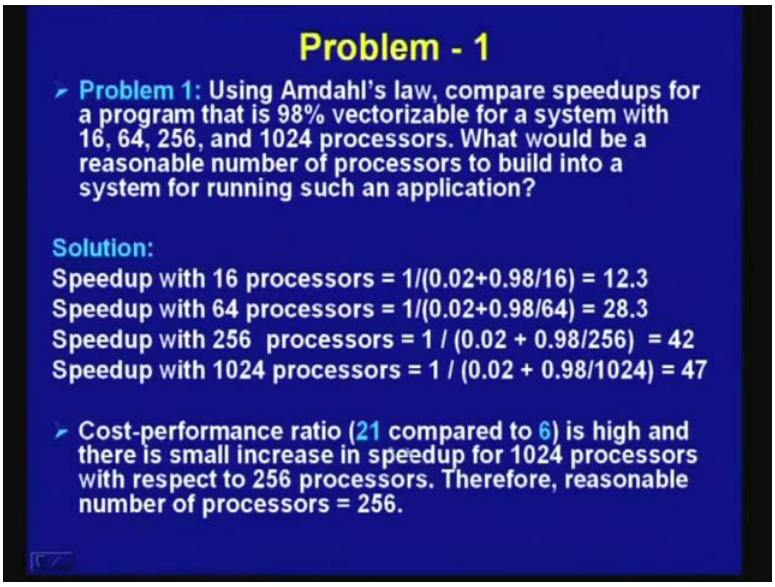
High Performance Computer Architecture
Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 20
Tutorial – 1

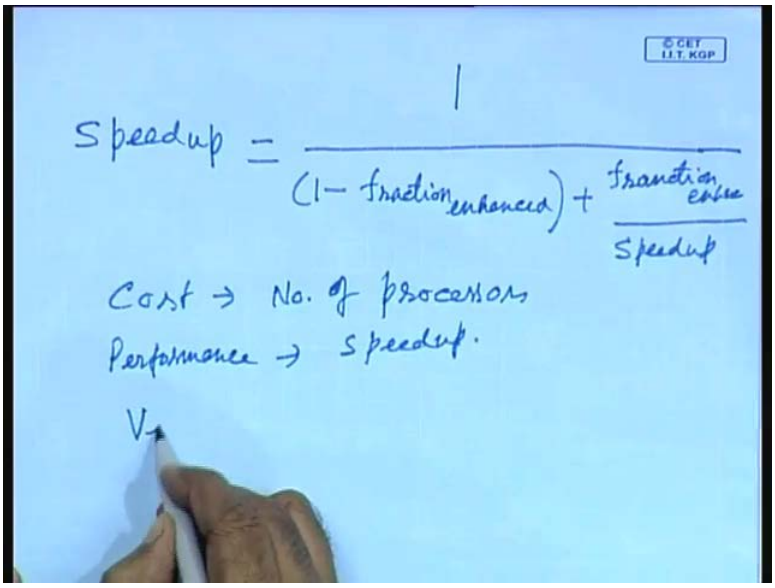
Hello and welcome to today's lecture, we have discussed various techniques for increasing the instruction level parallelism to enhance the speed of processor in the last 8 and 9 lectures. So, before we start, a new topic that is how you can enhance the speed of memory by using hierarchical memory organization, today we shall discuss about some problems, in this tutorial class.

(Refer Slide Time: 01:34)



So, first problem is using Amdahl's law compares speedups of a program that is 98 percent vectorizable, for a system with 16 64 256 and 1024 processors what would be reasonable number of processors, to build into a system for running such an application.

(Refer Slide Time: 02:05)



As you know the speed up can be expressed in the form of an equation, where speed up will be equal to 1 by 1 minus fraction enhanced, as you know a fraction of the program can be executed in enhance form. So, 1 minus fraction enhanced plus fraction enhanced and divide by the speed up, because of enhancement. So, here the speed up, that we are discussing maybe because of several reasons, the speedup may arise, because of you know pipelining, because of you know as you know, you can use parallelism, we have already discussed various techniques.

Whatever you do, that may not be applicable, to all parts of the program or you may use a special processor, like floating point processor. So, in such a case enhancement will take place, only in the small part. So, using this expression, we can do the computation, so if you consider 16 processors, it will be speedup will be equal to 1 by 0.02 that is 1 minus this 90 percent is vectorizable, vectorizable means you have got a vector processor, where you can execute in parallel, 90 percent 98 percent.

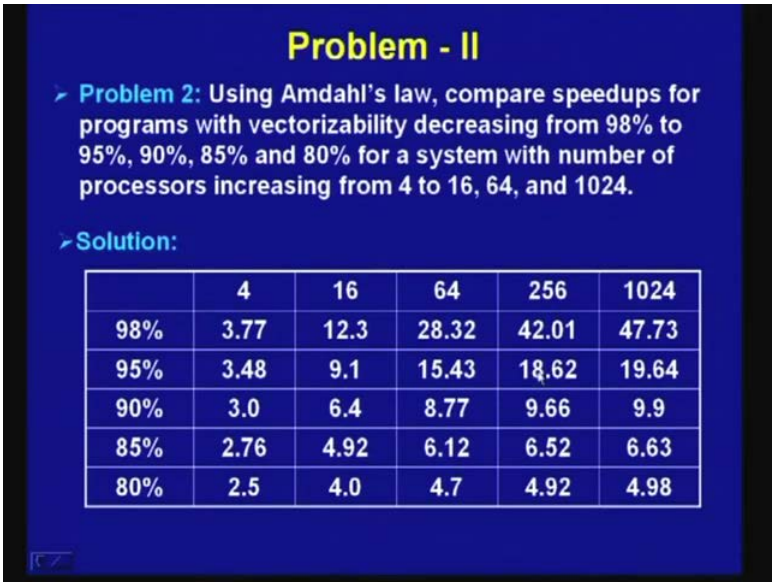
So, 1 minus 0.98 is 0.02 plus 0.98 by 16 where, 16 is the number of processors, you can use whenever you do the vector processing, so since you have got 16 processors, this will be 0.98 by 16. So, the speed up that you get is 12.3 by using sixteen processors, so in a similar way, you whenever you use 64 processors, you get a speedup of 28.3 and whenever, you use 256 processors, you get speedup of 42 and similarly whenever, you use one 1024 processors, you get a speedup of 47.

So, as you can see here, the cost performance ratio, that means the number of processors that you are utilizing is increasing, but the speedup of increasing and not increasing at that rate. So, if you consider, this cost performance, cost means, essentially in this particular case, cost represents number of processors and performance means, speedup that you have achieved by using, so many processors. If you consider that, you will find that, for the first case speedup is very close to 1, it is 16 by 12.3, because 16 is a number of processors and speedup is 12.3 16 by 12.3.

In the second case, it is 64 by 28.4 little less than 4 on the other hand, in this particular case, it is 256 processor you are using and 42 is the speedup. So, you are cost performance ratio is 256 by 42, it is close by 6. On the other hand in this case, whenever you go for 1024 processors, you get only a speedup of 47. So, where is marginal increase in speed up from 42 to 47, although the cost is increasing cost by cost performance ratio is quite high 21, in this particular case; that means, 01024 by 47.

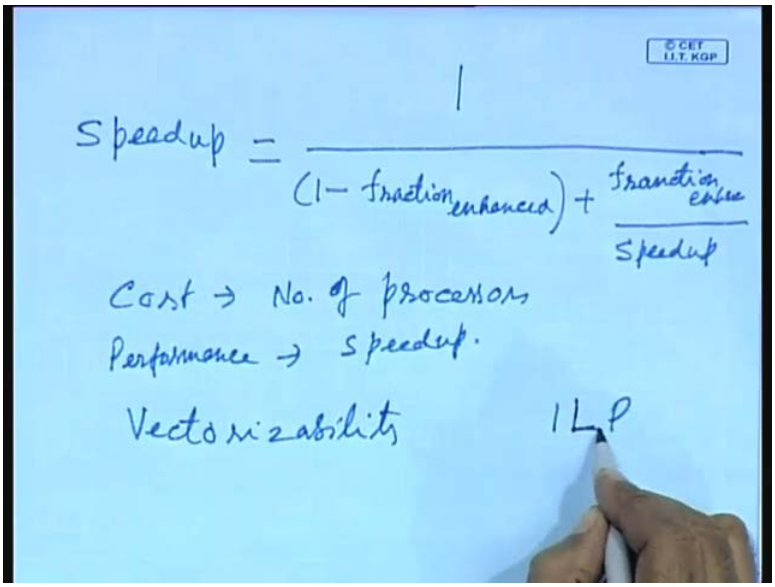
So, therefore, we find that, since the speedup is increase in speedup is marginal and cost performance is increasing significantly, whenever we go from 256 to 1024 processors. So, reasonable number of processors that, you can use in your system is 42, so you can conclude from this observation, so the second problem that, we shall be discussing is also related to this.

(Refer Slide Time: 06:53)



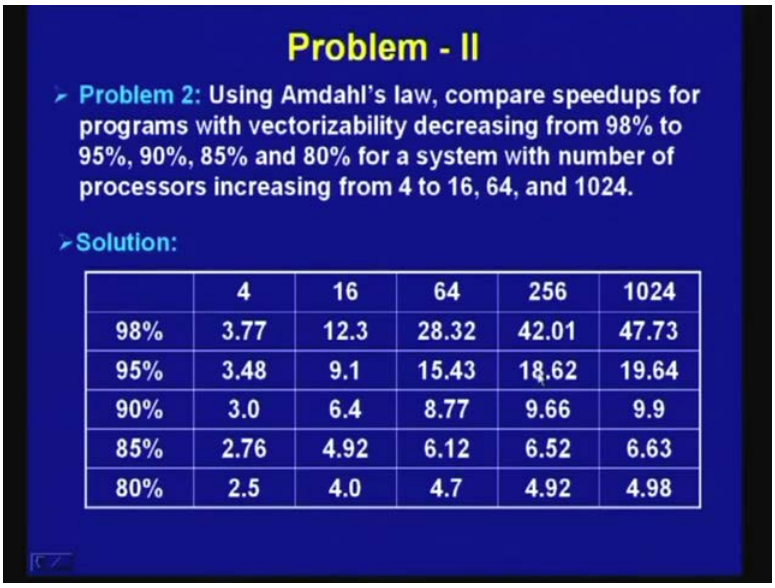
In this case using Amdahl's law compare speedups, for programs with vectorizability decreasing from 98 percent to 95 percent 90 percents 85 percent 80 percent. Here, the objective is to see, how these speedup changes as the vectorizability or later on, we can say in terms of your instruction level parallelism, you can say vectorizability.

(Refer Slide Time: 07:25)



So, or it can be instruction level parallelism, a kind of another parameter, I mean what is the parallelism that is exists in a program, so vectorizability and instruction level parallelism is somewhat, similar in notion. So, as you as the instruction level parallelism decreases, there also we will find that, you will not get good speedups. Similarly in this particular case, we are considering in a context of vector processors, where you can perform computation in parallel using a large number of processors. So, as the vectorizability decreases, how the performance degrades, even if you increase the number of processors.

(Refer Slide Time: 08:20)



	4	16	64	256	1024
98%	3.77	12.3	28.32	42.01	47.73
95%	3.48	9.1	15.43	18.62	19.64
90%	3.0	6.4	8.77	9.66	9.9
85%	2.76	4.92	6.12	6.52	6.63
80%	2.5	4.0	4.7	4.92	4.98

So, that is the study that, we can make from this problem, so we can see here, the vectorizability is decreasing, for a system with number of processors increasing from 4 to 16 to 64 and 0 1024. So, we shall be increasing the number of processors and for different cases of the vectorizability. So, solution is given in this case, in the tabulated form, so whenever the vectorizability is 98 percent and as you increase the number of processors, we can see the speedup is increasing, but obviously, the cost performance will be decreasing as you increase the number of processors.

So, for 4 processors, you are getting a speedup of 3.77, it is very close to 1 for 16 processors again little less than 1 and as we have discussed in a first problem and in this case as we have seen that, the reason will the number of processors that, you can use in your system is 40 is 256 and you will get a speedup of 42 40, roughly close to 42. Now, if the vectorizability decreases to 95 percent, you can see for the same number of processor the speedup is less, so speedup is less, but the speedup decreases more rapidly as we increase the number of processors.

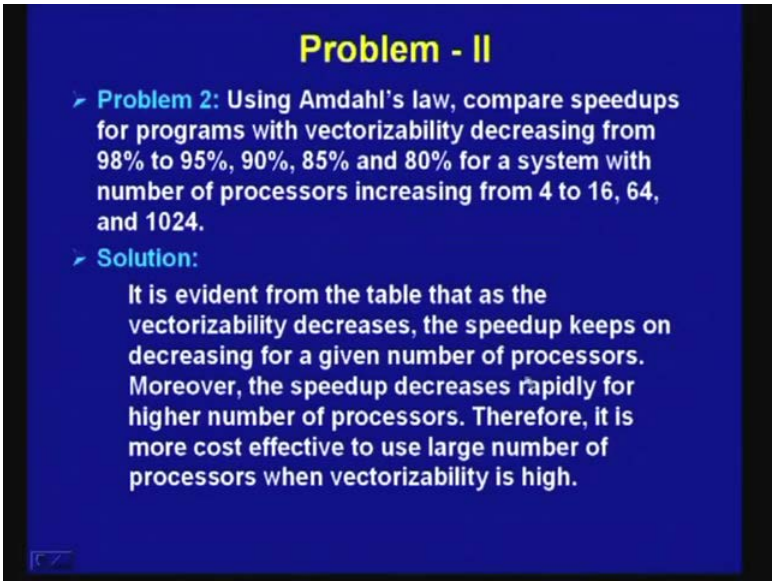
So, you can see here the speedup is 9.1 for 16 processors little more than half, I mean 2 is to 1, then for 64 processor, it is close to I mean, it is 4 is to 1, for 256 processor, it is less than 10 about, 12 is to 1. And so you can see the speedup is decreasing, compared to the previous case, similarly as you have decreased the vectorizability, similarly for 90

percent vectorizability, we see the speedup for different number of processors is increasing, but there is very small increase, as you go for larger number of processors.

So, as you increase the larger number of processors the speedup does not increase much and similarly for 80 percent increase, 85 percent vectorizability in a program. You find that speedup is 2.76 for 4 processors 16 for 4.92 processor 6.12 for 64 processor and you can see, if you even, if you increase from 64 to 256 processors, the speedup is only increasing by fraction; that means from 6.12 to 6.52. And again, if we increase the number of processors to 1024, the speedup will be only 6.63, so very small increase in speed up.

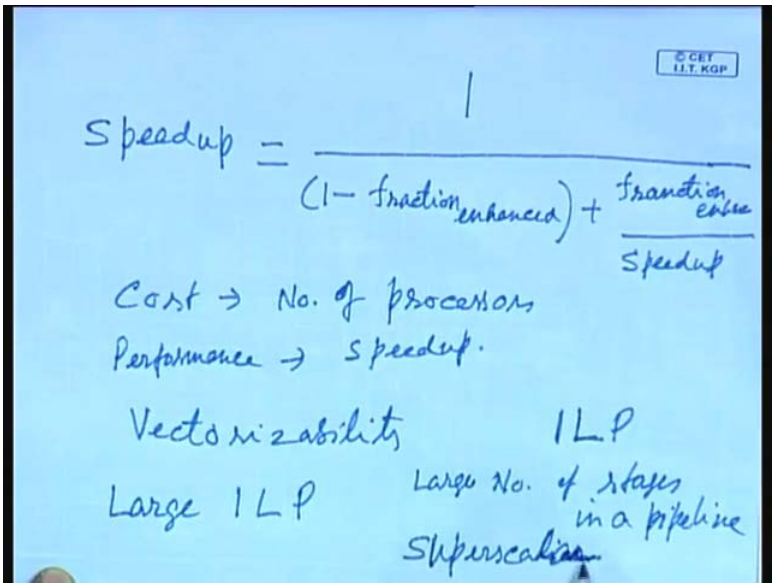
And similarly, for 80 percent vectorizability, situation is that much more worse, as you can see for 4 processors, we get a speedup of 2.5 for16 processors only 4 4 is to 1 cost performance for 16 64 processor little more than 4 and you can see it, does not exceed 5, even for 1024 processors. So, what is the lesson that, we learn from this observation.

(Refer Slide Time: 11:53)



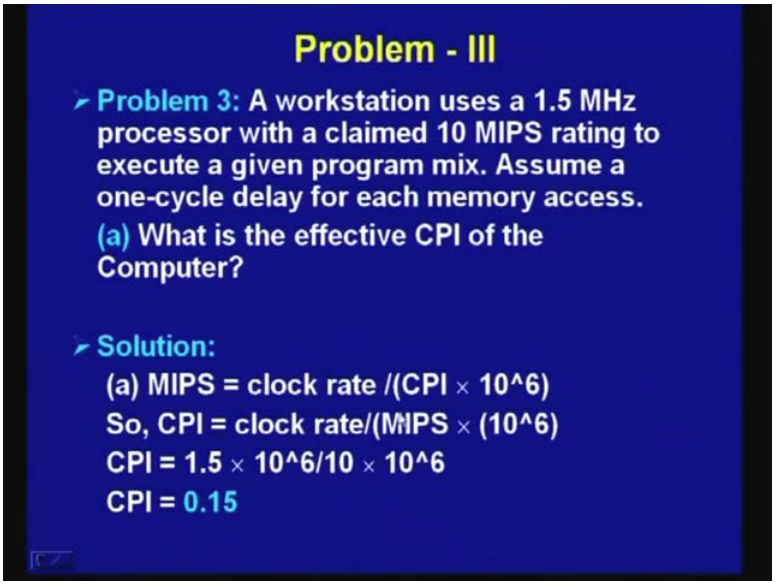
So, the lesson that, we learn from this observation is from it is evident from that table that, as the vectorizability decreases, the speedup keeps on decreasing, for a given number of processes that, we have seen. Moreover the speedup decreases rapidly for higher number of processors, therefore it is more cost effective to use large number of processors, when vectorizability is high, so that means, if we translate, it in terms of I L P.

(Refer Slide Time: 12:26)



So, unless you have large ILP instruction suction level parallelism, you cannot really go for, I mean large number of stages in a pipeline or if you are using superscalar processor then you should not increase. If you increase go for say superscalar processor then you should not increase the number of functional unit in the superscalar, if the instruction level parallelism is not high, that means, that is the reason why. You will see the number of functional units used in superscalar processors does not exceed beyond 6 typical values are 456 not more than that and this is what we have learned from this problem.

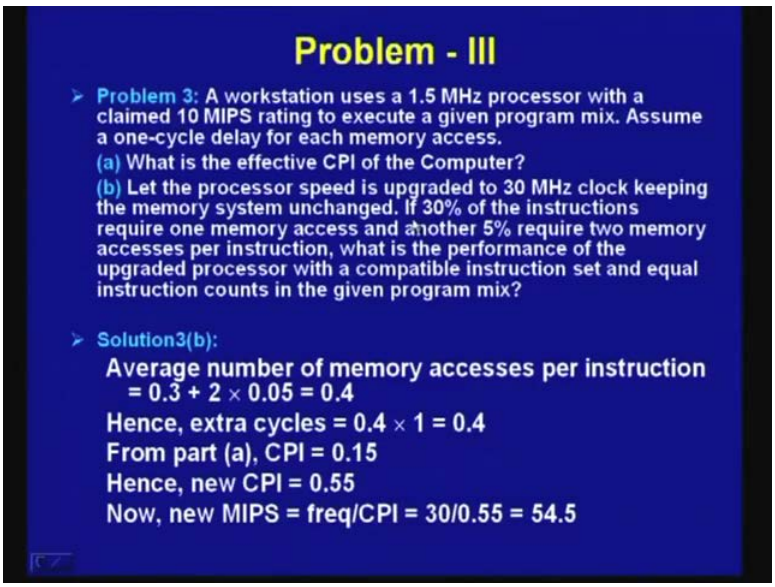
(Refer Slide Time: 13:29)



Let us now switch to third problem, in this case a workstation uses 1.5 mega hertz processor with a claimed MIPS rating of 10 MIPS rating, to execute a given program mix assume a 1 cycle delay, for each memory access, what is effective CPI of the computer, CPI of the processor. You can see in this particular case, the MIPS can be expressed as clock rate by C P I into 10 to the power 6, also CPI is equal to, so you can therefore, CPI that you get is clock rate by MIPS into 10 to the power 6 and by substituting different values, we get 1.5 into 10 to the power 6 by 10.

So, into 10 to the power of 6, that is on y only 1.15, so CPI is equal to 0.15, what does it mean, it is a superscalar processor and since it is superscalar processor, that is why you are getting a CPI of less than 1. So, CPI of less than 1 can be obtained, for superscalar processors and in this case, you get CPI of 1.5 and that is a reason, why you know, in case of CPI, in case of superscalar processor and normally instead of stating in terms of CPI, it is stated in terms of instruction per cycle IPC and that will be more than 1.

(Refer Slide Time: 15:09)

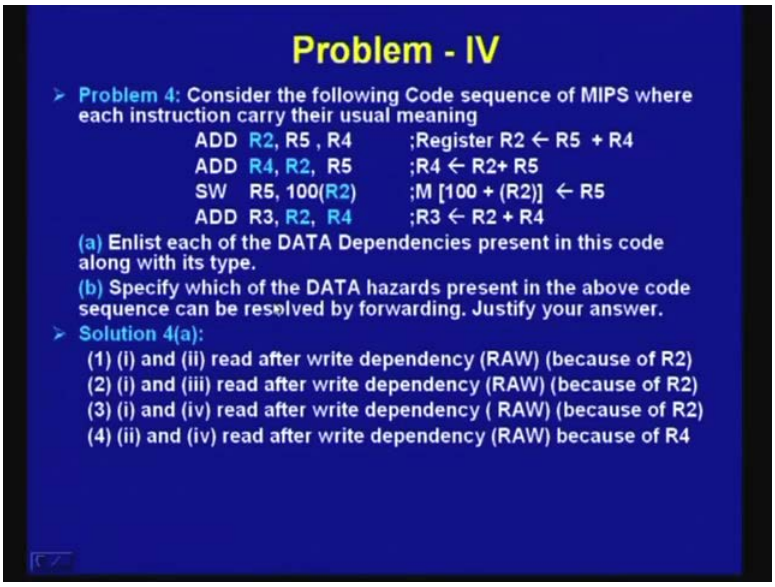


Let us now go to the second part of the problem is, let the processor speed is upgraded to 30 mega hertz clock keeping the memory system unchanged, if 30 percent of the instructions require 1 memory access and another 5 percent requires 2 memory accesses per instruction. What is the performance of the upgraded processor with the compatible instruction set and equal instruction count in a given program mix.

So, in this particular case, what is happening, you have increased the speed of the processor and however, using this value the 30 percent of the instructions require 1 memory and 5 percent requires 2 memory. So, if we substitute it, here that is your vocalist average number of memory accesses per instruction will be 0.3 into plus 2 into 0.05 that is 0.4, so number of extra cycles will be 0.4 0.04 into 1. So, if we had this with 0.15, we get a CPI of 0.55.

So, that means, were adding with the previous case, whatever we got that is your 0.15, now because of the memory accesses, there is a delay and so your CPI is now 0.55. So, with this CPI, we get a MIPS rating of 30 by 0.55 that is your 54.5, so that is the MIPS rating that, we get in this particular case. So, we can see although, it is a superscalar processor with a large, I mean quite large degree, we are not getting much benefit, because of the delays in the memory.

(Refer Slide Time: 16:55)



So, now let us switch to problem 4, in case of problem 4 consider the following code segment, code sequence of MIPS, where the each instruction carry their usual meaning, add R 2 comma R 5 comma R 4. So, here you are writing the value into add 2 by adding the content of R 5 and R 6 and so it in the second instruction again, you are adding the value of R 2 with that of R 5 and storing, it loading it in R 4. And here, it is a store word you are storing the content of register R 5 into memory location, computed by adding 100 with the content of R 2.

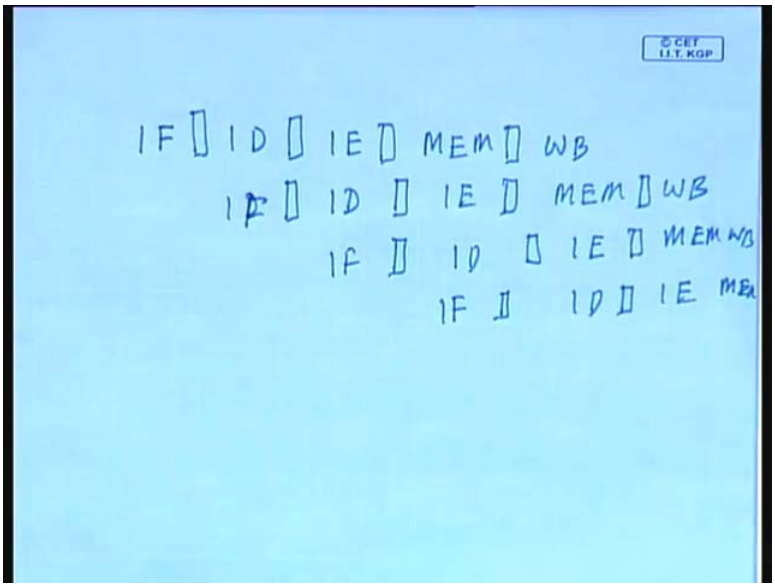
So, you find that and 4th instruction is again an add instruction, which is storing the value of the content of R 2 and R 4 in register R 3. Now, we find you have got a number of the question was enlist each of the data dependencies present in this code along with it is type, specify which of the data hazards present in the above code sequence can be resolved by forwarding, justify your answer. Now, you see this instruction one and instruction 2, these 2, they have read after write hazard, because you know, you are reading after this write.

So, there is read after write hazard busy between the instruction 1 and instruction 2 similarly, instruction 1 and 3 also has a has got a read after write hazard. Because, you are using R 2 to compute the effective address, that is used for the purpose of storing in memory location, I mean in memory, in content of R 5 is being stored in a memory location computed by using R 2 100.

So, again this is a read after write hazard similarly the 4th instruction is also having a read after write type of hazard, because of R 2 R 2 and your writing value of R 2 in instruction 4 and there is another hazard present here, you see that, between R 2 and instruction 2 and instruction 4. So, that content of that R 4, you have loaded in I mean, the computed by using the content of R 2 and R 5 and stored it on in R 4 and R 4 is being used instruction 4.

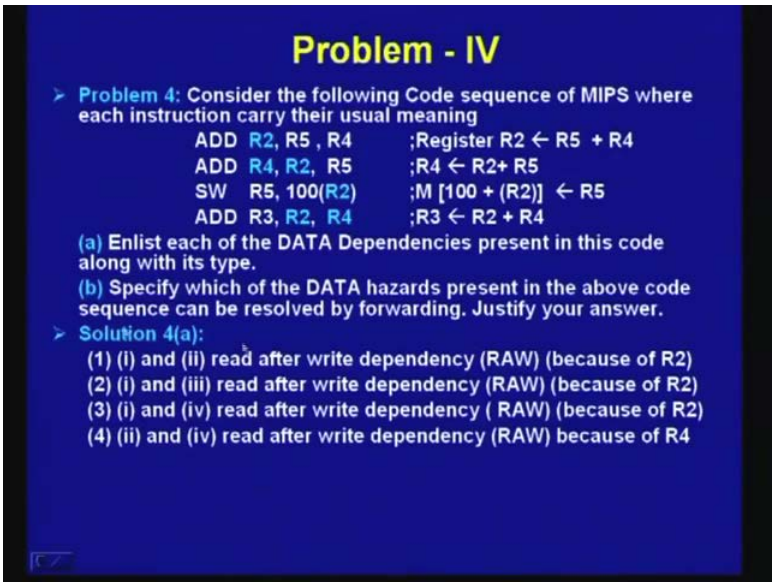
So, you find, there are 4 read after write type of hazards, the question naturally arises, which of them will lead to hazard will all of them lead to hazard or some of them will lead to hazard and then we can answer the second part. So, let us see, which of them will lead to hazard, let us consider the 4 instruction.

(Refer Slide Time: 20:08)



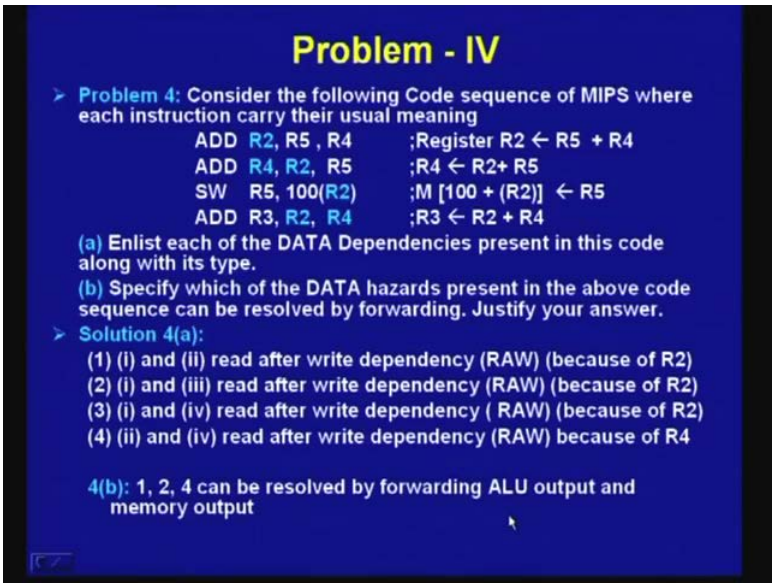
That you know, that you can write it, in this way instruction fetch, instruction decode instruction execution then memory access and write back and second instruction decode, instruction fetch, instruction decode, instruction execution, memory and write back. Third is instruction fetch, instruction decode, instruction execution then your memory and write back and this is the 4th instruction fetch, instruction decode, instruction execution, memory operation and write back. Now in between, you have got the registers pipeline registers present in all the cases, so here also you have got pipeline registers present in all the cases.

(Refer Slide Time: 21:19)



Now, the question was enlist each of the data dependencies present in this code specify, which of the data hazards, present in the above code sequence can be resolved by forwarding. First of all, let us find out, I mean we have already considered the data dependencies and now, which of them can be resolved by using forwarding, let us see.

(Refer Slide Time: 21:40)



Now, whenever this instruction execution is going on, this instruction execution has already taken place, so directly you can provide the data by forwarding from here to here. So, the first hazard, as you can see can be overcome by first the hazard that will arise will overcome by using forwarding, then 2nd 1 here, 2nd 1 again can be overcome by using forwarding, so you can see by using forwarding, this can be overcome.

Now, the 3rd hazard that, you have already discussed that between 1 and 4, read after write although, there is a dependency, but by the time that instruction execution is taking place data is now is already available in the in the register. So, you can directly feed it from here to the 4th instruction, so you can see here, this will this 4th instruction will not lead to any hazard, so there is no need for forwarding in this case.

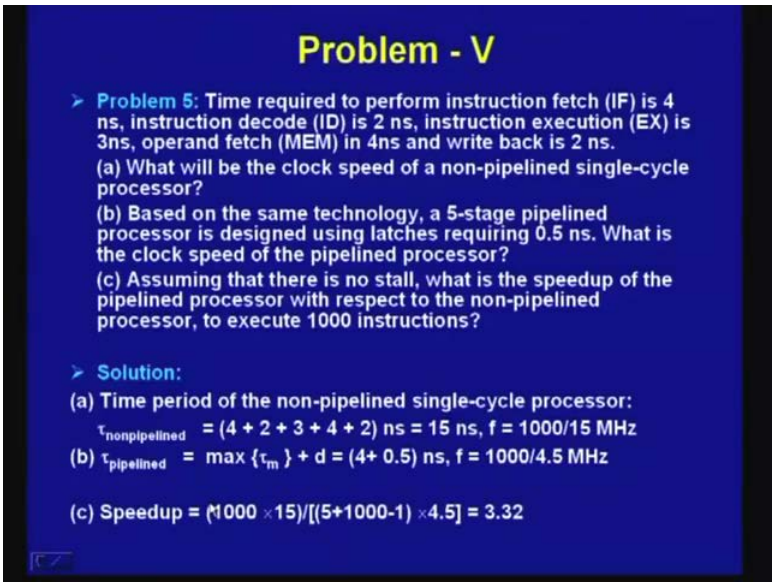
So, only 1 and 2 will require forwarding 1 and 2 can be resolved by using forwarding, the 3rd one need not be resolved by using forwarding, because already, the data has been written in to the register, it can be written from the register itself. So, there is no need for, reading it from the pipeline register, but it can be read from the architectural register, that

is present in the ALU, so it can directly read from R 2 itself, so it will come to the can be provided to the instruction.

Now, let us come to the 4 last one between 2 and 4, you see your 2 and 4, you are here, you will be reading, so that now the forwarding can be done, because it is available here, reading will take place here. So, you can from the, from this stage, you will do the forwarding and it will go to the instruction execution stage, so you will require 3 forwarding to resolve the hazards and that is what I have written 1 2 4 can be resolved by forwarding ALU output and memory output.

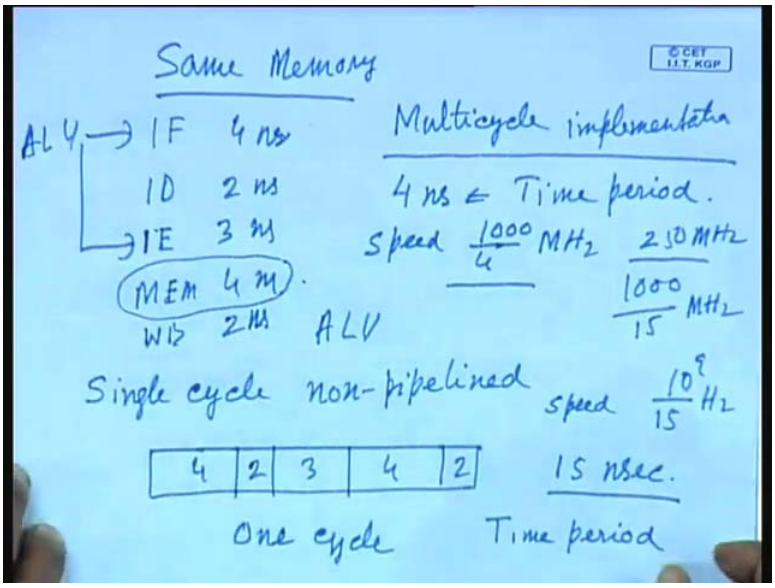
However, the 3rd one does not lead to any hazard, so this was this is the problem that deals with the dependencies and hazards that can be resolved by using forwarding. So, forwarding is a expensive hardware that is used, but this is very useful from this, as it is clear, it can overcome lot of hazards with the help of the forwarding unit.

(Refer Slide Time: 24:42)



Now, let us come to the problem 5, time required to perform instruction fetch is 4 nanosecond, instruction decode is 2 nanosecond, instruction execution is 3 nanosecond, operand fetch in memory is 4 nanosecond write back is 2 nanosecond. So, what has been done here, we are considering a processor.

(Refer Slide Time: 25:11)



Where the instruction fetch can be done in 4 nanoseconds, instruction decode can be done in 2 nanoseconds, instruction execution can be done in instruction execution can be done in 3 seconds then memory operation can be done again in 4 nanosecond. Because, instruction fetch is reading from instruction memory and this memory operation is usually, involved with that data memory. So, again four cycles will be required and then final operation that is your operand fetch or write back you can say, in is performed in 2 cycles.

So, these are the, this is the value, that is given in your that can be performed, now whenever you implement single cycle, non-pipelined processor then what you try to do, you execute, you perform all the operation in 1 cycle. That means, the instruction fetch maybe, it is taking 4 cycles instruction decode 2 cycles then 3 cycles 4 cycles. So, 4 2 3, I mean, nanosecond 4 nanosecond 2 nanosecond 3 nanosecond 4 nanosecond and 2 nanosecond.

So, the they will be considered in one cycle, since this is the single cycle then it will require, how much time, it will require 4 plus 2 plus 3 plus 4 plus 2 that means, total number of cycles that, you require is 10 plus 15 nanosecond. So, 15 nanosecond will be the time period, of the single cycle implementation, however, the clock frequency, that is the speed, which is specified in terms of clock frequency will be 1 by 15 into 10, the power 9 hertz or you can say 1000 by 15 mega hertz.

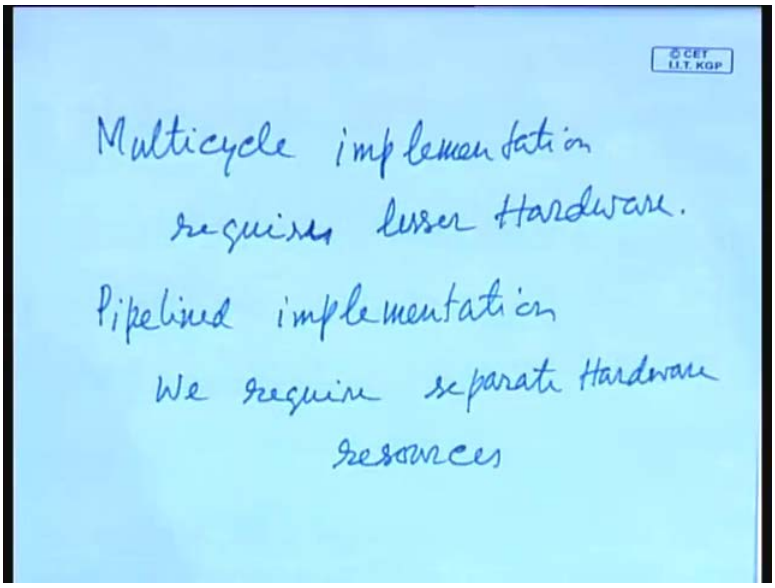
Now, this is for single cycle implementation, now let us assume that, we are interested in multi-cycle implementation that means, each of these operations instruction fetch, instruction decode instruction execution memory and write back all are performed in different cycles, in a multi-cycle implementation. So, as you know, there are 3 possible alternates, single cycle, multi-cycle then pipeline, so before we consider pipelining, pipelined in multi-cycle implementation, what will be the number of cycles required.

Number of cycles will be the clock frequency will be decided by you know, that the time period of the clock will be 4 nanosecond, that is the time period and the frequency clock frequency, that means, your speed will be 1 by 4 that is 1000 by 4 mega hertz, that is your 250 mega hertz. So, whenever we go for multi-cycle implementation this instruction fetch, instruction decode, instruction execution memory and write back all these cycles may not be required to execute in different programs.

For example, whenever you are using ALU operations, if it is a load store architecture and memory operation is performed separately by using, load store architecture and ALU operations are performed separately. In such a case, this particular clock cycle will not be necessary, whenever it is ALU type instruction on the other hand, if it is a memory type, then it will be required. So, in such a case all the 4 cycles 5 cycles may not be required, in different instructions to execute different instructions.

However this speed will the clock frequency will be 250 mega hertz and if all the cycles are required, total time will be longer then single cycle implementation. However, you may be asking, what is the benefit of multi cycle implementation, the benefit of multi-cycle implementation is the hardware resources, that is required can be less. Why the hardware resources that is required can be less, for example, the you can use same memory, there is no need for 2 separate memories, you can use same memory, whenever you go for multi-cycle implementation. So, multi-cycle implementation will not require 2 separate memories, similarly you will not require 2 ALU, for these 2 operations.

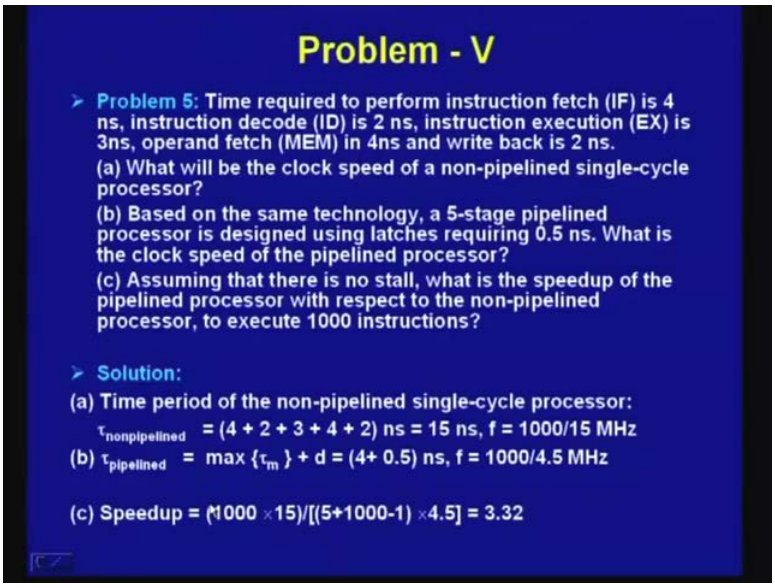
(Refer Slide Time: 30:53)



So, that is why multi-cycle implementation requires lesser hardware, so that is the reason, why multi-cycle implementation is also popular and compared to non-pipelined, because in non-pipelined all of them are performed in single cycles, so you will require separate hardware resources, for performing all these 3 operations. Similarly whenever we go for pipelined implementation, we require separate hardware resources, why have you require different hardware resources, because you will see different instructions will be executed in a overlap manner.

So, you will require to separate memories for 1, for I mean 1 for instruction and another for data, you will require 2 separate functional units maybe, for instruction execution and for computation of whenever, you perform the here, you will perform the computation of that P C plus 4. So, you will require separate hardware in your pipeline implementation, that means, in the, if you consider the hardware requirement, the single cycle and pipeline implementation will require more hardware. And regarding the clock frequency, for pipeline implementation, the for pipeline implementation the clock frequency will be same as the multi-cycle implementation. So, that way it will be a clock speed wise, it will be same, let us see the answer for this.

(Refer Slide Time: 33:09)



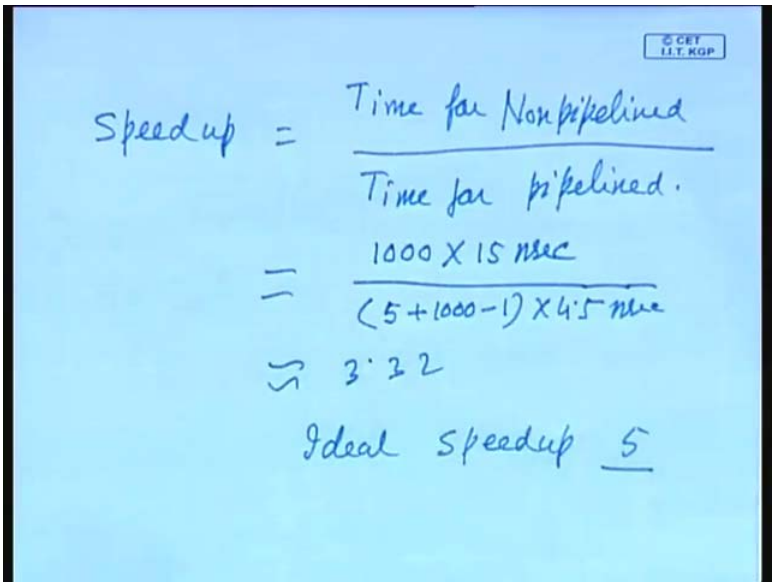
So, what will be the clock speed, for non-pipelined single cycle processor, based on the same technology 5 stage pipelined processor is designed using latches requiring 0.5 nanosecond, what is the clock speed of the pipelined processor. So here, additional delay will be required, because of the latches; that means the I told you that, the clock frequency will be same, for multi-cycle implementation and pipeline implementation.

But that is not true, if we do not assume that, the latches will have 0 delay, if we assume non 0 delay for latches, then the speed of pipeline processors, for the clock frequency of pipelined processor will be more. So, you have to and the delay of the latches and assuming that, there is no stall, what is the speed up of the pipelined processor with respect to the non-pipelined processor to execute 1000 instructions. So, this is your problem, we have already discussed different 3 different situation, now let us go to the solution.

So, time period for the non-pipelined single cycle processor will be 15 nanosecond, as I have told, so the frequency will be 1000 by 15 mega hertz, similarly for pipeline processor, it will be equal to maximum of the stage time. So, in this case maximum is 4, so you will require 4 plus delay of the latch, which has been given as 0.05 that is, that means, equal to 4 plus 0.05. So, you will require a frequency speed of 1000 by 4.5 mega hertz, that is that will be the speed of operation, for this particular pipelined implementation.

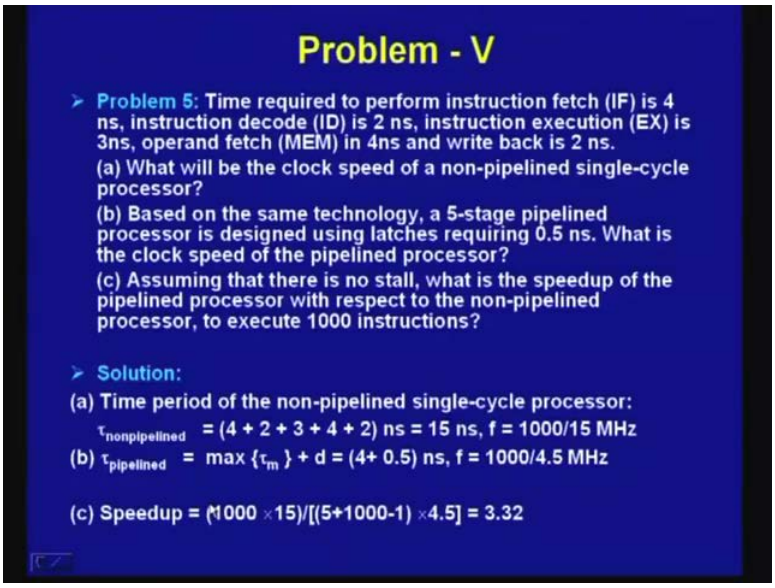
And as I have already told for, if it is multi-cycle implementation then speed will be thousand by 4, now speedup is will be equal to 1000 into 15.

(Refer Slide Time: 35:18)



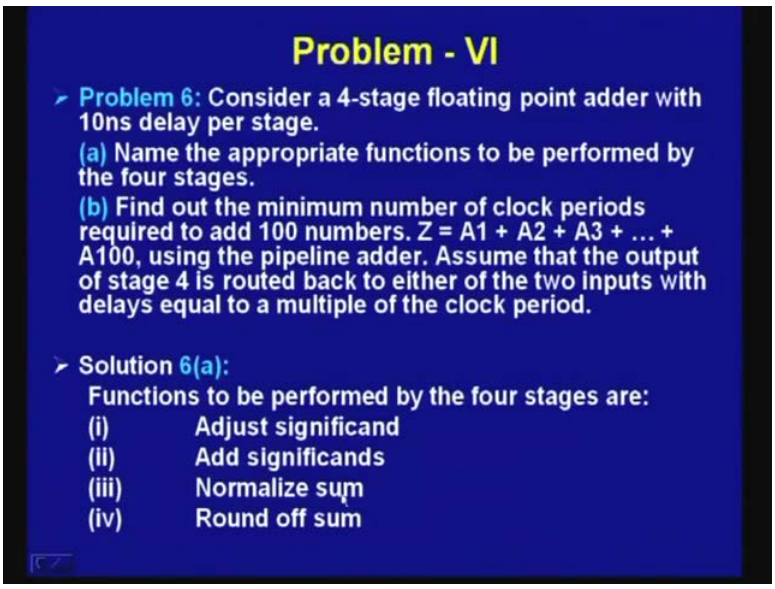
That is the we know that, speedup is calculated in this way time for execution, for non pipelined and by time for execution time for pipelined, in our case the time required execution time required for non-pipelined will be equal to we are executing 1000 instructions. So, 1000 into 15 nanosecond, so that is the time required, for non-pipelined implementation on the other hand time required, for pipeline implementation will be equal to 5 plus 1000 minus 1 k plus n minus 1, as you know into 4.5, that is the time period of the I mean, the period of the pipelined processor. So, this value is roughly equal to 3.32, so you can see in this case the speedup, ideal speedup is 5 instead of 5, we are getting 3.32.

(Refer Slide Time: 36:42)



So, we are getting a speedup of 3.32, even though there is no stall, but because of the increase in you have to consider clock frequency corresponding to the delay of the largest stage, I mean maximum stage and also, because of the delay of the latch. So, you see, even without stall, your pipelines implementation will give lesser speedup, now let us switch to problem 6.

(Refer Slide Time: 37:16)

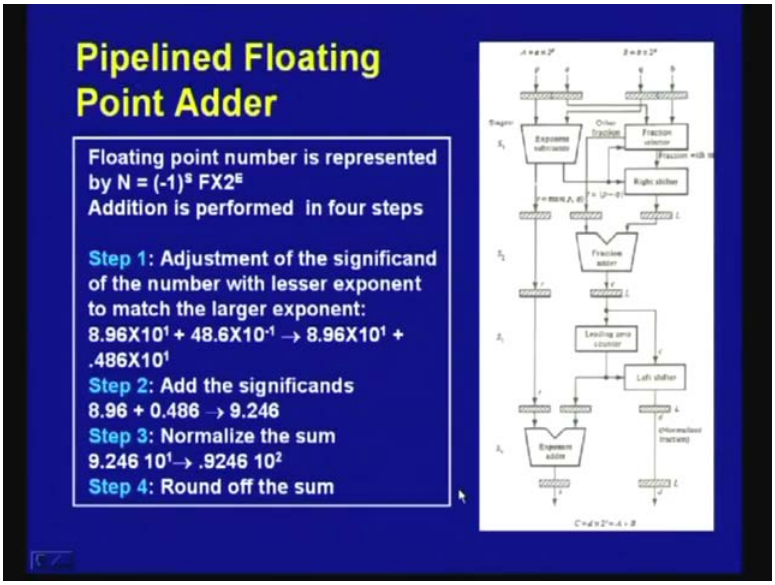


So, consider of 4 stage floating point adder with 10 nanosecond delay per stage name the appropriate functions to be performed by the 4 stages, find out the minimum number of

clock periods, required to add 100 numbers, that is Z is equal to A 1 plus A 2 in this way, you have got 100 numbers using the pipelined adder. Assume that, the output of stage 4 is routed back to either of the 2 inputs with delays equal to multiple of the clock period.

So, I have already discussed about 4 stage, I mean floating point pipelined adder and as we have seen the functions to be performed by the 4 stages are given here, number 1 is adjust significant, you have 2 adjust the second significant corresponding to the exponent of the largest significant, I mean to the 2 exponents will not be same. So, you have to adjust the significant corresponding to the number having higher exponent then you have to add the significant in the second stage. In the third stage, you will normalize the sum and in the fourth stage, you will round off the sum, if you have got leading 0s. So, these are the 4 operations to be performed in the 4 stages.

(Refer Slide Time: 38:47)

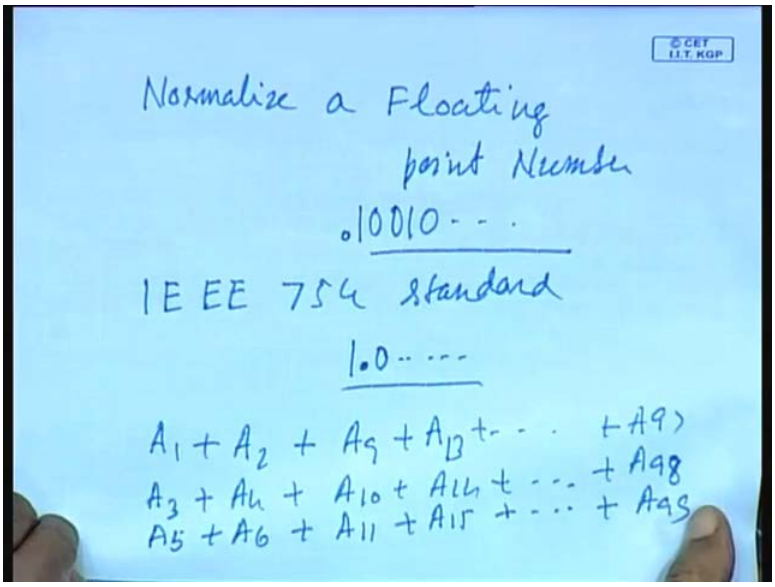


I have already discussed about, these in my in one of my previous lectures and with the help of an example. So, how the adjustment of the significant is required here, for example 2 numbers although, we have considered decimal numbers situation will be same, for binary numbers, you are adding 2 numbers, but here you have to adjust the significant for example, in this case it is minus 1.

So, the significant has been adjusted, for this to make the exponent same 10 to the power 1 then you will be adding the significant to get the sum of the significant, now you may

be asking 1 question. Sometimes, we use a term called fraction for significant, why you are using the name significant, conventionally.

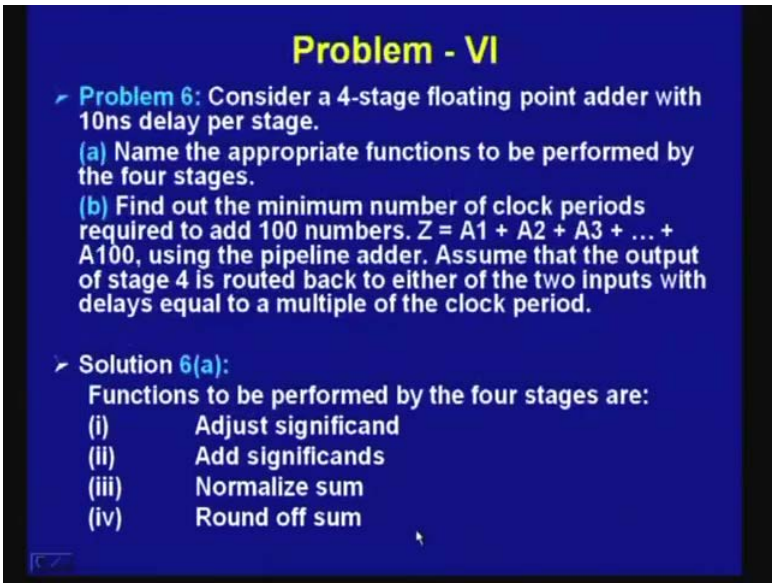
(Refer Slide Time: 39:50)



Whenever we normalize a floating point number, usually you shift in such a way that, you start your number starts after the decimal point, it can be 1 0 0 0 1 0 that means, after the decimal point, it is 1, I mean, I am considering the case for decimal numbers. So, then you have got other numbers, so this is how normalization is done, but there is a I triple E standard, I triple E 7 5 4 standard, where the where after normalization, the value is 1.0, that means, it is one before the decimal point and that means, your shifting by 1 and remaining numbers can be here.

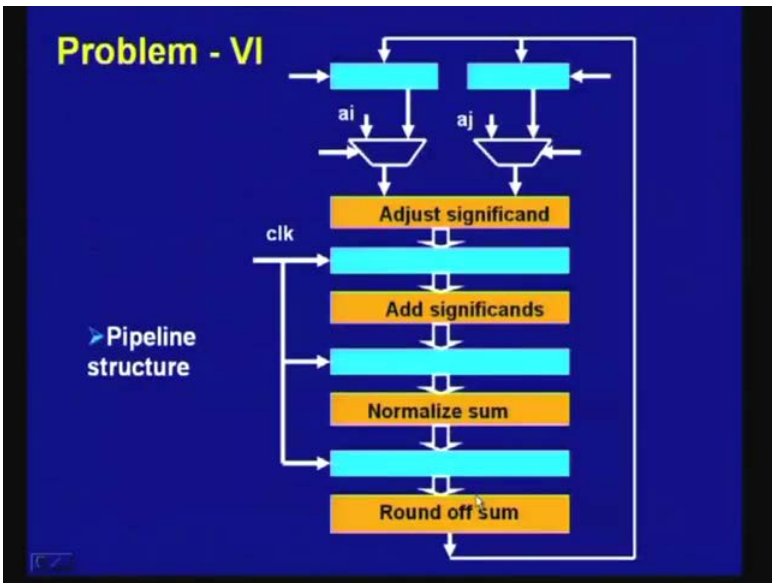
Question is why this has been done I triple E 7 5 4, the reason for that is by considering this, you are getting one additional bit in your fraction and as a consequence resolution is increasing and to increase the resolution that is used. So, in such a case, we cannot really call it a fraction, we have to give a separate name that is why, it is called significant. Anyway, so these were the 4 stages that, we have discussed, however our pipeline has to be little different compared to the pipeline, that is been shown in this diagram, because as you have seen.

(Refer Slide Time: 41:40)



Our requirement is assumed that the output of stage 4 is routed back to either of the 2 inputs with delays equal to a multiple of the clock cycles, to achieve this, we have to modify the pipeline little bit.

(Refer Slide Time: 41:56)



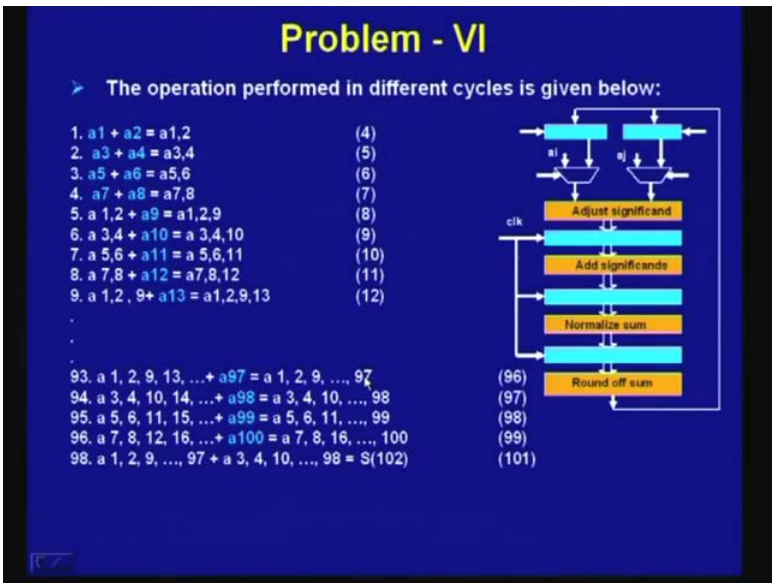
And the modified pipeline is shown in this diagram, here what has been done, the 4 stages are here adjust significant adjust add significant normalize sum, round off sum, the traditional stages that is required for hiding floating point numbers are given here. And these 3, these are, these blue-colored, I mean these are is essentially the latches, so

latches are shown in between different stages, now you see the output is routed back to the input. However, you will require to separate latches for the 2 operands and then they are routed through multiplexers.

So, in the multiplexers, you can feed either the number that is coming, that means, A 1 to A 100 that, you can feed or you can take the output and you can, that means, that is latched, that is been stored in these latches. They can be fed to this through this multiplexers that means, one of the operand can be say A 1 and 2 operands that can be fed is 1 is A 1 another is A 2 or it can be A 1 is fed here. And a number, I mean the output of the pipeline, that can be fed back and that will go through, this multiplexer to one of the as operand to the floating point pipeline adder.

So, in this way, you can do that or what can happen, you can set the output in 2 different latches and then both of them can be fed, to the input of the pipelined floating point adder, through this multiplexers. So, to add this, I mean to achieve this flexibility, we have added, these multiplexers and obviously, they will require separate clocks for latches and control signals, for these multiplexers, in other words the controller of this floating point adder will be little more complex.

(Refer Slide Time: 44:19)



Now using these type of pipelined adder, let us see how the computation is being performed. So, the operation performed in different cycles are given here, on the left hand side, you have got the clock numbers 1 2 3 4, so in the first 4 clock cycles, you are

adding the numbers A 1 plus A 2, we are generating partial sum, then we are adding A 3 and A 4, which can be done, as I told by feeding here, A i and A j, that means, A 1 A 2 is fed here and they are selected and applied to the pipeline adder.

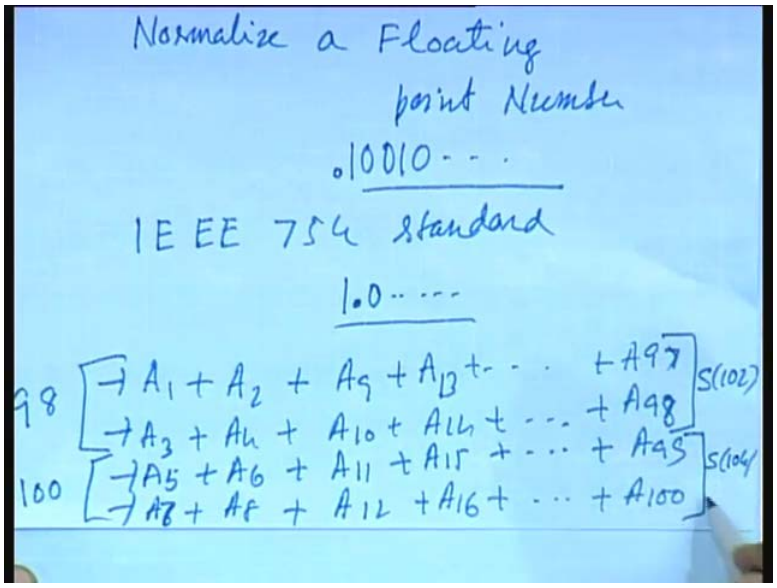
So, this is how the first input is going then the second input is going and after at the end of 4th cycle, you will get the output, the output will be available from this stage, at the end of 4th cycle and that is being shown here, on the on this right side column. Similarly A 3 plus A 4 is fed in the second cycle and output will be available in the 5th cycle A 5 and A 6 is fed in the 3rd cycle, output is available in the 6th cycle and a 7 plus 8 A 8 is fed in the 4th cycle and output is available in the 7th cycle.

Now the control will be changed and in the 5th cycle, what will be done 5th 6th 7th, in the remaining cycles, what will be done, one input will come as you have seen in the 5th cycle. So, output is generated in the 4th cycle, which is stored in this latch, which has been latched here and that will be fed to 1 of the inputs, in the 5th cycle. So, this A 1 2 A 1 2, that means the result of the addition of the number A 1 and A 2, that is been fed to 1 arm of the adder and A 9 is fed to another arm, that means, A 9 is fed here and a that a 1 2 is fed from here to through this multiplexers, through another input.

So, this is how in each cycle will be feeding 1 input and partial sums will keep on accumulating in the pipelined registers and they will be keeping, they will keep coming to these inputs, where they will be latched and they will be used as input, as it is shown in different cycles. So, in this way cycle 5 67 8 9 and it will continue, till clock number 96, so till clocks number 96, it will continue and where, you will be feeding 1 input from the output of the of the pipeline.

And another input will come from this, from the numbers that is been given, so in this way, it will continue and you can see, these 4 numbers, I mean partial sums you can say, that in clock cycle 96. This is been available and in clock cycle 98, this is been available, that means, this partial sum, that means, the partial, you can see, that the addition of numbers, has now has is grouped into 4 and those 4 groups will, now available in the pipelined registers.

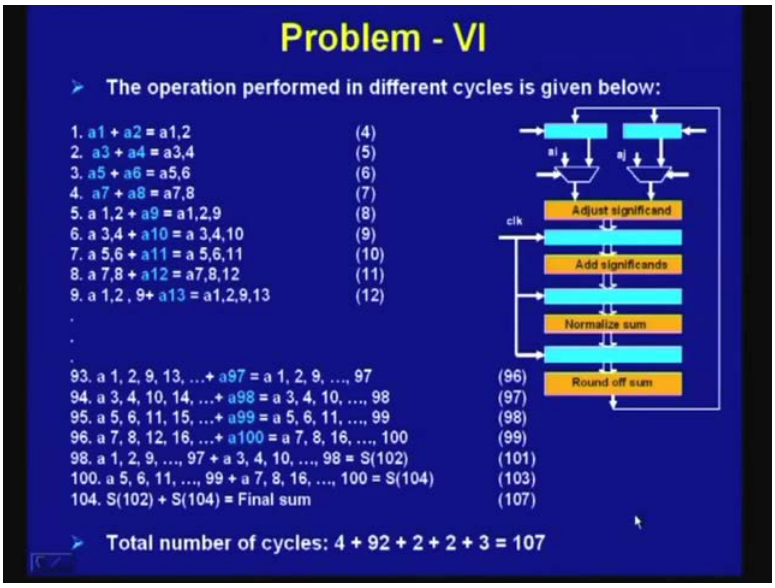
(Refer Slide Time: 47:49)



So, in one group you have got addition of A 1 plus A 2 plus A 9 plus 9 plus 413, in this way, it will continue till A 97, then another group, that means, this is one sum and another will be A 3 plus A 4 plus A 10 plus A14, in this way, it will continue till A 97 98. 3rd group is a A 5 plus A 6 plus A 11 plus A 15, it will go to A 99 and the 4th group will be, so these are the partial sums, which are available in the pipeline registers. So, A 6 plus A 7 plus A 8 plus A 12 plus A 16 plus in this way, it will continue till A 11 100.

So, you can see, now all the numbers are added, but they are available in different pipelined registers, so this is available in the 96 cycles and this is available in 97 cycl,e this is available in a 98 cycles, this is available in 99 cycles. So, now, you have to add these 4 numbers, how these additions will be carried out.

(Refer Slide Time: 49:20)



So, to perform the addition of these 4 numbers, what you have to do you have to wait for multiple of cycles, because you have to load the first number, this number and second number. So, one will be available in this latch another will be available in this latch, so this output will be available, so in the in one cycle, it was latched here in another cycle, it was latched here. So, in 2 latches, these are available and then by selecting multiple flexors, you will be it will be fed to the to this pipelined adder.

So, you can see in the in 98 cycles, you will be able to start adding the first 2 partial sums, that means, first 2 partial sums addition will start in a 98 cycle, similarly addition of this, can start in the 100 cycle, because output is available in the 100 at the end of 100 cycles. So, you can see in 100 cycles, this is been started and results will be these 2 results will be available in 102 cycles, since it is starting at 98, it will be available in the 100 and first cycle and this will be available in the 103rd cycle and these partial sums has been shown as S 1 0 2 and 1 0 4.

So, these 2, now you have to add and which can be fed to the pipelined only at the end of 1 0 4 cycle, so in the 1 0 4th cycle, you can feed these 2 numbers. So, in between there where they, were latched in these 2 latches and now in the 104 cycle, that S 1 0 2 and S 1 0 4, that means, S 1 0 2 is obtained by adding, these 2 and S 1 0 4 is obtained by adding these 2. So, now, you have got the sum of these 2 and you have to add these 2 numbers and which 1 can be fed in the 104th cycle.

So, as you feed it in the 104th cycle, you will get the result in the 107 cycle, so that means, the total the final output, you will be getting at the end of 107 cycle. So, you can say that total number of cycles required is 4, for the first 4, where 8 numbers were fed to the pipeline, then you will require 92 remaining numbers, were fed, I mean 8 plus 92 remaining 92 numbers, were fed then you will require 2 cycles, to which cycles for adding these 2 head cycles for this.

And then finally, 3 cycles you will require and total is 107 cycles, so this is the last problem that, I wanted to discuss, in this tutorial and I suggest that, you solve problems from the book, try to solve. And that is the best way to learn a subject solving problems and that is the reason why, I have discussed some problems in this particular tutorial and maybe later on, I shall discuss, some more problems in other tutorials.

Thank you.