we know that there are m and n bits, so it will be $(2^m - 1) \cdot (2^n - 1)$. So when we expand this multiplication, so we will be we are resulted with $2^{n+m} - 2^n - 2^m + 1$ is the number of bits what we needed.

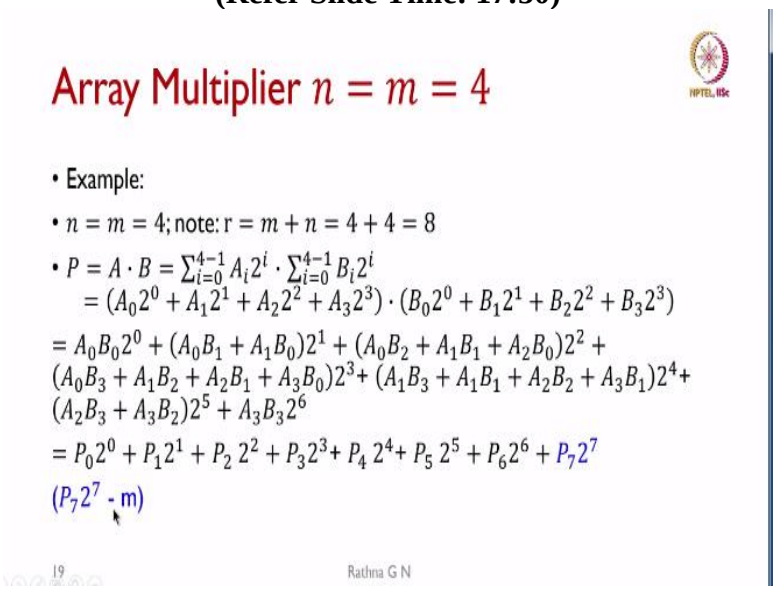
(Refer Slide Time: 16:40)



So we will see that how P_{max} can be represented. So, we have represented this and what the term is going to say is this terms can be represented as minus 1 that is it should be less than or equal to $2^{n+m} - 1$ for positive n and m. And approximately if n and n m are very large, we can approximate this P_{max} to 2^{n+m}. So therefore, we say that the maximum number of bits required to represent the product is less than 2^{n+m}.

So we said this as a tight bound, then what will be r, so we take the log of it $log_2(P_{max})$, which is nothing but the $log_2(2^{n+m})$, which is going to be $m + n$ is the number of bits, maximum bits what we needed. For this, it may be less than that that is what, what we have put for large n and m.

(Refer Slide Time: 17:50)

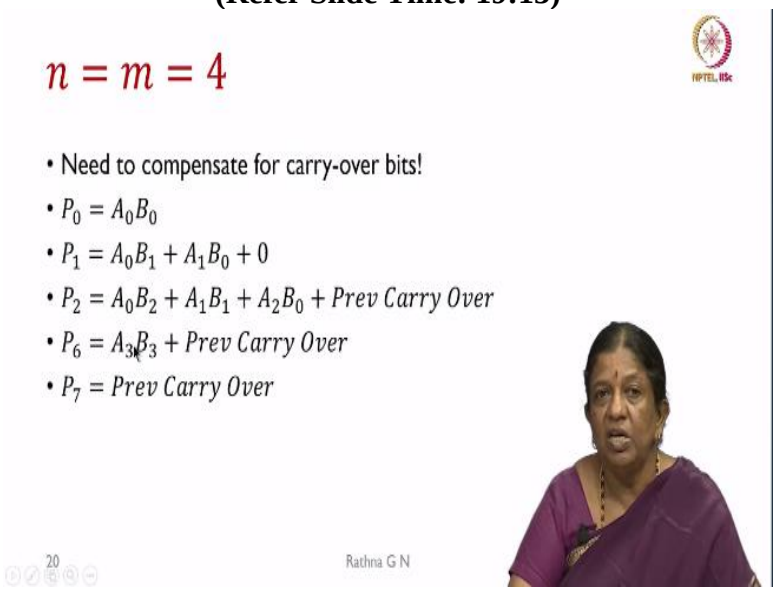


So how to do that, we will take an example. In this case, I have taken $n = m = 4$both of them are same. So then we know that we need r = m + n in the maximum that is 8 bits do we need

that we will see with the expansion of multiplication A and B are represented with the 4 bits A here 4 bits and B and when you expand this summation basically multiplication and summation you do the thing this is the result what we will be getting it.

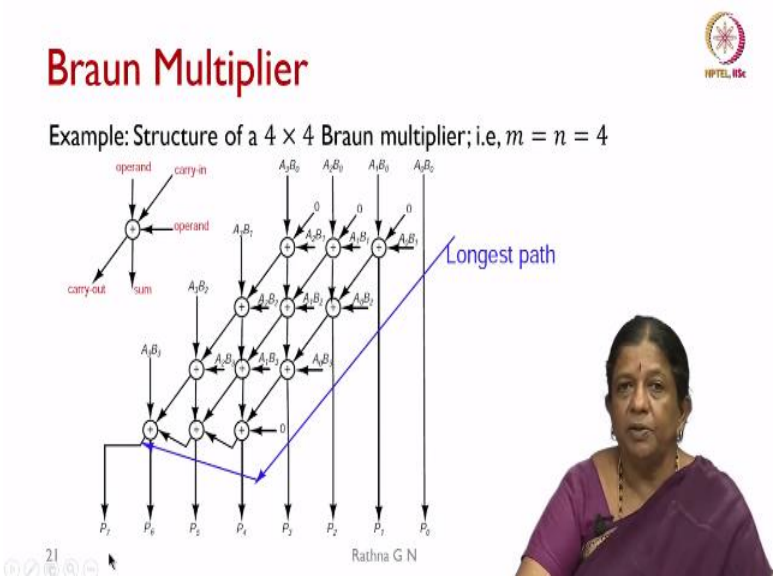
So when we further multiply both of them, so these are the products what we will be getting it, so from here to here are the products which have to be summation has to happen for them, then when we represent these with products as P_0. So, we will be putting their powers what are they because it has in terms of powers but it has been segregated. So we will be needing P_0 P_1 P_2 P_7 is the maximum what we need it. So 0 to 7 which gives me 8 bits so this is what, what we say ($P_7 2^7$ - m) what we will be putting the number.

(Refer Slide Time: 19:13)



So we will see why we need P_7 also we will see the thing. So, we have to compensate for carryover bits what will say that, so, we have represent P_0 to P_1 P_2 P_6, how the summation is going to happen. So, the first one is only the multiplication later on we have multiply and accumulate. And then this is a half adder what we need it for the P_1 product, but from P_2 onwards we need full adder as you are seeing it we are adding these bit numbers and with the previous carry and then same thing with the P_6 and then P_7 represent only the carry that is coming out of after adding in the last stage.

(Refer Slide Time: 20:04)

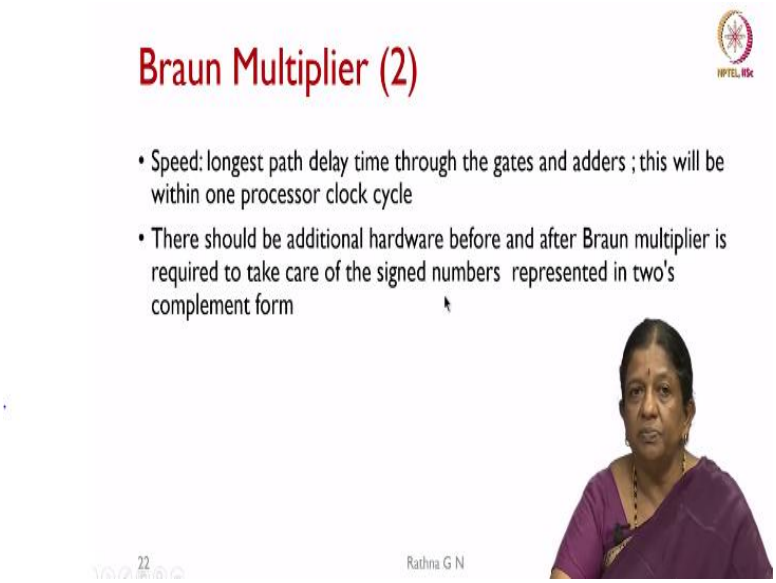


So we will see the architecture how does it look like so, most of the DSP processor use a Braun multiplier, you can see that 4×4 Braun multiplier how it is represented. So, this is your full carry. So, I have an operand and previous carry n is going to be input of this and the second operand is coming from the side and the result is going to be sum and then carry out is going to be generated from full adder is shown in this way. So, we say that I need because we have taken $m = n = 4$.

So, these are the structures what I need it from the multiplication and addition these are the adders. So, as you can see that this is the multiplier what we are representing but we need AND gates to do multiplication of these numbers whatever you are seeing that is the additional hardware what it requires. And then in the first stage you will be having no carry n is going to be there so, you will be pumped up we are using all full adders.

So, that is why the carry m is going to be 0 in this case, this is one operand and then the other operand the first one we said $P_0 = A_0B_0$ which comes out directly after that what we will be doing is our addition what it is going to follow with these numbers. And we say that the last stage in this multiplayer what we have to do is we have to do a ripple carry adder what we are supposed to use it that means to say this adder has to give its carry out to this and from here to here and then from here, what the P_7 bit is going to be generated.

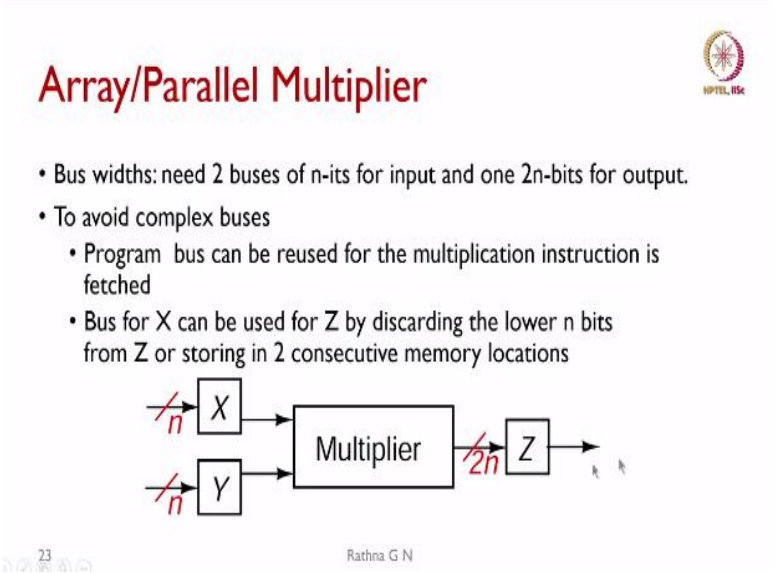
(Refer Slide Time: 21:55)



So, coming to the Braun multiplier, we say the speed that is longest path delay as we are seeing the thing in this cases all these additions and then the last stage ripple carry adder delay also one has to add it which gives the longest path delay. So what is it through the gates and then adders, we said this will be within 1 processor clock cycle. So my in internal clock rate may be much faster than the outer clock. So see that all these are done in 1 clock cycle.

And we say that there should be additional hardware before and after Braun multiplier required to take care of signed numbers because we took it as unsigned number. So, as we did sign multiplication, if one of the number is both of them are negative, then we have seen that we have to do the 2's complement, this is the hardware which is required further, before we do the multiplication of the number for the sign numbers.

(Refer Slide Time: 23:06)

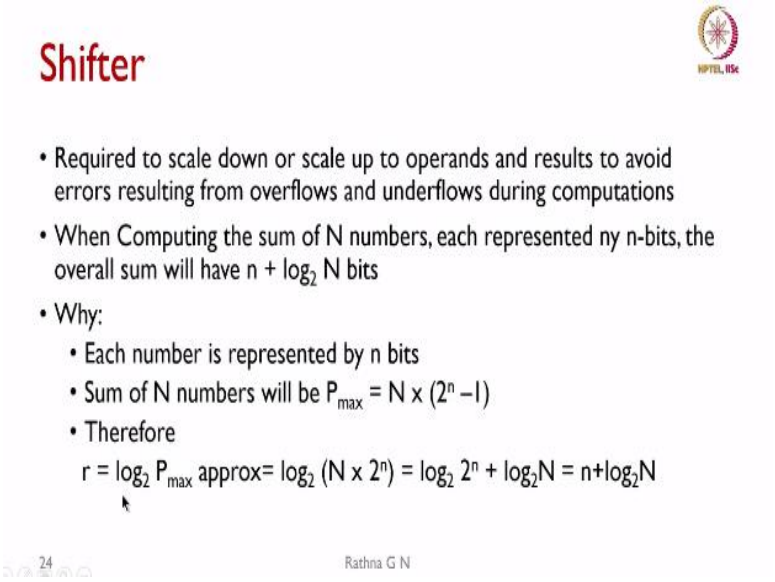


Coming to the array multiplier or parallel multiplier, what is the bus width required so we need 2 buses of n bits to directly give the parallel we give the input to our multiplier. And we know that X into Y the product we said maximum length is m plus n which is going to be if I am taken both are n actually bits for the 2 inputs then $2n$ will be the maximum on bits what I need it output of the multiplier. So that can be Z. It depends on the application that is what, what it says.

So program bus can be reused for our multiplication instruction is when it fetch after fetching the instruction. And that is most of the cases coefficients what we will be taking it so which can be pre stored in the ROM and then we can fetch it from the programmers. So bus for X can be used for Z once we have done use the input so I can use the same bus for my Z that is discarding the lower n bits from Z or storing 2 consecutive memory locations.

So as we know that if we want to store 2n bits output in the memory it is going to take twice that of the memory requirement. And then computation loading when we have to load $2n$ I would not be able to load them in 1 shot I had to do them in 2 clock cycles so which is going to slow down. So most of the DSP processor what we do is we only store the higher bits as we discarded in our example also lower bits, high bits n bit whatever is defined is going to be stored in our memory through Z bus.

(Refer Slide Time: 25:10)



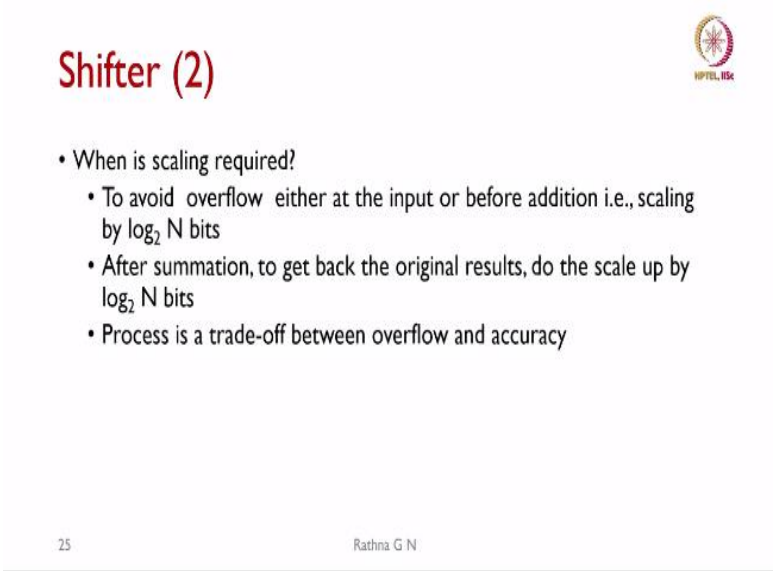
So, the next one is after the multiplier we need the shifter. So, first is why do we need the shifter one is it is to scale down or scale up to operands and results to avoid errors resulting from overflow and underflows during the computations. So, as we saw in the previous case, in the

example, we scale down all our numbers and then use it for our multiplication that is converted into fractional from integers to fractional numbers, so that fractional multiplication is not going to overflow. So, we have we need that scale downing there.

And then once the result is there, we may have to scale it up back to the previous whatever scaled down value and then give the output. So, this avoids are overflows and underflows. So overflows, all of you must be knowing that the value is much more underflow is which goes below or whatever maximum negative value what it has been provided. So, when computing the sum of n numbers, so each is going to be represented by n bits, the overall sum will have we know that $n + \log_2 N$ bits.

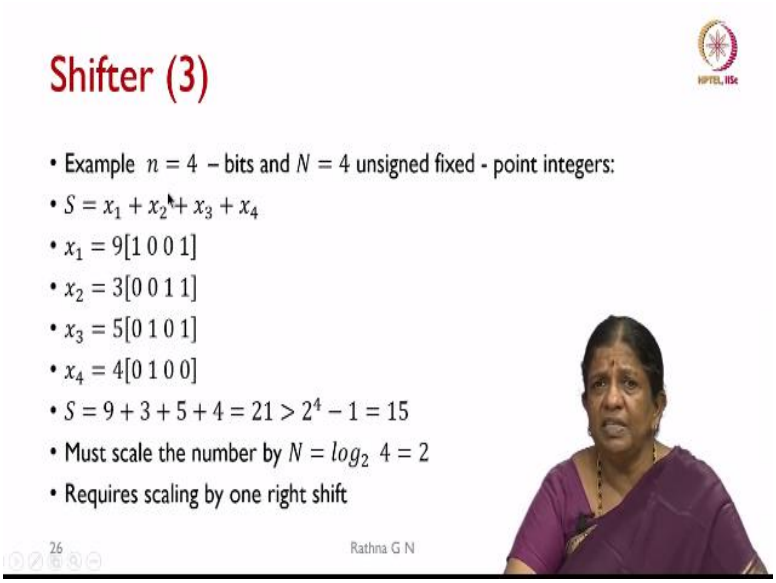
So, we have to say why this is so each number is represented by n bits we said and some of n numbers you have seen that $\sum_{i=0}^{n-1}$ ⬚, then the maximum width going to be $N \times (2^n - 1)$. So therefore, r whatever for the bits that is required for my product, we will be putting it as $log_2(P_{max}) \cong \log_2(N \times 2^n)$ so which is nothing but $log_2(2^n) + log_2 N$. So, we know that $log_2(2^n)$ is nothing but $n + log_2 N$. is the representation what we are going to have it.

(Refer Slide Time: 27:18)



So depending on this we will say when is scaling required? So we said to avoid overflow either at the input or before addition that is scaling by $log_2 N$ bits. So that is what we said that maximum what we can represent here. So depending on that, we may have to scale the number by $log_2 N$ bit. So after summation, to get back the original results, we will be doing the scale up by $log_2 N$ bits. So this is a trade off between overflow and then accuracy one has to consider.

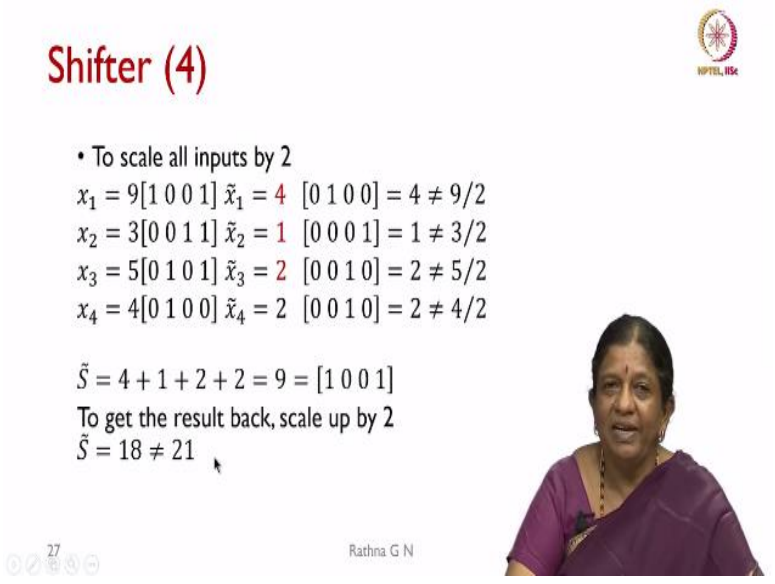
(Refer Slide Time: 27:58)



So we will see that how we can do the shifting in this case, why we need scale up or down. So, as an example, $n = 4$ bits, and then capital N number of additions what we are going to have it also 4 basically we have a chosen unsigned fixed point integers case. So then our summation is going to be x_1 is represented with 4 bits. And then x_3, x_4 four variables are there we are adding it up as an example, these are the values what we have taken their binary representation shown in the square bracket.

So when we sum up, we are going to get it as 21 is the result. So the maximum, because we are representing it 4 bits, what I can represent maximum value is 15. That is to $2^4 - 1$ so which is much greater than this 15, 21. So then we have to do the scaling. So in this case, because we have assumed n is equal to also 4 numbers what we have it, so log_2 is 4, so we have divided by that is scaling by 2. So that is scaling by 1 right shift is basically equivalent dividing by 2.

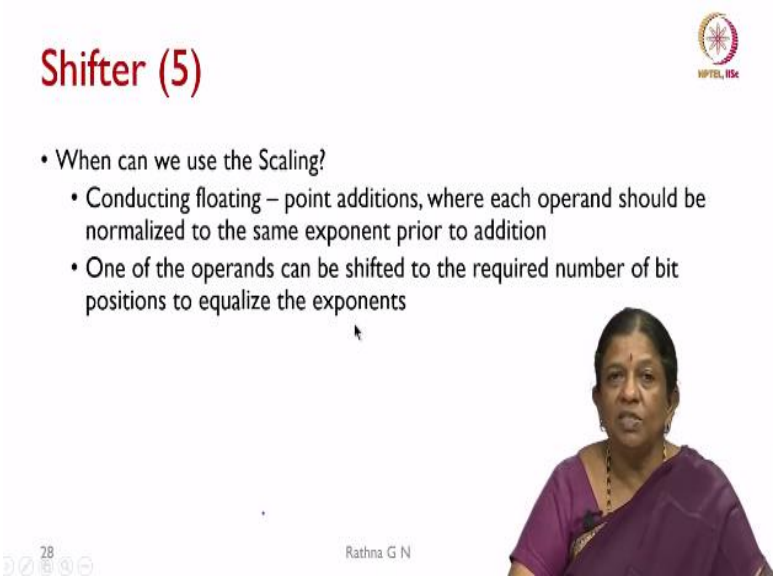
(Refer Slide Time: 29:14)



So we will be scaling all inputs by 2, we will see the same numbers when we do the scaling, what you will be getting is x_1 hat, so that is not equivalent to x_1. So when it is going to be 4, when I do multiply by 2 here, I will get 8, not 9, same way whatever shown in the red or the approximated value to the original one. So if they are divisible by 2, you know that we will be getting back the number otherwise we will be having the approximation.

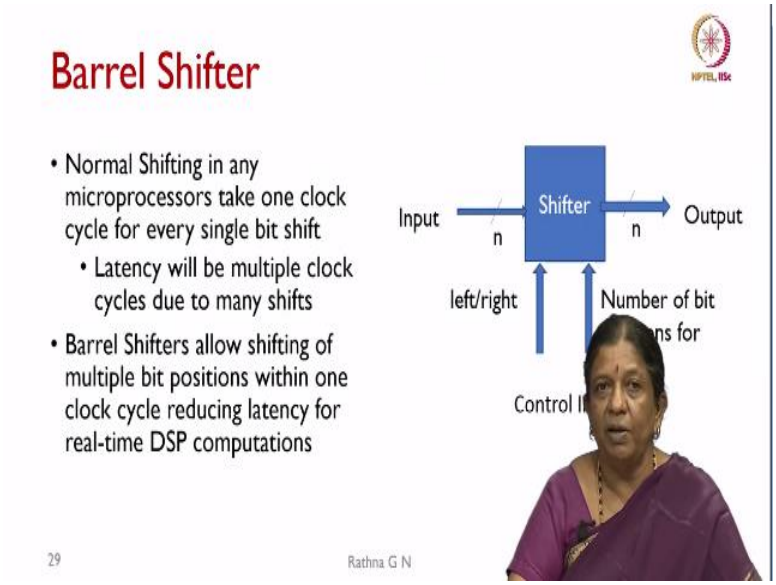
Then we will see what is the summation? I am going to get it when I add these numbers, which is going to be 9. So to get back result we have to scale up number by 2, which is going to be 18 which is not equivalent to 21. So, these are the errors one has to consider. If it is not tolerable then we might have to go further floating point number representation. Otherwise if it is within whatever errors usually we call it a signal to noise ratio. So, if it is in that we can accept this and go ahead with this processor architecture.

(Refer Slide Time: 30:28)



So when can we use the scaling? That is what we said conducting floating point additions, when we need to align my our exponent that is what, what it says where each operand should be normalized to the same exponent prior to addition, I need the scaling and one of the operands can be shifted to the required number of bit positions to equalize the exponents, that is what we did in our example.

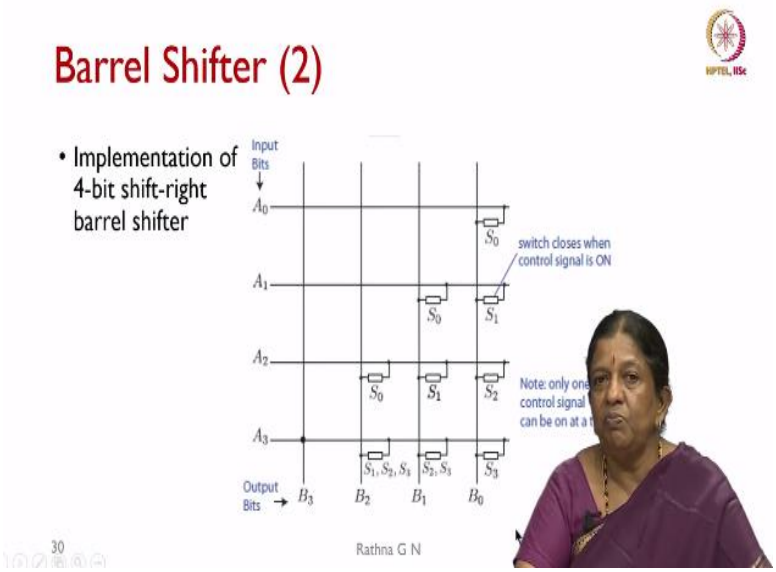
(Refer Slide Time: 30:58)



So, how we can accommodate the shifting? So, we know that sequential shifting is going to cost us n clock cycles if we need n bit shift. So we have a barrel shifter that is normal shifting any microprocessor that is what, what it says 1 clock cycle for every single bit shift. So what the latency will be multiple clock cycles due to many shifts. Whereas the parallel shift is allow shifting of multiple bit positions within 1 clock cycle, reducing the latency for a real time DSP computation.

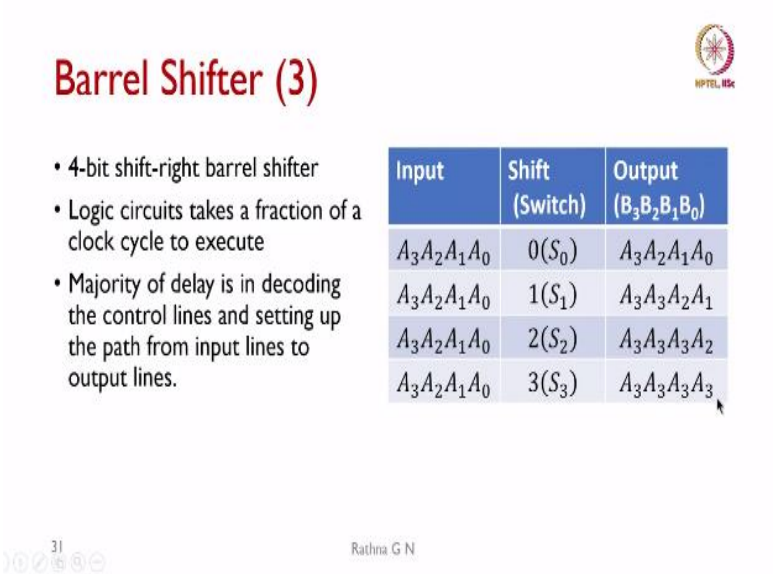
So what we have, I have a shifter here, input is going to be n bits and output will also will be maintaining n bits, and we will be telling these are the 2 control bits. One is whether we want to do a shift left or shift right. And then how many number of bit positions I want to shift it is given as the control inputs.

(Refer Slide Time: 32:05)



So we will see how barrel shifter works, we have taken example as a 4 bit shift right barrel shifter has been designed here. So you will be seeing these are the $A_0A_1A_2A_3$ or the bit inputs, and then outputs are going to be $B_0B_1B_2B_3$. So what are the input bits it will be going into this what we will be seeing in a while, and we have switches $S_0S_1S_2$ and S_3, and we say that switch closed on control signal is on and then only once which can be operated at a time. So we will see how this right operation is going to help us.

(Refer Slide Time: 32:51)



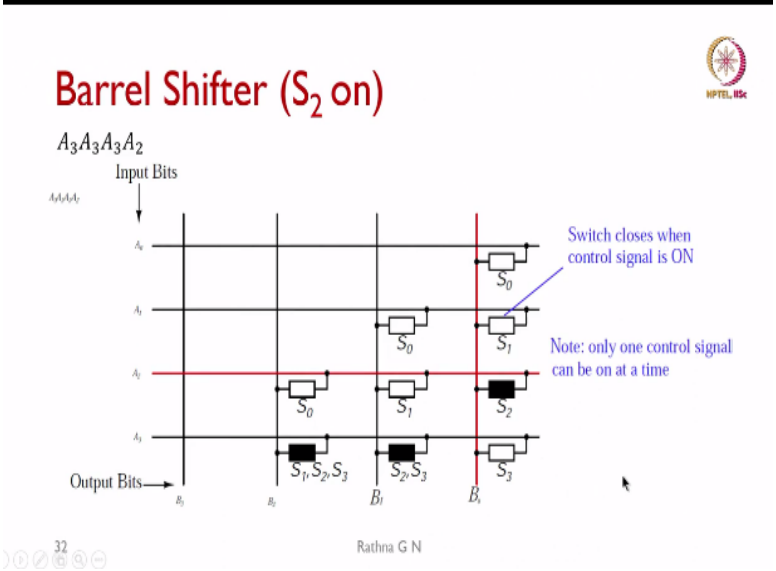
So the first is 4 bit right shift barrel shifter what we are considering it and logic circuit takes a fraction of a clock cycle to execute this. And majority of delays in decoding the control lines and setting up the path from input lines to the output lines. These are causing the delay otherwise we will be getting the output in a fraction of a second or clock cycle we will say. So my input first is $A_3A_2A_1A_0$ are thing.

And we will see the switch positions actually first is 0, that is represented with S_0 and output is$B_0B_1B_2B_3$, since only S_0 is being used, I am not doing any shift operation. So that input whatever you have given will be going into these output lines as it is, when switch 1 is operated, I have to do a shift right operation by 1. So, in this case, whatever the MSB value is going to be pushed into the last place basically.

So we will be shifting A_0 out of it, it becomes $A_1A_2A_3A_3$. So when S_2 is on, then 2 shift what I need it so A_0A_1 is going to be shifted out and the result will be $A_3A_3A_3$ and then A_2 which are going into this output B_0 to B_3. So when I want the maximum shift, so you will be seeing

that all the A_0 to A_3 or shifted out and then the result will have bits are going to be only A_3 in this.

(Refer Slide Time: 34:33)

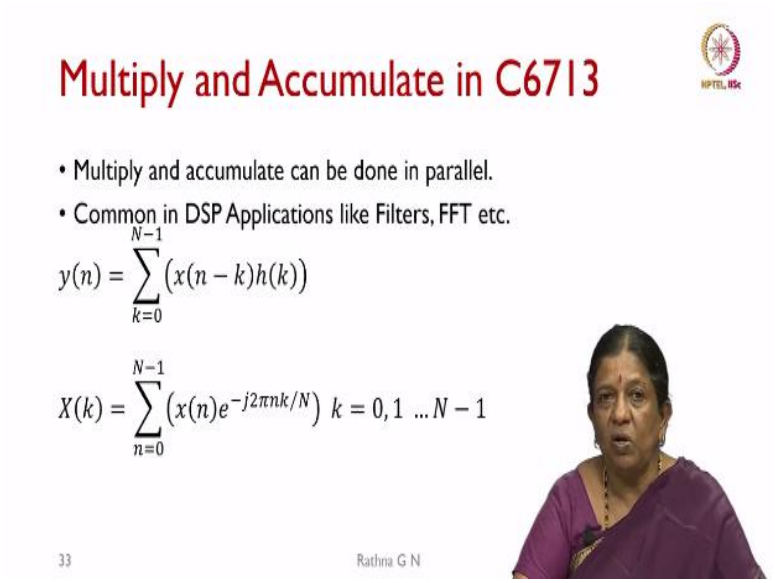


So, as an example, how the switching is going to be done is shown in this. So, what is the result I want $A_2A_3A_3A_3$ on the my B zeros. So, you will be seeing that S_2 switches closed wherever you have seen the S_2 switch you have it is marked highlighted here. So, these are the ones which is going to these lines are going to be connected. So, when you connect this one what happens, so, you will be seeing this red line correct.

So, your A_2 is going to be pushed into B_0 and then your A_3 is going to be pushed into B_1 and then A_3 will be pushed into B_2 also and A_3 as it is it will be in B_3. So, you will be getting the output as $A_2A_3A_3A_3$ that is what, what we wanted since the switch is closed, this is how the barrel shifter works. So, one can try how the left shift operation is going to happen. So, hope you have done a little bit on microprocessor 8085 or something like that there are 2 ways of shifting left.

So, either you can push the zero into the first LSB bit or you can push the carry into your last MSB into LSB bit. So, you can try how you can implement using the barrel shifter.

(Refer Slide Time: 36:08)

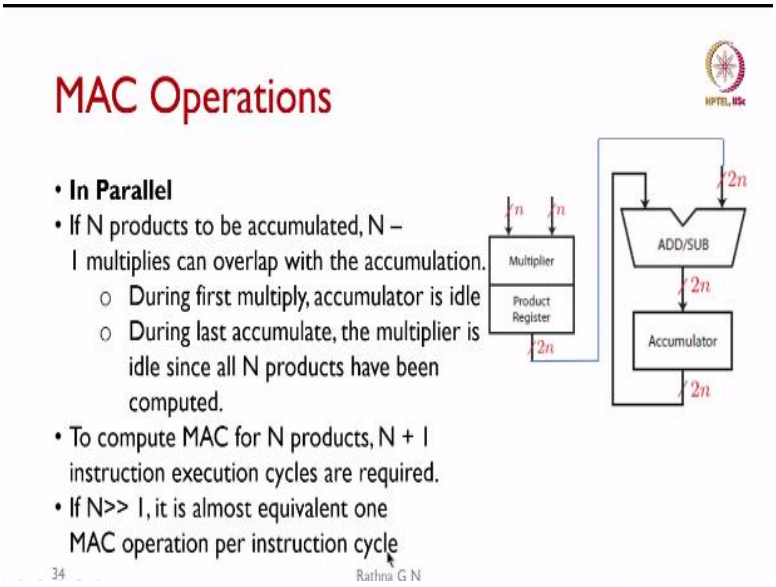


Now, we say that multiply and accumulate what we need it how it is going to be represented in 6713. So, we said we want a Mac operation. But the later versions all 6, 6 processes have only multiplier and accumulator separately, which will be done in parallel. So, common DSP applications we said filters and then Fast Fourier Transform what we want to implement to have it and then rest of the things you can consider. So what is their equation what it is shown here.

So, here $y(n)$ is $x(n-k)$, that is my input, and this is my filter coefficients. So, you would have seen that some of the equation it looks like your convolution equation. So, either $\left(x(n-k)h(k)\right)$ would have been comfortable. But we use here input for $x(n-k)$ and then $h(k)$, why I will consider it when we take up the architecture further, I need a circular convolution what I need it. So, we will be taking it up in the next class and then see why I need that.

So, the next one I know that $X(k)$ is my Fourier output and $x(n)$ input $x(n)$ and then $e^{-j2\pi nk/N}$. And this also I need multiplication and accumulation I can consider sine and then cos function separately and then do it parallely. So, we will see when I take up FFT how we will be doing the operations in our DSP processor.

(Refer Slide Time: 37:59)



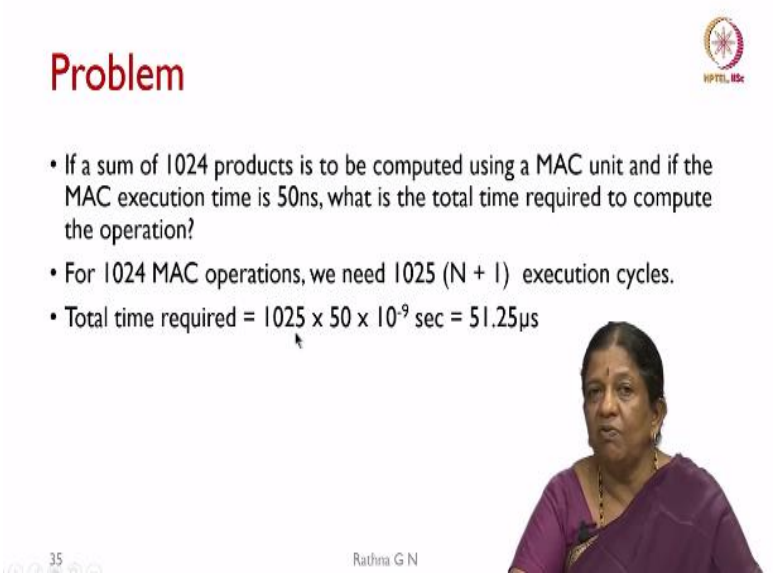
So, coming to MAC operations, we said in parallel what it is going to happen, so, what are the things we need it. So, if N products to be accumulated, we know that we need N minus 1 multiplies can overlap with the accumulation basically, that is during the first multiply accumulator is going to be idling. In the next clock onwards, we can do the thing parallely and the last clock cycle, because 1 value is left out to be added. So, this will be adding the last product.

So to compute MAC for N products, we say that we need $N + 1$ instruction execution cycles are going to be required. If $N \gg 1$, it is almost equivalent to 1 and then we say that MAC operation per instruction cycle what I will be getting it instead of $N + 1$. So you will be seeing that either adder or subtractor. We have not discussed this subtractor we will be using the adder itself for our subtraction because we are doing it in the 2's complement.

So I do not need a separate subtractor only the one of the thing what we had to say is I want to do the subtraction, so that it will be happening in the 2's complement. So there are $2n$ what is the input and then for the addition and subtraction you are seeing this sorry first multiplier is n bit, $2n$ bits are coming, I will be doing the multiplication and product register will hold the $2n$ bits, which are fed as input to our adder.

And then which is coming in here as $2n$ and whatever accumulator is storing in the thing which is fed back. So, which is going to be added in the accumulator, and then the output will be coming out of it, then we will final round will be making $2n$ to n when we are storing back in the memory.

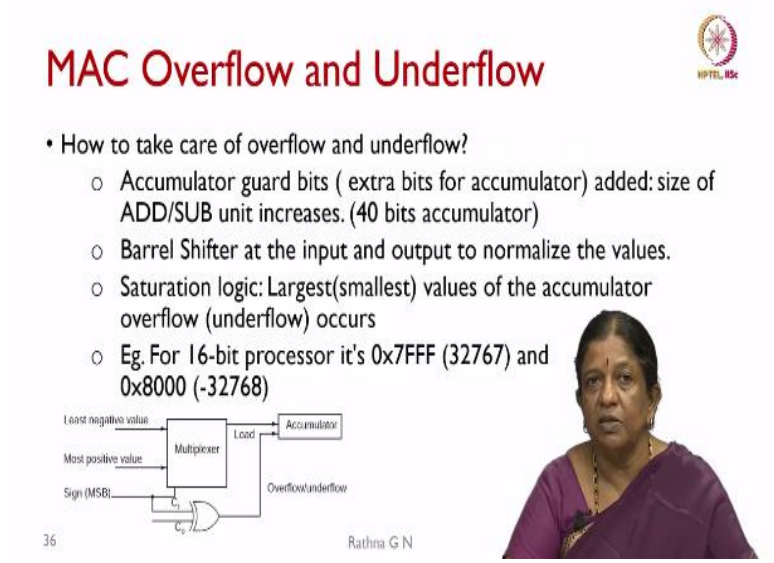
(Refer Slide Time: 40:17)



So, just to see that, how each instruction is going to take, we will see that what is the time required for MAC operation. So, we say that if I want to do is 1024 products to be computed using a MAC unit and if the MAC execution time we have given it as 50 nanosecond the hardware takes, so we have to say what is the total time required to compute the operation. So, we said 1024 MAC what we need it. So, we need basically $N + 1$ which is nothing but 1025 execution cycles what we need it.

So, the total time required what we are putting is 1025 into because each operation is taking 50 nanosecond which is nothing but 50 x 10^{-9} sec, 1 nanosecond = 10^{-9} sec, so, which comes to about 51.25μs, which is required to do this computation. So, you can see which is very small it depends on the clock speed of your processor.

(Refer Slide Time: 41:22)



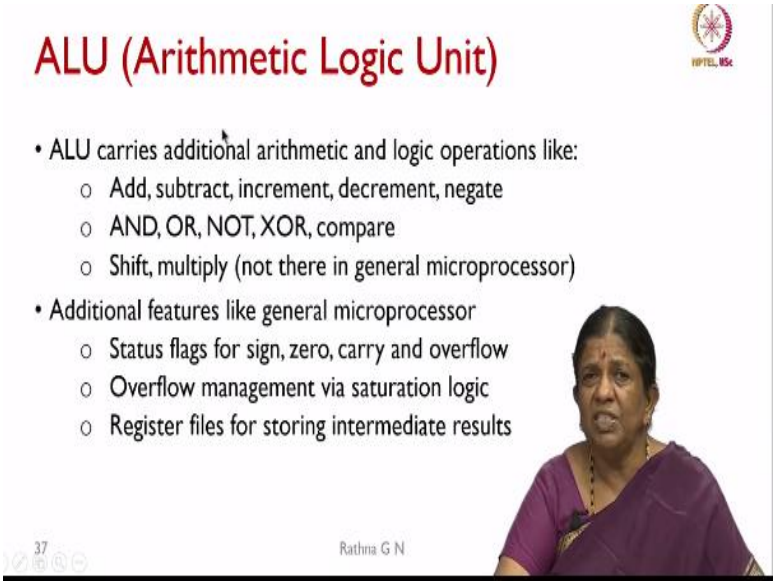
Coming to we said we need this scaling, we have taken care of in the multiplication by considering only fractional numbers so that the result of the multiplier is not going to overflow, but we have a constraint in the adder. So, how overflow and underflow can be taken care in the MAC operations. So, our addition or subtraction clause of overflow or underflow to take care of this, most of the DSP processor have 40 bits for as an accumulator, we call it as long addition what we can have it.

So, then 40 bits are going to be used, if it is a 16 bit to operands the maximum can be 32 + 1 carry bit, but we see we can accommodate the overflow up to 2^8 bits in this case. So, the barrel shifter at the input and output to normalize the value is what we need it and the other one is the saturation logic. We know that when there is saturation happens, if the maximum value is more than whatever the processor can represent, it goes as a negative value.

So, we want to restrict to that the maximum value, so the largest and then what we say is overflow accumulator has to stop. So, in that case for 16 bit processor, the maximum value of what we can represent a sign number what we are considering it, it is going to be $0 \times 7FFF$ in x which is nothing but 32767 and the minimum number what I can represent is $0 \times 8000 - 32768$. So, these are the values what has to be output if it is below this value or above this value so, this is taken care of in the hardware.

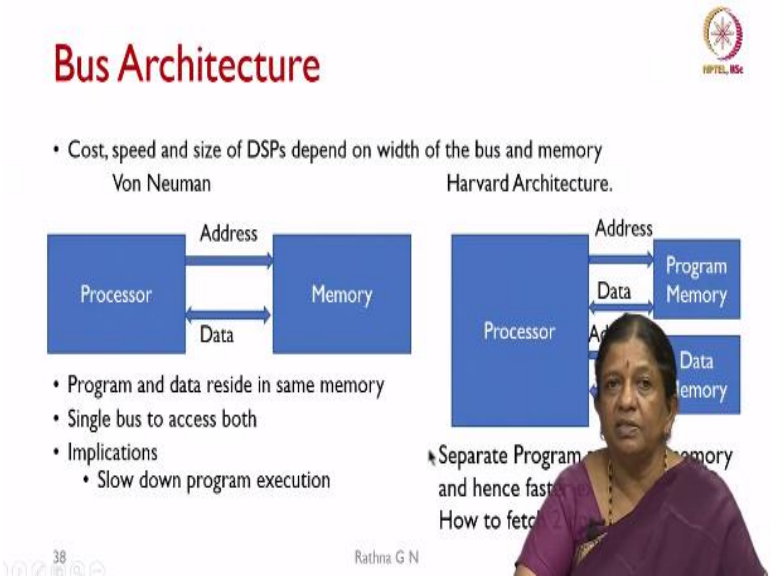
So, we have the least negative value and then the most positive value and you are taking the sign bit MSB and then you are going to consider whether it is going to exceed one of it you are doing the exclusive or so, you will be setting overflow or underflow bit is set then you will see that the maximum value is going or minimum value is going to be stored in the accumulator which is going to be output.

(Refer Slide Time: 43:53)



So, this is arithmetic logic unit apart from addition it has to take care of are the increment decrement negate or other operations and then shift multiply and additional features in the like other processes we should have a status flags for sign, zero, or carry or overflow. And then overflow management via saturation logic can be incorporated and registered files for storing the intermediate results.

(Refer Slide Time: 44:25)



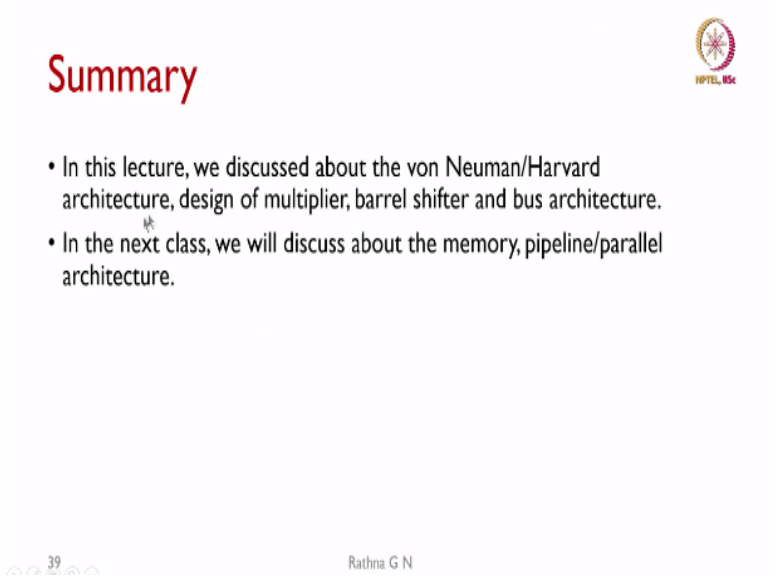
So coming to the bus architecture it depends on the cost, speed and size of DSP process basically width of the bus and memory. So, we have 2, architecture a lot of for normal CPUs use Von Neuman architecture whereas DSP processor use the Harvard architecture. So, in this case both address and data bus as you will be seeing that program and data reside in the same memory and single bus to access both your address as well as data, so which is going to slow down the program execution.

Whereas in this case, so we will be having program memory separate and then data memory separate. So, I can access the program from the program memory and then data from the data memory as we know that there are 2 operands what I need it for my multiplication and accumulation. So the coefficients which are already designed and fixed, I can store it in my program memory. And once the program fetch operand opcode has been fetched, I can release this bus and then get the data from here.

Make 5 lines per paragraph

So that is what, what it says separate program and data memory, faster execution that is what it says how to fetch 2 operands.

(Refer Slide Time: 45:50)



So in summary of this class, we discussed a little bit on von Neumann and then Harvard architecture, how we can design parallel multiplier, and then how we can do the shifting parallel shifting using barrel shifter. And then the bus architecture we saw it, so whether we have a $2n$ bus and n bus. In the next class we will discuss about memory, how we will be incorporating a pipeline in parallel architecture using hardware. So thank you. So we will meet in the next class.