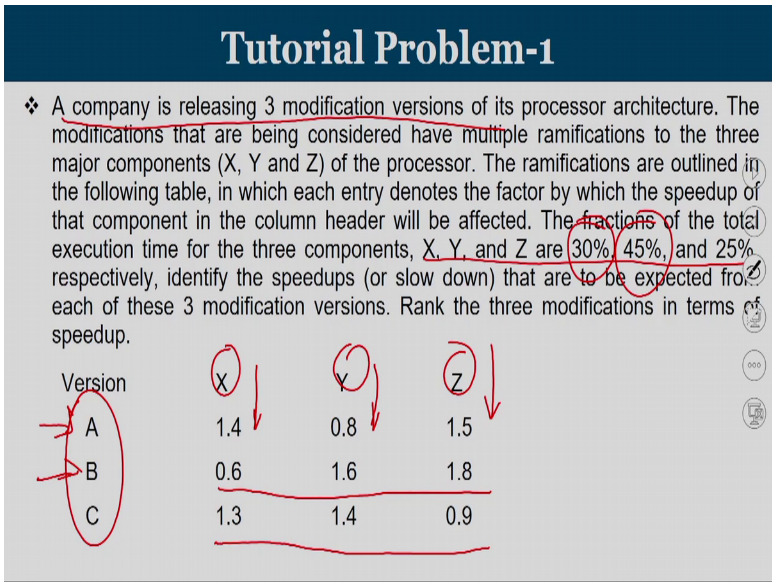
Multi-core Computer Architecture - Storage and Interconnects

Dr. John Jose
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture - 06
Tutorial 1B Instruction Pipeline & Performance

Welcome to the tutorial session of the first week. In the last tutorial, we have worked out few numerical problems, pertaining to performance improvement of processors and pipelining issues. We will continue today's tutorial video also on the same topic. So, we will go in to the first problem. So, here goes the problem.

(Refer Slide Time: 00:57)



Tutorial Problem-1

- A company is releasing 3 modification versions of its processor architecture. The modifications that are being considered have multiple ramifications to the three major components (X, Y and Z) of the processor. The ramifications are outlined in the following table, in which each entry denotes the factor by which the speedup of that component in the column header will be affected. The fractions of the total execution time for the three components, X, Y, and Z are 30%, 45%, and 25% respectively, identify the speedups (or slow down) that are to be expected from each of these 3 modification versions. Rank the three modifications in terms of speedup.

Version	X	Y	Z
A	1.4	0.8	1.5
B	0.6	1.6	1.8
C	1.3	1.4	0.9

A company is releasing 3 modification version of it is processor architecture. The modifications that are being considered have multiple ramifications to the 3 major components X Y and Z of the processor. The ramifications are outlined in the following table, in which each entry denotes the factor by which the speed up of their component in the column header will be affected.

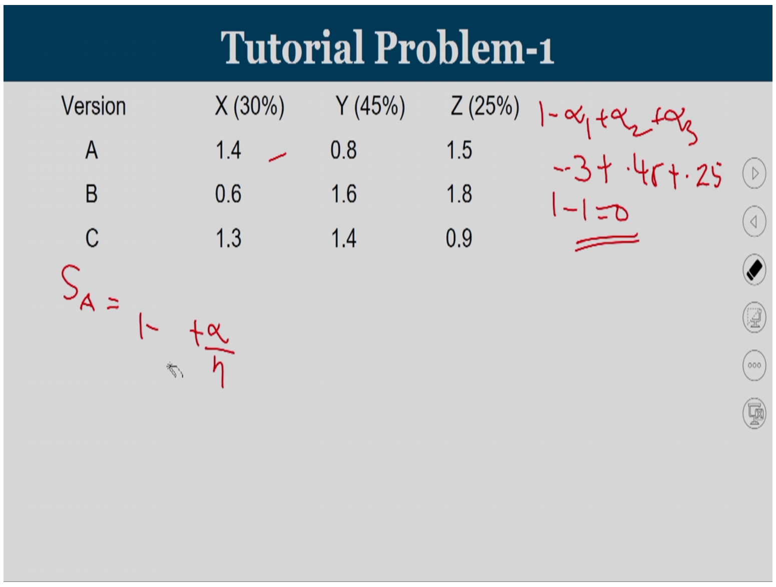
The fractions of total execution time for these 3 components X Y and Z are 30, 45 and 25 percentage respectively. Identify the speed up or slow down that are to be expected from each of these 3 modifications. So, we are supposed to rank these 3 modifications in terms of speed up. So, the question that is given here is; I would write to redraft, a company is releasing 3 modification versions of it processor. So, these are the 3 versions that are mentioned.

Now, version A if you try to implement version A. The component X in the processor there are 3 components A X Y and Z 3 components are there. X will be speeded up by 1.4 times. Y will be speeded up by 0.8 times. And Z will be speeded up by 1.5 times. So, any number less than 1 means it is actually slowing down. If you apply version B, then we can see that these are the speeding a factor correspondingly on X Y Z. And for ,C, these are the speeding a factor.

So, we can see that when you make or when you going to use version A, component X is going to be speeded up by a factor of 1.4 whereas, component Y and component Z are going to have a different kind of a speedup. Similarly, for version B and version C also appropriate changes are there. And we it is already mentioned that the fraction of usage of X Y and Z. There are 30 percent of the code is going to work with X, 45 percent of the code is going to work with component Y. And 25 percent of the code is going to work with component Z.

So, what is the overall speed up that you are going to a chain. So, there are basically 3 components that we are going to divide X Y and Z. All this code will either go to X or go to Y or go to Z. Let us try to solve this problem.

(Refer Slide Time: 03:34)



Tutorial Problem-1

Version	X (30%)	Y (45%)	Z (25%)
A	1.4	0.8	1.5
B	0.6	1.6	1.8
C	1.3	1.4	0.9

$1-\alpha_1+\alpha_2+\alpha_3$

$-.3+.45+.25$

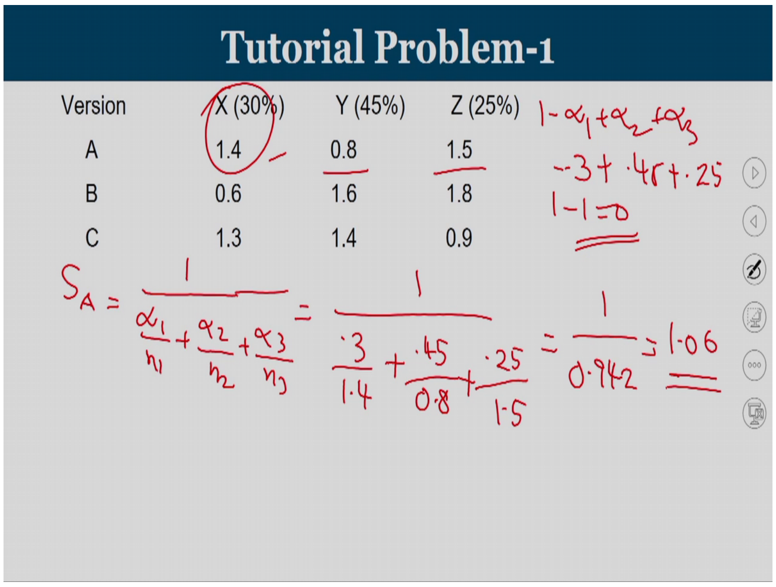
$1-1=0$

$S_A = 1- +\frac{\alpha}{\eta}$

This is the way by which the summary of the task is been given. And we know that the Amdahl's law is going to apply here, because we are going to have some kind of a speed up. So, the speed up for version A can be defined as 1 by since the first version is 1 minus alpha plus alpha by n; where alpha is the component that gets affected. Now in this case if you look into 1 minus alpha 1 plus alpha 2 plus alpha 3. So, alpha 1 we can see that it is 30 percent. So, it is 0.3 and alpha 2 it is 0.45 and alpha 3 it is going to be 0.25.

If you add up all this 3 you will get 1. So, essentially it is 1 minus 1 equal to 0. So, there will not be any component which is not going to be affected by the change. So, we can redraft the Amdahl's law in this way.

(Refer Slide Time: 04:26)



Tutorial Problem-1

Version	X (30%)	Y (45%)	Z (25%)
A	1.4	0.8	1.5
B	0.6	1.6	1.8
C	1.3	1.4	0.9

$1-\alpha_1+\alpha_2+\alpha_3$

$\cdot 3+\cdot 45+\cdot 25$

$1-1=0$

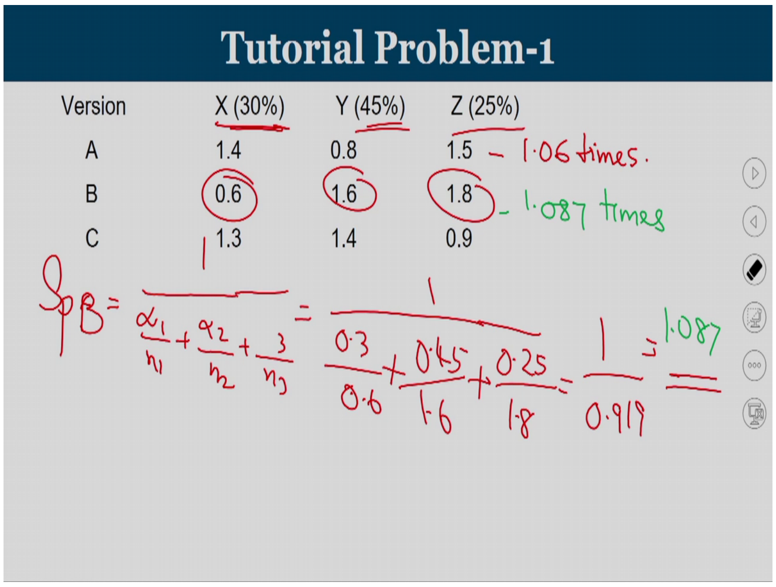
$$S_A = \frac{1}{\frac{\alpha_1}{n_1}+\frac{\alpha_2}{n_2}+\frac{\alpha_3}{n_3}} = \frac{1}{\frac{\cdot 3}{1\cdot 4}+\frac{\cdot 45}{0\cdot 8}+\frac{\cdot 25}{1\cdot 5}} = \frac{1}{0\cdot 942} = 1\cdot 06$$

It is 1 by alpha 1 by n 1, plus alpha 2 by n 2, plus alpha 3 by n 3. There is a way we are going to define this. So, it is 1 divided by, what is first portion? It is 0.3 divided by 1.4. A component X is going to be affected by 1.4 times. And the percentage fraction of the task that is going to be using axis 30 percent plus, that of Y it is 0.45 is diffraction. And the speed up that is going to impact the miss 0.8. And the third one is 0.25 divided by 1.5.

So, if you sort out this you will get a value 1 divided by 0.942 and that is 1.06 times. So, this is the summary of the first performance. The version A, if you are going to use version A, you are going to get a speed up of 1.06 times. Because each component this is the fraction that you are going to get the benefit. So, let us try to summarize the result.

Whatever we are getting in version A so, if you apply version A then the result at that we are going to get is 1.06 times.

(Refer Slide Time: 06:03)



Tutorial Problem-1

Version	X (30%)	Y (45%)	Z (25%)	
A	1.4	0.8	1.5	– 1.06 times.
B	0.6	1.6	1.8	– 1.087 times
C	1.3	1.4	0.9	

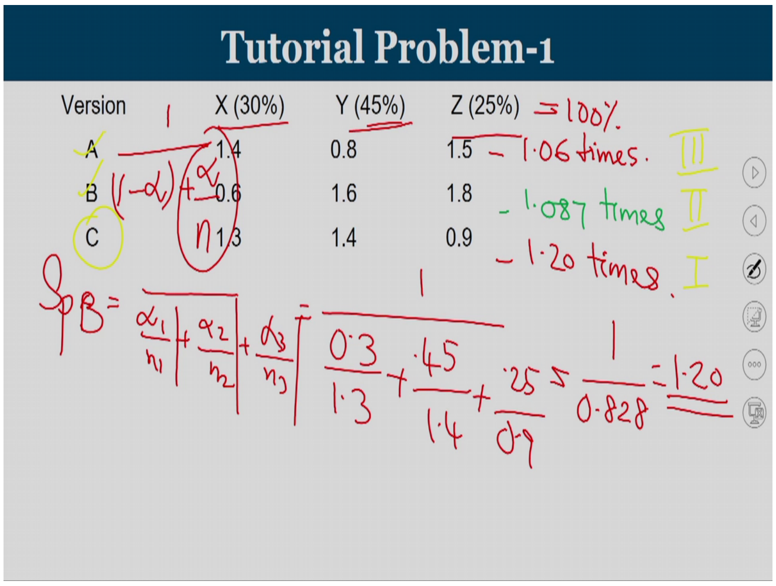
$$Sp_B = \frac{1}{\frac{\alpha_1}{n_1} + \frac{\alpha_2}{n_2} + \frac{3}{n_3}} = \frac{1}{\frac{0.3}{0.6} + \frac{0.45}{1.6} + \frac{0.25}{1.8}} = \frac{1}{0.919} = \underline{\underline{1.087}}$$

So, version A has 1.06 times. Now we will try to work up on version B. The speed up that you are going to get for version B is 1 divided by it is 0.6 that is a alpha 1. So, it is going to be 30 percent. So, 30 percent divided by 0.6. 0.3 divided by 0.6. So, 0.3 comes from the 30 percent factor, divided by n 1 is 0.6. Now the second component is 0.45 that is alpha 2, this is the value of alpha 2, and this is n 2 it is 1.6 times, plus the last component is 0.25, divided by n 3 n 3 is 1.8.

Upon solving we get the value 0.919. And that will be equal to 1.08 times. So, it is 1.087 times is the speed up that we are going to achieve if we operate on version B. So, from this we can see that, first version that is version A is getting a speed up of 1.06 times whereas, version B is going to get if going to give us a speed up of 1.08 7 times. Now we will work into the third version, and see how much it is going to affect the performance.

So, when you substitute the values of the third one. We have 1.3 as the first speed up.

(Refer Slide Time: 07:56)



Tutorial Problem-1

Version	X (30%)	Y (45%)	Z (25%) = 100%		
A	1.4	0.8	1.5	- 1.06 times.	III
B	0.6	1.6	1.8	- 1.087 times	II
C	1.3	1.4	0.9	- 1.20 times.	I

$$\frac{1}{(1-\alpha)+\frac{\alpha}{n}}$$

$$Sp_B = \frac{1}{\frac{\alpha_1}{n_1}+\frac{\alpha_2}{n_2}+\frac{\alpha_3}{n_3}} = \frac{1}{\frac{0.3}{1.3}+\frac{.45}{1.4}+\frac{.25}{0.9}} = \frac{1}{0.828} = 1.20$$

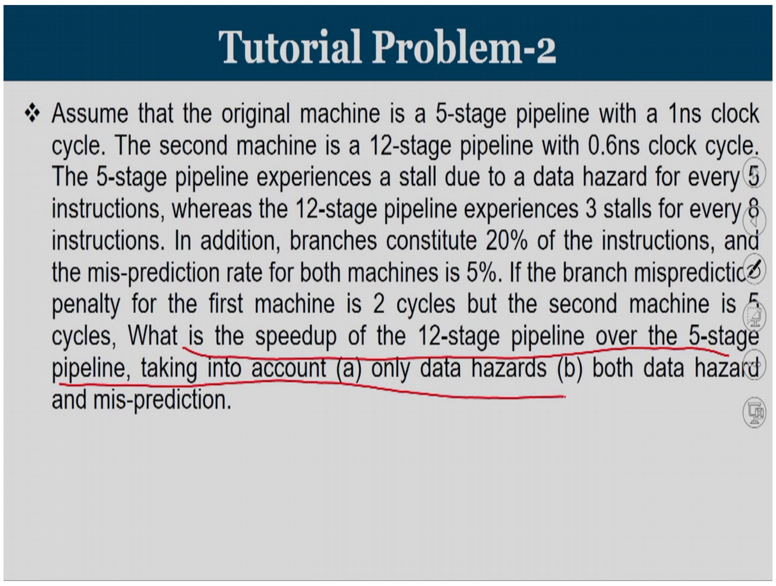
So, 1 divided by 0.3, that value you will get from here, by 1.3 plus 0.45 divided by 1.4 plus 0.25 divided by 0.9. And this will add up to a result of 0.828. And that is equal to 1.20 times. So, the last one is going to give us 1.20 times performance.

So, this is a standard application of Amdahl's law. We have learnt in Amdahl's law that the speed up that, you are going to obtain is given by the equation 1 by 1 minus alpha plus alpha by n. Now if you look at this, in order to get a clearer concept of this let me redraft it once again 1 by 1 minus alpha plus alpha by n. That is been given by Amdahl's law, it states that alpha is the component that is not going to be affected by the enhancement. And n is the number of times the performance is going to improve on the enhancement applied section.

So, if you work on this you can see that in this application; since 30 percent plus 45 percent plus 25 percent added together to 100 percent, then this 1 minus alpha will essentially become 0. That is why we are using only this component alpha 1 by n 1 alpha 2 by n 2 and alpha 3 by n 3. These are the 3 versions that we are trying to use. So, in the question it is asking there we have to put this speed up. So, the best performance we are getting in version C. The second best performance comes in version B, and the third worst performance is going to be there.

So, C is the best, and followed by B and then we are going for A. So, this is how we solve the first question.

(Refer Slide Time: 10:21)



Tutorial Problem-2

- Assume that the original machine is a 5-stage pipeline with a 1ns clock cycle. The second machine is a 12-stage pipeline with 0.6ns clock cycle. The 5-stage pipeline experiences a stall due to a data hazard for every 5 instructions, whereas the 12-stage pipeline experiences 3 stalls for every 8 instructions. In addition, branches constitute 20% of the instructions, and the mis-prediction rate for both machines is 5%. If the branch mispredictio penalty for the first machine is 2 cycles but the second machine is 5 cycles, What is the speedup of the 12-stage pipeline over the 5-stage pipeline, taking into account (a) only data hazards (b) both data hazard and mis-prediction.

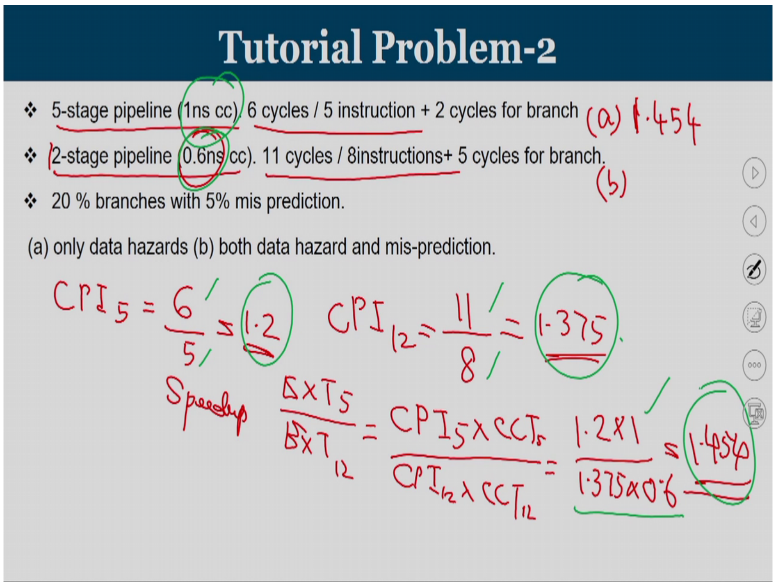
We will move into the second problem. Assume that an original machine is a 5 stage pipeline with a 1 nanosecond clock cycle. There is a second machine which is at 12 stage pipeline with 0.6 nanosecond clock cycle. The 5 stage pipeline experiences a stall due to data hazard for every 5 instruction whereas; the 12 stage pipeline experiences 3 stalls for every 8 instruction. In addition, branches constitute 20 percent of the instruction and misprediction rate for both the machines is 5 percent.

If the branch misprediction penalty for the first machine is 2 cycle. And for the second machine is 5 cycle. What is the speed up of the stalls 12 stage pipeline over the 5 stage pipeline taking into account only data hazards, and both data hazards and misprediction? Let us try to understand what the question is here we are given data of a 5 stage pipeline as well as a 12 stage pipeline. Now the 5 stage pipeline it has a data hazard issue where for every 5 instruction there is s stall. So, the meaning is completion of 5 instruction will take 5 plus 1 that is 6 clock cycles.

Similarly, for the 12 stage pipeline, you have 3 stalls for every 8 instruction. So, when you do 8 instruction there are 3 more stalls, the meaning is 8 instruction will take 11 cycles level means, 8 plus 3 11 cycles to complete. So, that is the aspect of data hazard. So, from this we can get CPI cycles per instruction. Now both the machine if you go for this branch hazards, due to misprediction of branches, they both are going to have some kind of stalls. So, we have 20 percent of the instructions are branches. Now out of the branches only 5 percent of them are mispredicted. These 5 percent of branches are going to take 2 cycle stall in the 5 stage pipeline machine and 5 stalls or 5 cycles stall in the case of a 12 stage pipeline.

So, considering all these factors, the question that is asked is how much speed up the 12 stage pipeline has over the 5 stage pipeline.

(Refer Slide Time: 12:49)



Tutorial Problem-2

- 5-stage pipeline (1ns cc). 6 cycles / 5 instruction + 2 cycles for branch (a) 1.454
- 12-stage pipeline (0.6ns cc). 11 cycles / 8instructions+ 5 cycles for branch. (b)
- 20 % branches with 5% mis prediction.

(a) only data hazards (b) both data hazard and mis-prediction.

$CPI_5 = \frac{6}{5} = 1.2$ $\quad CPI_{12} = \frac{11}{8} = 1.375$

Speedup $\frac{I \times T_5}{I \times T_{12}} = \frac{CPI_5 \times CCT_5}{CPI_{12} \times CCT_{12}} = \frac{1.2 \times 1}{1.375 \times 0.6} = 1.454$

So, I am just trying to summarize what we have discussed so far. The 5 stage pipeline is going to work at 1 nanosecond and the 12 stage. This is not 2 stage it is 12 stage pipeline is going to work on 0.6 nanosecond. So, 5 stage pipeline has a data hazard. It takes one cycle for every 5 instruction; that mean, 6 cycles are required to complete 5 instructions. Whereas, this will take 11 cycles are required to complete 8 instruction.

Let us try to find out what is the stalled at these both these are going to encounter if it is only data hazard. So, the CPI of the 5 stage pipeline is you require 6 cycles to complete 5 instructions. So, it is 6 by 5 is basically the value. Now this is going to be 1.2. Now the CPI of the 12 stage pipeline is it requires 11 cycles to complete 8 instruction. So, it is going to be 1.375. These are the CPI values. Now the question is how much speed up you are going to get if you revisit the question, what is the speed up of the 12 stage pipeline over the 5 stage pipeline by taking into the data hazard.

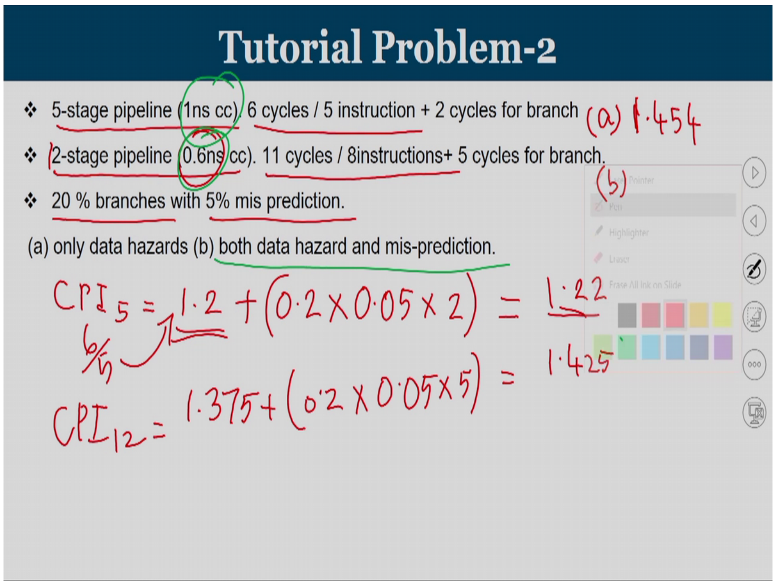
So, if you look at the speed up that you are going to get in the 5 stage pipeline, speedup is defined as execution time of the 5 stage pipeline divided by execution time of the 12 stage pipeline; that is, CPI of the 5 stage pipeline in to clock cycle time divided by CPI of the 12 stage pipeline into it is clock cycle time, clock cycle time of 5 this is clock cycle time of 12. This will take us to 1.2 nanosecond 1.2 is a CPI into 1 divided by 1.375 into 0.6. This is going to give us a speed up of 1.454 that is a speed up we are going to a chain.

So, the answer for the first component is the 12 stage pipeline has a speed up of 1.454 times over the 5 stage pipeline. Let us try to revisit what we have done. From the given question, considering only the data hazards the 5 stage pipeline is going to have. It is take 6 clock cycles to complete 5 instructions. So, 6 by 5 1.2 is a CPI. And the CPI for the 12 one it is going to take 11 clock cycles to complete 8 instruction. So, it is 1.375 they both are operating at a different clock the 5 stage pipeline is having a 1 nanosecond clock whereas, a 12 stage pipeline is going to have a 0.6 nanosecond clock.

So, the CPI into clock cycle time is the execution time per instruction. So, each instruction will take 1.2 nanosecond to complete in the case of a 5 stage pipeline whereas, in the case of a 12 stage pipeline we are going to take 1.375 into 0.6. Let us say clock cycle time getting an average speed up of 1.454. Now let us try to visit the second portion both data hazard and misprediction are taken together, how much is the performance that you are going to get.

So, in the second case along with the data hazard, we have to add the stalls that are incurred by the pipeline when there is a branch misprediction and we how to find that the branch misprediction happens how frequently. The branch misprediction happens in 20 percent of the branches.

(Refer Slide Time: 17:15)



Tutorial Problem-2

- 5-stage pipeline (1ns cc). 6 cycles / 5 instruction + 2 cycles for branch (a) 1.454
- 12-stage pipeline (0.6ns cc). 11 cycles / 8instructions+ 5 cycles for branch. (b)
- 20 % branches with 5% mis prediction.

(a) only data hazards (b) both data hazard and mis-prediction.

$CPI_5 = 1.2 + (0.2 \times 0.05 \times 2) = 1.22$

$6/5$

$CPI_{12} = 1.375 + (0.2 \times 0.05 \times 5) = 1.425$

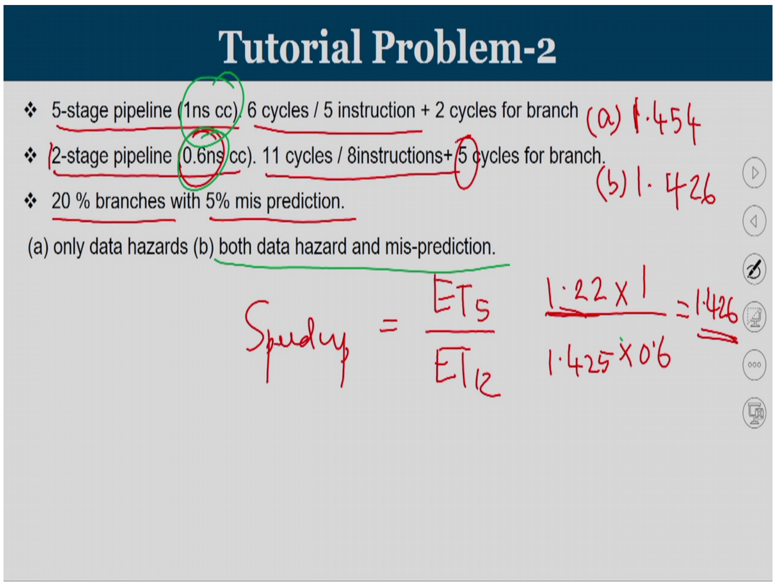
And 5 times it will going to mispredict. So, there are total of 20 percent is branches, and out of the branches 5 percent of the branches as misprediction.

So, the CPI of the first machine we have seen that with the data hazards it is 1.2, plus there are 20 percent of branches. Now each branch out of the 20 percent branches [noise, only 5 percent of them are going to be mispredicted. And so, this much is the stall that is called by the branch. And the value is so, this value 1.2 is the stalls that are incurred by the data hazard that is basically a 6 5 by this 6 by 5 is 1.2. So, this will give you a value 1.22.

Now, when it comes to the 12 stage pipeline, the value that they were having is the initial CPI was 1.375, plus 0.2 that is a total percentage of branch instruction. And out of the branch 0.05 is misprediction, but 12 stage pipeline for every misprediction it is going to take 5 cycles. So, this will give us a value 1.425. Now when you are going to find out the speed up, then the speed up can be mentioned as the ratio of the execution time.

So, when you consider the execution time. The execution time is defined as CPI into the corresponding clock cycle of one machine divided by other.

(Refer Slide Time: 19:02)



Tutorial Problem-2

- 5-stage pipeline (1ns cc) 6 cycles / 5 instruction + 2 cycles for branch
- 2-stage pipeline (0.6ns cc). 11 cycles / 8instructions+5 cycles for branch.
- 20 % branches with 5% mis prediction.

(a) only data hazards (b) both data hazard and mis-prediction.

(a) 1.454

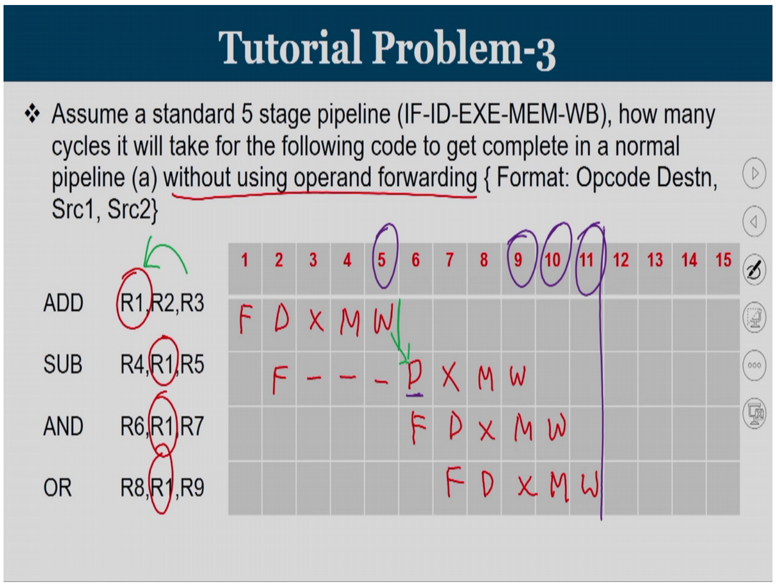
(b) 1.426

$$\text{Speedy} = \frac{ET_5}{ET_{12}} \quad \frac{1.22 \times 1}{1.425 \times 0.6} = 1.426$$

So, it is execution time of the 5 stage pipeline divided by the execution time of the 12 stage pipeline. So, it is 1.22 into 1 divided by 1.425 into 0.6. And the value is 1.426. So, when you consider the misprediction, then the speed up obtained is slightly less, because it is going to have 5 cycles for the branch misprediction.

So, the answer for this one is, when you consider only data hazard, then the speed up that the 12 stage pipeline is having over the 5 stage pipeline is 1.454. And when you consider both data hazard and the branch hazard then it is 1.426.

(Refer Slide Time: 19:57)



Tutorial Problem-3

- Assume a standard 5 stage pipeline (IF-ID-EXE-MEM-WB), how many cycles it will take for the following code to get complete in a normal pipeline (a) without using operand forwarding { Format: Opcode Destn, Src1, Src2}

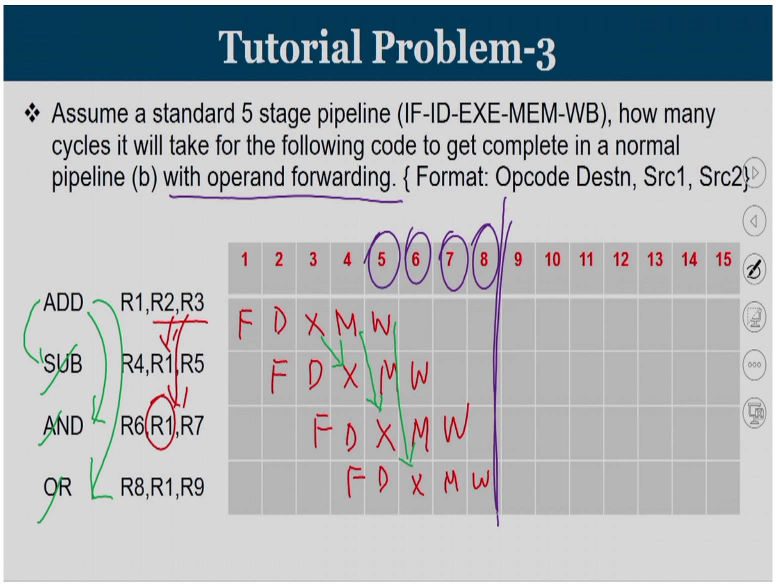
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ADD	R1,R2,R3	F	D	X	M	W										
SUB	R4,R1,R5		F	-	-	-	D	X	M	W						
AND	R6,R1,R7						F	D	X	M	W					
OR	R8,R1,R9							F	D	X	M	W				

Moving into the third problem assume a standard at 5 stage pipeline IF ID EX MEM and write back, how many cycles it will take for the following code to get complete? So, by what end execution will get over, if you are using a normal pipeline without using an operand forwarding. So, let us considered this one, the first add instruction. So, it is going to perform fetching on clock cycle one fetch, decode execute MEM and write back.

So, there is no operand forwarding. So, you have to wait for the result. Second one there is a raw dependency. You can see that this is R 1, this is also R 1. So, you are not going to perform the decoding because there is no operand forwarding. So, decoding can be done only here. So, these are stalls, why decoding is done only there? Meaning is, the first instruction is going to write into R 1 only at clock cycle 5. So, then only I will get the value. So, here only I can perform decoding. So, that lead to is case that X, this is MEM and this is write back.

Now, what we have to do is, in this case now fetching of the third one that is going to happen only here, fetch. The third one is also having a dependency on R 1, but now the R 1 value is ready. So, fetch decode execute MEM and write back. And the last one is also having a dependency, but since we are using a normal simple pipeline, this is the order in which the instructions are getting over. So, if you revisit what happened? The first instruction get over it 5. Second instruction can perform decode only at clock cycle 6, because second instruction has to read from R 1. And R 1's value is available only a when the first instruction is complete. So, second instruction can perform decoding at clock cycle 6, and that gets over at 9 and thereafter your 10 11 and so, you are going to complete by clock cycle number 11.

(Refer Slide Time: 22:12)



Tutorial Problem-3

❖ Assume a standard 5 stage pipeline (IF-ID-EXE-MEM-WB), how many cycles it will take for the following code to get complete in a normal pipeline (b) with operand forwarding. { Format: Opcode Destn, Src1, Src2}

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ADD	R1,R2,R3	F	D	X	M	W										
SUB	R4,R1,R5		F	D	X	M	W									
AND	R6,R1,R7			F	D	X	M	W								
OR	R8,R1,R9				F	D	X	M	W							

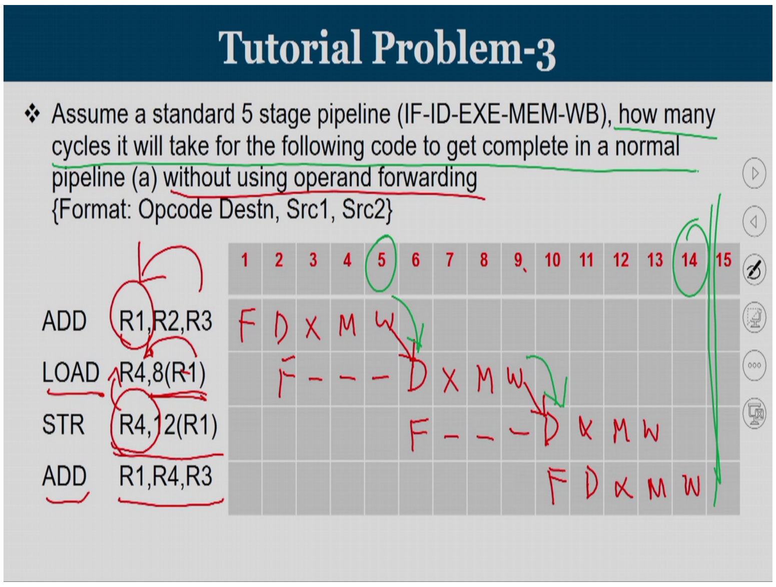
Now, the same question is being redrafted what happens if it is with operand forwarding. So, the same question is there we are trying to understand what will happen if it is with operand forwarding. So, we will work out the same problem. So, fetch decode execute MEM and write back. These are the 5 stages now operand forwarding is permitted so, I have fetch here, I will perform decode. From the output of ALU that is a value of R 2 plus R 3 is forwarded to the input of ALU.

So, in this case forwarding happens. So, what is forwarding here? The value from the output of ALU is forwarded to the input of ALU. So, it is an ALU to ALU operand forwarding. So, there is no stall that happens, data will reach at the correct time now. When you go to the next instruction, it is fetch decode and execute. So, this wants the same R 1 data, and that R 1 data is to be forwarded, but this time the forwarding happens from the MEM stage. So, this is how the data will reach the execution stage and if you go to the last instruction. So, it is fetch decode execute MEM and write back. Here also the value is copied into the ALU.

So, we have 3 operand forwarding, the value from output of ALU is given to the input of ALU. Here from output of MEM stage it is given to the input of ALU. And here from the output of write back stage it is given to the input of ALU, such that the second third and the 4th instruction can work in its assigned time slot. Because ALU will get the data by appropriate operand forwarding.

So, here we can see that first instruction complete by 6. And there after every instruction is going to complete at the same time. So, the whole thing will get over at 8 clock cycles.

(Refer Slide Time: 24:21)



Tutorial Problem-3

- Assume a standard 5 stage pipeline (IF-ID-EXE-MEM-WB), how many cycles it will take for the following code to get complete in a normal pipeline (a) without using operand forwarding
{Format: Opcode Destn, Src1, Src2}

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ADD	R1,R2,R3	F	D	X	M	W										
LOAD	R4,8(R1)		F	-	-	-	D	X	M	W						
STR	R4,12(R1)						F	-	-	-	D	X	M	W		
ADD	R1,R4,R3										F	D	X	M	W	

Now, let us go for another pair of instruction this is also, but the code is changed. And we have an add instruction followed by a load followed by a store, and then you have another add this is also without operand forwarding. So, if it is without operand forwarding here you have fetch, decode, execute, MEM and write back. Let us see whether there is any dependency. Here there is a value R 1 and that R 1 is used in the address computation for load.

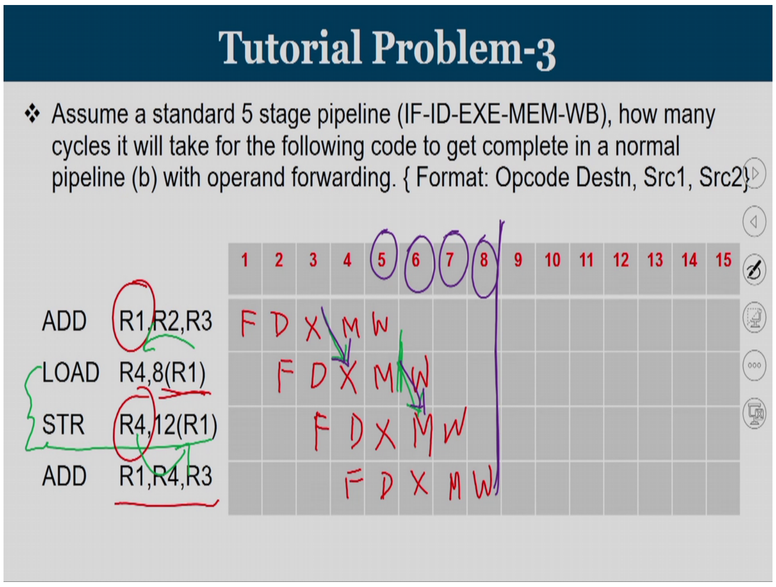
So, R 1 will be available here. So, it is fetch decode R 1 plus 8 has to be used. So, it is if it is without operand forwarding, I can perform. So, if it is without operand forwarding, then the decode stage cannot be complete. Because my id state is possible only after value is written to R 1 because a load instruction has to read the value of R 1. So, reading of R 1 is possible only the writing of R 1 by the previous instruction is over. So, at this point only we can perform the second instruction.

Now, whatever you are going to load the value that loaded value has to be written by this. So, the next instruction store, value of will be written to R 4, only at this point only at clock cycle number 9. So, store can read the value only at this. So, fetch, decode, execute, MEM and write back. So, these are the stalls, I will repeat what happens for the store instruction. The store instruction has to read the data that is written by the load instruction previously. The load instruction will complete it is operation. That is value will be written to R 4 only at clock cycle number 9. So, store can perform the register read only at clock cycle 10 these both are dependent, that is a dependency here then it completes.

Now, you have to perform add of R 1, R 4 and R 5. So, if you look at that it is dependent on R 4, but R 4 value is already available. So, it can run it is assigned time slot. So, if it is without operand forwarding, what are we trying to see, the first instruction get complete by 5. Second instruction decode stage cannot happened before the write backstage of the first instructions. So, there exist a dependency like this. Similarly, the second instruction after it writes then only the third instruction can read. So, it happens like this and the last instruction happens.

So, in this case, the instructions are getting over at the 14th clock cycle. So, the question is, how many cycles it will take for the following code to get complete. So, this code will get over only in 14 clock cycles. Let us try to do this problem with operand forwarding. The same question is with operand forwarding. So, if it is with operand forwarding.

(Refer Slide Time: 27:36)



Tutorial Problem-3

- Assume a standard 5 stage pipeline (IF-ID-EXE-MEM-WB), how many cycles it will take for the following code to get complete in a normal pipeline (b) with operand forwarding. { Format: Opcode Destn, Src1, Src2}

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ADD	R1,R2,R3	F	D	X	M	W										
LOAD	R4,8(R1)		F	D	X	M	W									
STR	R4,12(R1)			F	D	X	M	W								
ADD	R1,R4,R3				F	D	X	M	W							

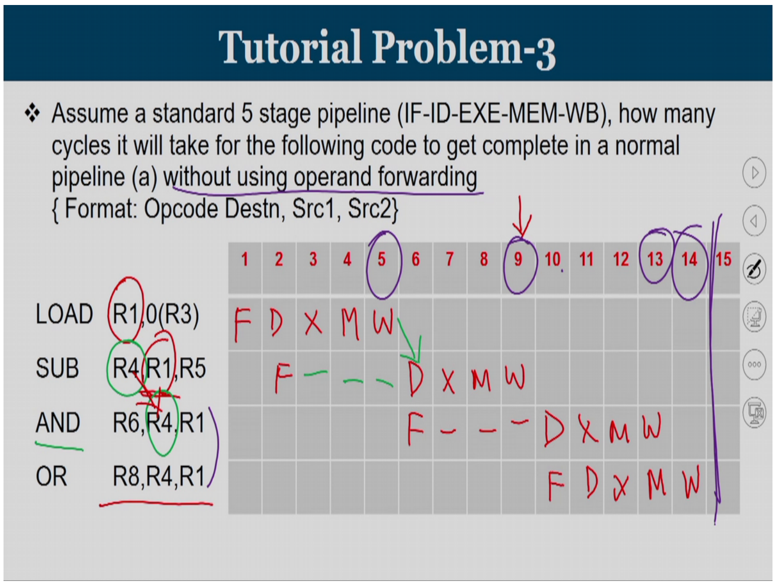
The first instruction will run in it is assigned time slot. Now if you look at the second instruction there is operand forwarding there. And the value to be written to R 1 can be forwarded to the input of ALU for address computation.

So, in this case it will run in it is assigned time slot. Because I could forward the value of R 2 plus R 3 from the ALU to the input of ALU such that R 1 can be used for effective address computation. And now we are going to store the value so, by what time the value will be available? The value is available at this point. Now if you have a store operation, the store operation will decode, but 12 of R 1 is the address computation the address computation happens in it is normal slot because we want the value of R 4 only before storing. So, it is actually a value from the input of MEM stage.

So, at this point the data is ready the loaded. Value is ready and that loaded value is to be copy to other location. So, if you look at these 2 addresses, I load a value into R 4 and whatever is there in R 4, I am storing into a different address. So, the value that is read from the memory is forwarded to the input of memory units such that you can complete the operation. So, this will be your write back stage. And then we have an R 1, R 4 and R 3. There is now dependency issue there. So, fetch, decode execute, MEM and write back.

So, this shows that in this case with 8 clock cycle, we are able to complete up the operation. So, the first instruction get over at 5. Because of effective operand forwarding, second instruction will complete at 6 because from the output of ALU the data is copied to the input of ALU. And from the output of MEM stage data is copied to the input of MEM stage. So, then the third instruction also will get over in it is a same time slot. So, overall this instruction with operand forwarding it is going to complete in 8 clock cycles. Whereas, you can see that without operand forwarding in the previous example it was taking 14 clock cycles.

(Refer Slide Time: 30:02)



Tutorial Problem-3

- Assume a standard 5 stage pipeline (IF-ID-EXE-MEM-WB), how many cycles it will take for the following code to get complete in a normal pipeline (a) without using operand forwarding
  { Format: Opcode Destn, Src1, Src2}

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LOAD	R1,0(R3)	F	D	X	M	W										
SUB	R4,R1,R5		F	-	-	-	D	X	M	W						
AND	R6,R4,R1						F	-	-	-	D	X	M	W		
OR	R8,R4,R1										F	D	X	M	W	

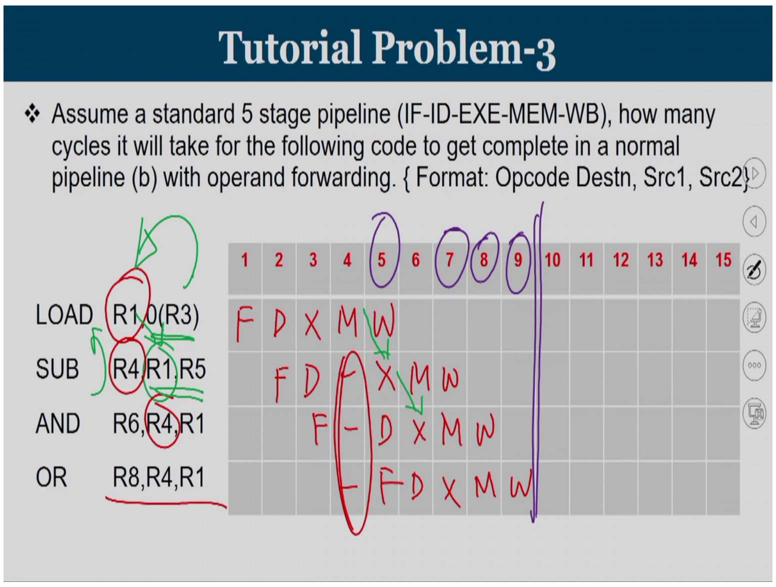
Now, let us goes to another set of code; where we have a load instruction followed by a couple of ALU instructions. This is also without using operand forwarding. We will try to see how we can do that. So, fetch, decode execute, MEM and write back. These are the 5 stages for the first instruction. Since there is no operand forwarding, and there exist a data dependency between the first and second instruction. Second instruction can perform register read only if the first instruction is going to complete. So, this is; second instruction has to read from R 1 value of R 1 will be available only at the end of the fifth clock cycle. So, we are going to have stall like this.

Now, we have the next instruction is and is dependent on the output of subtraction. So, if you look at that, we will fetch will happen here, decode can happen only this point because R 4 is needed. Value will be written to R 4 only at this point; that is, at clock cycle number 9 only we are going to write the value into R 4 and that R 4 is needed. So, R 4 can be read only after the ninth clock cycle, these are the stalls. Now if you look at the last one. There is no dependency issues there. So, they can be assigned in it is normal time slot.

So, the whole instructions will get over at clock cycle 14. So, if you try to conclude first instruction will over at 5. Second instruction decode is possible only at 6 because it should happen after 5. So, thereby second instruction getting over at 9. There exist a row dependency between the second and third instruction for the value R 4. So, second instruction completes writing into R 4 at clock cycle 9. So, third instruction can do it is register reading or decode operation only at 10. Third instruction will complete at 13. There is no issues between the third and 4th. So, the last instruction is going to complete at 14.

So, this code will complete at clock cycle number 14.

(Refer Slide Time: 32:27)



Tutorial Problem-3

❖ Assume a standard 5 stage pipeline (IF-ID-EXE-MEM-WB), how many cycles it will take for the following code to get complete in a normal pipeline (b) with operand forwarding. { Format: Opcode Destn, Src1, Src2}

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LOAD	R1,0(R3)	F	D	X	M	W										
SUB	R4,R1,R5		F	D	-	X	M	W								
AND	R6,R4,R1			F	-	D	X	M	W							
OR	R8,R4,R1				-	F	D	X	M	W						

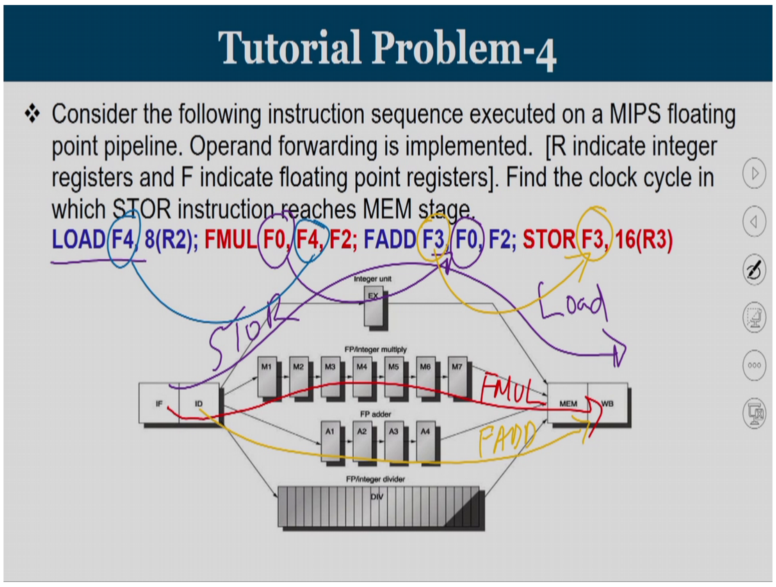
Let us see what will happen if we are going to enable operand forwarding. So, if you enable operand forwarding so, fetch decode execute MEM and write back. Now the first instruction will get it is data to R 1 only at clock cycle number 4. So, execution is possible only at this stage. So, we have A 1 cycle stall. The meaning is, whenever you have an ALU instruction immediately, after a load instruction then there exist A 1 cycle stall.

So, from this memory stage only we can copy, because the value that is taken from 0 of R 3 is accessed from memory, only at the end of the MEM stage. From the MEM stage you are forwarding it to the input of ALU because there exist a data dependency. So, this is the way how you compute this subtraction instruction. And then when you go to the and there is a data dependency between them, but if you use an effective operand forwarding, then without a stall we could manage. From the output of ALU so, at this point the value of R 1 plus R 5 is computed, and that is to be written to R 4 that can be forwarded to the input of ALU such that the next instruction can work. And then go for the last one. Decode, execute, MEM and write back. Here we can see that there exist no data dependency between them. So, it will complete in the assigned slot.

So, these are the places where you get the stalls. So, one stall is propagating, and if you look at the completion first instruction completed 5. Second instruction it will have a stalled so, it get complete only at 7. So, whenever you have an ALU instruction immediately after a load instruction. Then there is a stall, even with operand forwarding, and then all the instructions will complete after one cycle each. So, at the end of the ninth clock cycle we are able to complete all the instructions.

So, this gives you a fair id about how things work when operand forwarding is enabled or when operand forwarding is not enabled. Now let us try to understand what happens when we go for a floating point pipeline, consider the following instruction sequence executed on a MIPS.

(Refer Slide Time: 34:59)



Tutorial Problem-4

❖ Consider the following instruction sequence executed on a MIPS floating point pipeline. Operand forwarding is implemented. [R indicate integer registers and F indicate floating point registers]. Find the clock cycle in which STOR instruction reaches MEM stage.

LOAD F4, 8(R2); FMUL F0, F4, F2; FADD F3, F0, F2; STOR F3, 16(R3)

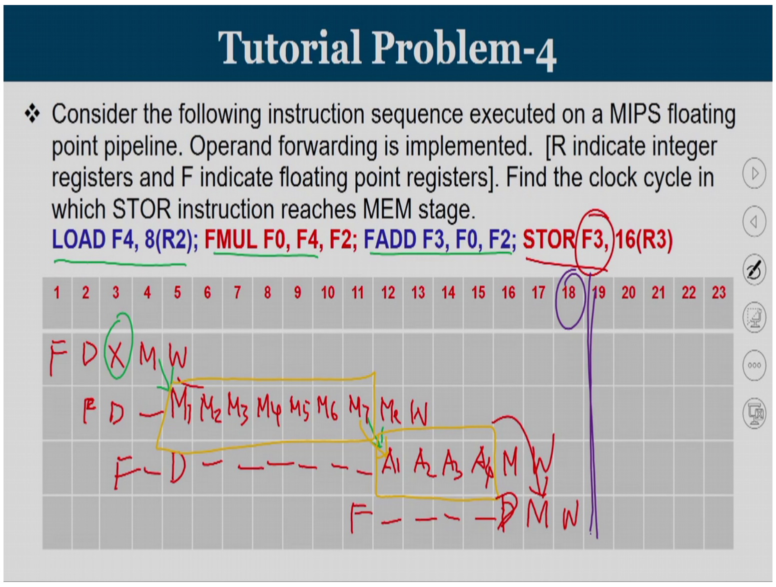
Floating point pipeline operand forwarding is implemented. R indicate integer register and F indicate floating point registers. Find the clock cycle in which store instruction reaches the MEM stage. So, we have a load instruction. So, the meaning of this load instruction is from the address you are copying the value into F 4.

Now, that F 4 is used to multiply with F 2. So, that you store the result in F 0. Now that F 0 is used to get the result in F 2. So, you get this value F 0 and F 2 are added to do store the result in F 3. And F 3 value is stored back into the memory location. Now this is how your floating point unit looks like. So, the load operation is carried out by the integer unit. That is by the load. Now the second instruction gets the multiplication instruction is carried out by the multiplication unit that is called FMUL floating point multiplication.

Now, the third instruction is carried out by the floating point add unit. So, it is FADD and the last instruction is again carried out by this unit and that is called is store. And we have to understand that the floating point multiplication floating point adders are pipelined, but there exist a dependency between them we are going to write the result into F 0. And that is to be used by floating point add. So, there is a dependency this is going to write a value into F 3 that is going to be used to by the store. So, there is one more dependency. And the first dependency exist we are going to write a value into F 4 that is to be used by this multiplication.

So, there are 3 dependency at each stages.

(Refer Slide Time: 36:59)



Tutorial Problem-4

❖ Consider the following instruction sequence executed on a MIPS floating point pipeline. Operand forwarding is implemented. [R indicate integer registers and F indicate floating point registers]. Find the clock cycle in which STOR instruction reaches MEM stage.

LOAD F4, 8(R2); FMUL F0, F4, F2; FADD F3, F0, F2; STOR F3, 16(R3)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
F	D	X	M	W																		
	F	D	–	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M	W										
		F	–	D	–	–	–	–	–	–	A_1	A_2	A_3	A_4	M	W						
										F	–	–	–	–	D	M	W					

Let us try to fill it. For the first instruction we are going to fill it up it is a normal integer operation. So, it take only 5 cycles fetch, decode, execute, MEM and write back. Since operand forwarding is permitted, if you look at what stage M 1 can start? You are going to load a value and your M 1, this amount stands for multiplication has 7 stages so, this is your M 1. M 2, this is M 3, now M 4, M 4, M 6 and M 7. And here comes the MEM stage and this is the write backstage.

Now, let us try to understand why we are starting M 1 are clock cycle number 5. The first instruction is a load instruction. So, 8 and R 2 are added at the execution unit. And the value will be available in the memory unit at the end of MEM operation. So, by operand forwarding, multiplication can start as early as the fifth clock cycle. Now there exist a dependency between the second instruction and the third instruction.

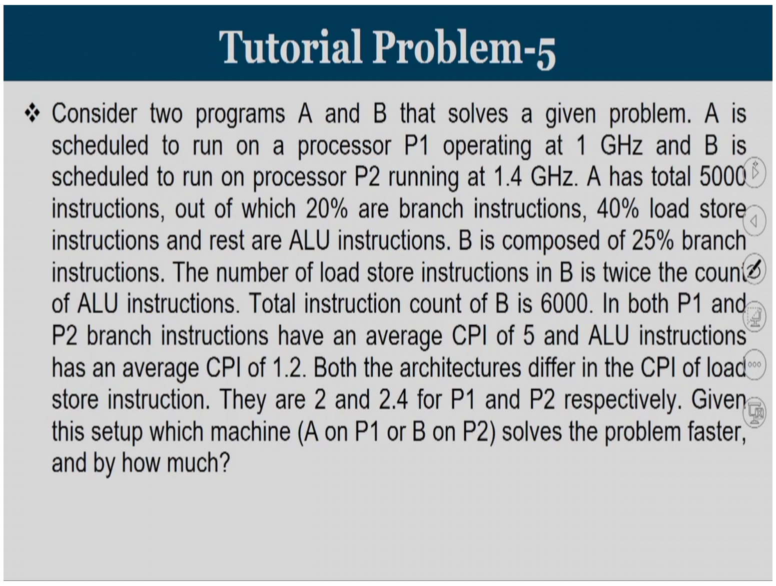
So, your value is available only at this point. So, even if you apply operand forwarding. The adding unit can start it is up adding only here. So, it is A 1, A 2, A 3 and A 4 can happen only at this point. So, when are you going to fetch. You could fetch here, you could decode here. So, this is a stall now then you have to wait. Because your A 1 can start only at this point. This is due to data dependency.

So, these are the 7 stages of multiplication, the 7 stages of multiplication. And these are the 4 stages of the addition. So, we are trying to forward from the output of multiplication is forwarded to the input of adding and if you continue the operations it is MEM and write back. So, the add operation get over at the 17th clock cycle.

Now, you have to perform a store operation on F 3. So, if operand forwarding is permitted store can perform it is memory operation, but 2 memory operation cannot happen together. So, this is MEM and write back. So, this is the process where fetching can do. So, you fetch and then you have a decode because the store operation is cannot proceed, because it is dependent on F 3. So, until the add operation is over, store cannot proceed with the operation and with operand forwarding you can forward the value.

So, essentially at the end of the 18th clock cycle, this floating point operation is getting over this is the dependency between them.

(Refer Slide Time: 40:06)



Tutorial Problem-5

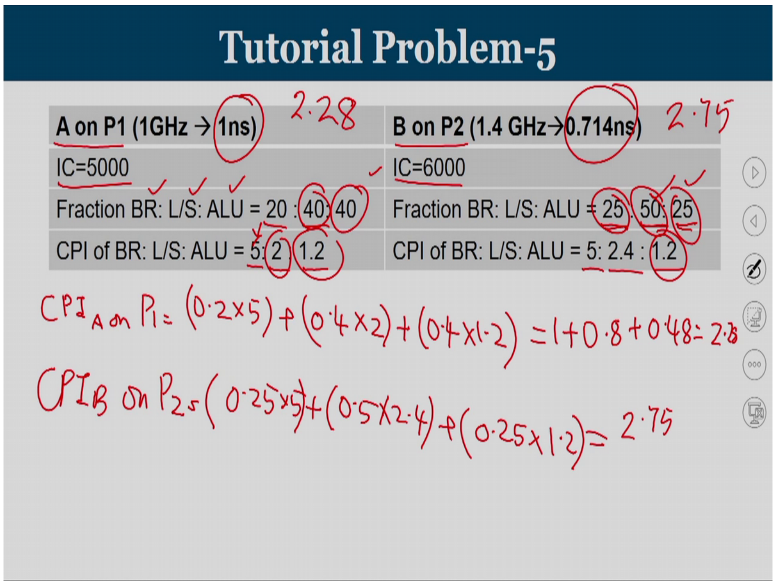
❖ Consider two programs A and B that solves a given problem. A is scheduled to run on a processor P1 operating at 1 GHz and B is scheduled to run on processor P2 running at 1.4 GHz. A has total 5000 instructions, out of which 20% are branch instructions, 40% load store instructions and rest are ALU instructions. B is composed of 25% branch instructions. The number of load store instructions in B is twice the count of ALU instructions. Total instruction count of B is 6000. In both P1 and P2 branch instructions have an average CPI of 5 and ALU instructions has an average CPI of 1.2. Both the architectures differ in the CPI of load store instruction. They are 2 and 2.4 for P1 and P2 respectively. Given this setup which machine (A on P1 or B on P2) solves the problem faster, and by how much?

Now, let us move into the final problem for today. The fifth problem; the fifth problem reads like this consider 2 programs A and B that solves a given problem. A is scheduled to run on a processor P1 operating at 1 gigahertz and B is scheduled to run on processor P2 running at 1.4 gigahertz. A has total 5000 instructions out of which 20 percent are branch instructions, 40 percent are load store instructions and the rest are ALU instruction. B is composed of 25 percent branch instructions. The number of load store instruction in B is twice the count of ALU instruction.

Total instruction count of B is 6000. In both P1 and P2 branch instructions have an average CPI of 5 cycles per instruction of 5. And ALU instruction has an average CPI of 1.2. Both the architectures differ in the CPI of load store instruction. So, CPI of load store instruction for P1 is 2 and that of P2 is 2.4. Given the set up the program A on P1 or the program B on P2 will solve the problem faster. So, this is a question of whether there are 2 contacts. 2 programs A and B are there A is going to run on P1 and B is going to run on P2.

Now, the question is whether A 1 P1 is more faster in solving or B 1 P2 is more faster on solving. A and B are 2 different programs so, the instruction count of A and B is different. Since they are running on different architectures the cycles per instruction for each of these architecture is also different that is being given and the clock cycle of P1 and P2 is also different. So, you are trying to compare with 2 different design scenarios.

(Refer Slide Time: 41:58)



Tutorial Problem-5

A on P1 (1GHz → 1ns)	2.28	B on P2 (1.4 GHz→0.714ns)	2.75
IC=5000		IC=6000	
Fraction BR: L/S: ALU = 20 : 40 : 40		Fraction BR: L/S: ALU = 25 : 50: 25	
CPI of BR: L/S: ALU = 5: 2 : 1.2		CPI of BR: L/S: ALU = 5: 2.4 : 1.2	

$$CPI_{A \text{ on } P_1} = (0.2\times5)+(0.4\times2)+(0.4\times1.2) = 1+0.8+0.48 = 2.28$$

$$CPI_{B \text{ on } P_2} = (0.25\times5)+(0.5\times2.4)+(0.25\times1.2) = 2.75$$

Let us try to summarize the data that is given A is running on P1. So, and P1 is A 1 gigahertz processor. So, the clock cycle speed is 1 nanosecond. B is running on P2, it is a 1.4 gigahertz. So, 1.4 is 1 divided by 1.4 it is 0.714 nanosecond.

Program A has 5000 instructions whereas, program B has 6000 instruction. The instruction count is different. Now there are 3 category of instructions, they are branch instruction, load store instruction and ALU instruction. It is mentioned in the question that there are 20 percent of branch instruction in program A and 40 percent of them are load store instruction the balance is ALU instruction. So, out of the 100 percent instruction if 20 percent is branch and 40 percent is load store then the balance 40 percent is your ALU.

When it comes to the fraction of instructions in the case of B they are mentioning that 25 percent of them are branch instruction, the remaining are load store and ALU. The peculiarities the number of load store instruction is doubled that of ALU and that is satisfied only if this is the load store instruction that is doubled. So, load store instruction has to be 50 percent and ALU has to be 25 percent. The CPI is given the CPI of branch instruction is 5 in both the case. The CPI of load store instruction is 2 in the case of P1, and 2.4 in the case of P2 and CPI of ALU instruction is same.

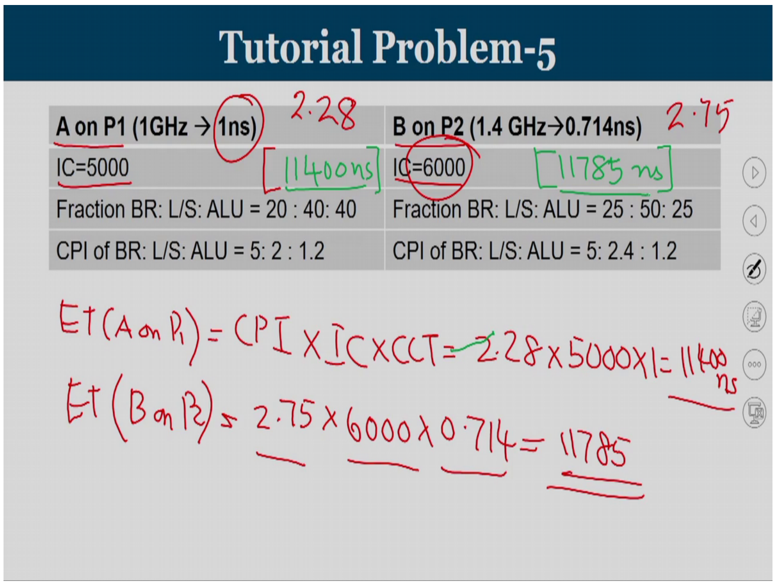
Now, let us try to understand what is the average CPI of program A. So, CPI of program A on P1 so, we have 0.2 that is 20 percent of the instructions are branch. So, 0.2 into 5, plus 40 percent of them are branch instruction. So, 0.4 into 2 plus remaining 40 percent are ALU instruction. So, they are having in CPI of 1.2.

So, what are we trying to do here is, different instructions as different CPI, let us try to find out what is the CPI of the program A. A has different mix of instruction; it has branch instruction it is load store instruction and it has ALU instruction. So, with the percentage fraction we are trying to combine all these. So, the value that you get here is 1 plus 0.8 that is the second one and the third one is 0.48.

So, ultimately we get the value 2.28 so, the CPI is 2.28. So, the CPI for A is 2.28. Now let us try to work on the CPI of program B on P2. So, there the fraction is 25 percent of them are branch instructions, plus 50 percent of them are load store instruction; which will take a CPI of 2.4 plus remaining 25 percent of them are ALU instruction which will take a CPI of 1.2.

So, this upon solving you will get the number 2.75. So, the CPI of the second problem is 2.75. Now with CPI; how will you find out the execution time? So, program A there are 5000 instruction, each instructions average CPI value is 2.28. And for program B there are 6000 instruction, each instruction CPI value is 2.75. The instruction count is varying; the CPI is varying as well as the clock cycle time is varying.

(Refer Slide Time: 46:40)



Tutorial Problem-5

A on P1 (1GHz → 1ns) 2.28	B on P2 (1.4 GHz→0.714ns) 2.75
IC=5000 [11400ns]	IC=6000 [11785 ns]
Fraction BR: L/S: ALU = 20 : 40: 40	Fraction BR: L/S: ALU = 25 : 50: 25
CPI of BR: L/S: ALU = 5: 2 : 1.2	CPI of BR: L/S: ALU = 5: 2.4 : 1.2

$ET(A \text{ on } P_1) = CPI \times IC \times CCT = \cancel{2}\, 2.28 \times 5000 \times 1 = 11400\ ns$

$ET(B \text{ on } P_2) = 2.75 \times 6000 \times 0.714 = 11785$

So, execution time of A on P1. Then is CPI into instruction count into clock cycle time. So, that will come to 2.28 into instruction count is 5000 into clock cycle is 1 nanosecond.

So, the final value that you get is 11400 nanosecond. So, the first 2 program is going to solve our problem in 11400 nanosecond. Now let us try to see what is the execution time of B on P2. So, same equation apply CPI. So, CPI here the found out CPI is 2.75 into instruction count is 6000, but the clock cycle time is little lower, it is 0.714 only. And this will give us a value 11785.

So, 2.75 is the CPI, 6000 is the IC value, and clock cycle is 0.714 because P2 is operating on a different clock. So now, we will write what is the execution time for program B on P2. It is 11785 nanosecond. Now you compare your 2 designs. Your first design is going to solve the task that is program A 1 P1 will take 11400 nanoseconds, and program B on P2 will take 11785.

So, then smaller the execution time the better. So, A 1 P1 solves the problem faster than B 1 P2. So, in this way given 2 designs, 2 processors of different frequency, 2 different programs running on these 2 processors and what is approximate time that you are going to take effectively it is going to be managed by this.

So, with this we come to the end of today's tutorial session. I hope this session was useful to you, because you got familiar with some numerical exercise pertaining to pipeline. We have taken few examples and 5 to find out what is the role of operand forwarding how much is a change that happens in terms of number of cycles.

And then we worked out what happens for a different 5 stage and a 12 stage pipeline, and the stalls that are happening. And this is basic performance equation. And we also see what is how this Amdahl's law is helpful for us in order to efficiently manage our given constraints of different modification versions. So, with this we conclude today's tutorial.

Thank you.