Multicore Computer Architecture - Storage and Interconnects

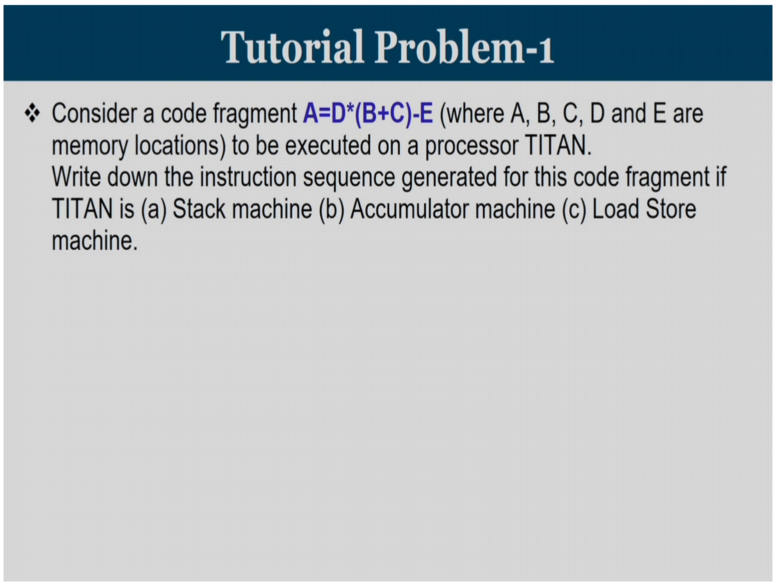
Dr. John Jose
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati, Assam

Lecture – 05
Instruction Pipeline & Performance - I

Welcome to the tutorial session, we have already completed 3 lectures that talks about the fundamentals of computer architecture as well as the basic principles of instruction pipeline. Today's session is planned such that you will be familiar with the certain numerical problems on the topics that these already covered in lecture1, 2 and 3. I will be solving these numerical problems, so that you will get a better idea of how to understand the topic in depth. We will be covering topics related to performance of processor design including Amdahl's law and then some fundamental concepts in instruction pipeline and the certain aspects related to the hazards and about multi cycle floating point pipeline operations.

So, we will be having such kind of similar tutorial exercises every week, this will help you in deeper understanding the topic. So, I request you to go through the sample questions that is been already posted in the forum and get familiarized with those questions. The assignments that are plotted every week will be somewhat in sync with the kind of questions there we solve in tutorial exercises. So, let us move to the first question.

(Refer Slide Time: 02:01)

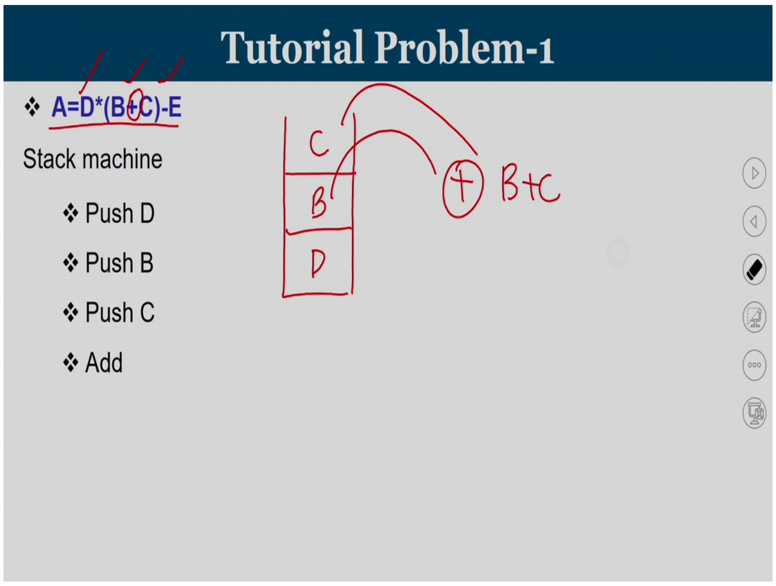


Tutorial Problem-1

- Consider a code fragment A=D*(B+C)-E (where A, B, C, D and E are memory locations) to be executed on a processor TITAN. Write down the instruction sequence generated for this code fragment if TITAN is (a) Stack machine (b) Accumulator machine (c) Load Store machine.

Consider a code fragment A equal to D star B plus C minus E; where, A, B, C, D and E are memory locations. So, we have to execute this code on a processor called TITAN. Now we have to write down the sequence of instructions, the instruction sequence that is generated for this code fragment; if TITAN is going to be a stack machine or it is going to be an Accumulator machine and the third one if it is a Load and the Store machine.

(Refer Slide Time: 02:36)



Tutorial Problem-1

- A=D*(B+C)-E

Stack machine

- Push D
- Push B
- Push C
- Add

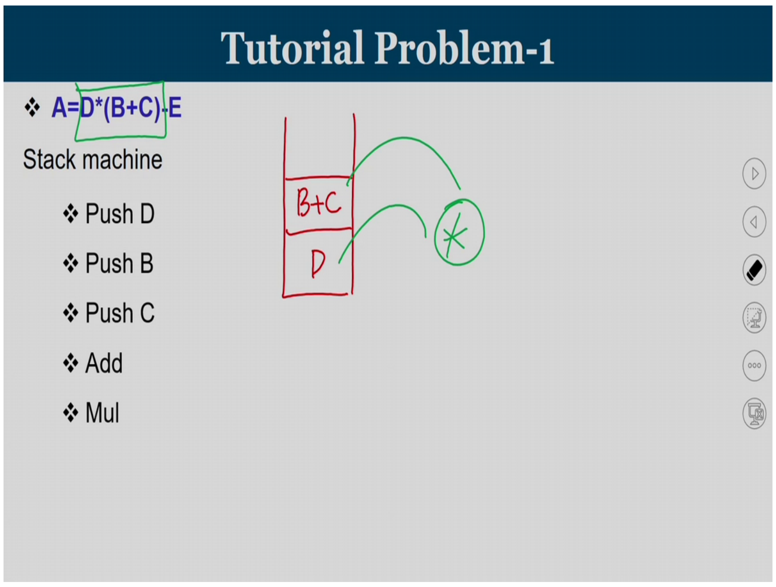
So, this is the code that is given A assigned equal to D star B plus C. So, if this is going to be the code how a Stack machine is going to operate on it? We know that in the case of a stack machine, we have a stack that is from memory, you have to push the operands into the stack and whenever we wanted to perform an operation, the operation will be carried out on the top of the stack and the very next element after the top of this stack.

So, when you look at this particular, example, we know that A equal to D star B plus C since a bracket is given B star C is to be done first and we know that we cannot directly perform any add or subtract operation. Any operation that is done on a stack machine, the operands are implied they are on the stack. The two top elements on the stack will be the operands. So, the B plus C is the first operation to be done. So, before doing this plus operation, we wanted to make sure that both B and C are there in the stack.

So, the first operation that we will do is we have to push the value of D then we have to push the value for B. So, now if you look at this stack, the stack may contain D then followed by the stack may has B unit then we are going to push the third operand C. So, now your top of the stack is C and top minus 1 is B. Now, we have sufficient operands to perform the operation, so we can perform the operation, the operation is plus. So, we here going to perform an add operation.

Once add is done, then what happens is you are going to push out these two elements and we are going to perform B plus C and the sum of B plus C is going to be added into the stack.

(Refer Slide Time: 04:55)



Tutorial Problem-1

- A=D*(B+C)-E

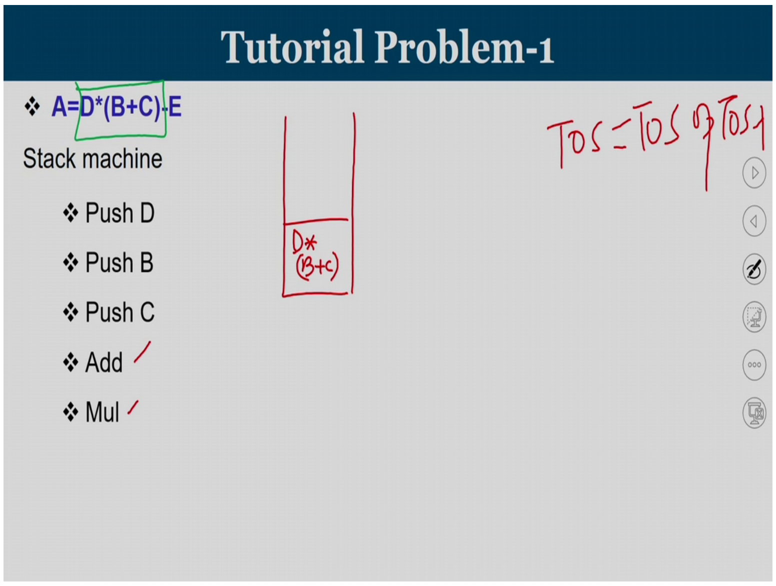
Stack machine

- Push D
- Push B
- Push C
- Add
- Mul

So, B plus C, once you perform A that B and C adding operation, the adding operation then B plus C gets stored into the stack. So, the top of the stack is know B plus C. Now, we know that as per the sequence that is given we have to perform D star the sum of that. So, that is operation to be performed. So, if you wanted to perform that operation, then we have to work on the multiplication aspect.

So, next is the product operation that is to be done. So, you when you perform multiplication these two elements are popped out and the star operation is carried out. So, in this context let us try to see that D star B plus C that the product is going to be added into this stack.

(Refer Slide Time: 05:41)



Tutorial Problem-1

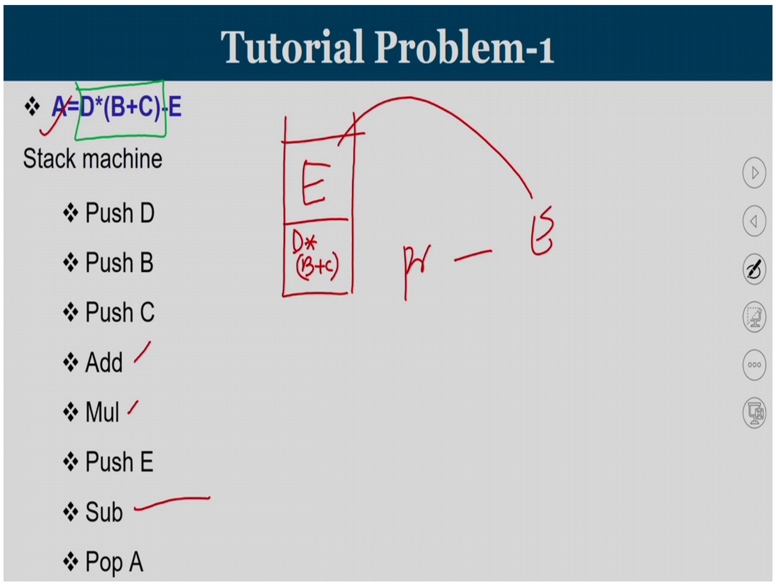
- A=D*(B+C)-E

Stack machine

- Push D
- Push B
- Push C
- Add
- Mul

So, we have now the latest value in stack is, D star B plus C which is already saved in the stack and there is only one element in the stack. So, we have to understand that when you perform any operation, let us say it is an add operation or a multiplication operation that top of this stack is defined as top of the stack operator, top of the stack minus 1; that is the operation that is carried out here. So, now we have to push the next element.

(Refer Slide Time: 06:09)


Tutorial Problem-1

- A=D*(B+C)-E

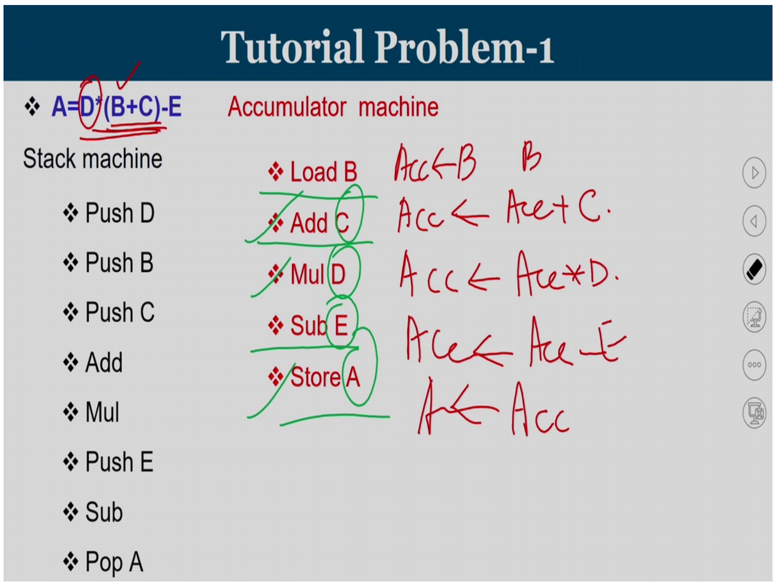
Stack machine

- Push D
- Push B
- Push C
- Add
- Mul
- Push E
- Sub
- Pop A

So, our next operation is going to be push E. So, the result of this is A E is going to be pushed on to this stack and now we have this is going to pop out that is what we get and we get the product value and then product minus E that is what this we going to do. So, when you perform a subtraction operations, E is going to be popped out the product value that is D star B plus C is going to be popped out and then you perform the operation and then we are going to push the result back.

So, and then at the end once this is over at the end of this subtraction operation the resultant value is stored into that. Now our final value should be available inside A. So, the last operation that we need to carry out is pop of A. So, that completes our code generation for the stack machine. Now we try to understand as given in the question what is going to be for an Accumulator machine.

(Refer Slide Time: 07:07)



Tutorial Problem-1

❖ A=D*(B+C)-E

Stack machine

❖ Push D
❖ Push B
❖ Push C
❖ Add
❖ Mul
❖ Push E
❖ Sub
❖ Pop A

Accumulator machine

❖ Load B
❖ Add C
❖ Mul D
❖ Sub E
❖ Store A

Acc←B B
Acc ← Acc+C.
Acc ← Acc*D.
Acc← Acc-E
A← Acc

Let us try to understand what is an Accumulator machine? Accumulator machine means for every operation one of the operand, the first operand is going to be the accumulator and the resultant is also going to be Accumulator, the second operand can be a memory location and there are operation called Load and Store. Load operation means, Load of x means condense of memory location x is copied to accumulator; similarly, store of x means condense of accumulator is stored into memory location x.

So, in the question we have this been specified that A D B C and E, all are memory location. So, we have to make sure that before performing an operation in we have to load one of the operand into the Accumulator. So, the first operation that we will do is load B. So, B is the value that we have. So, now, with this means B value is copied to Accumulator. So accumulator now carries the value of B and now we are going to perform add of C. So, then this means accumulator assigned is equal to whatever be the value of accumulator plus C because it is an add operation. So, already we have value B in the Accumulator. So, it is B plus E. So, and the resultant value of B plus C is stored in the Accumulator.

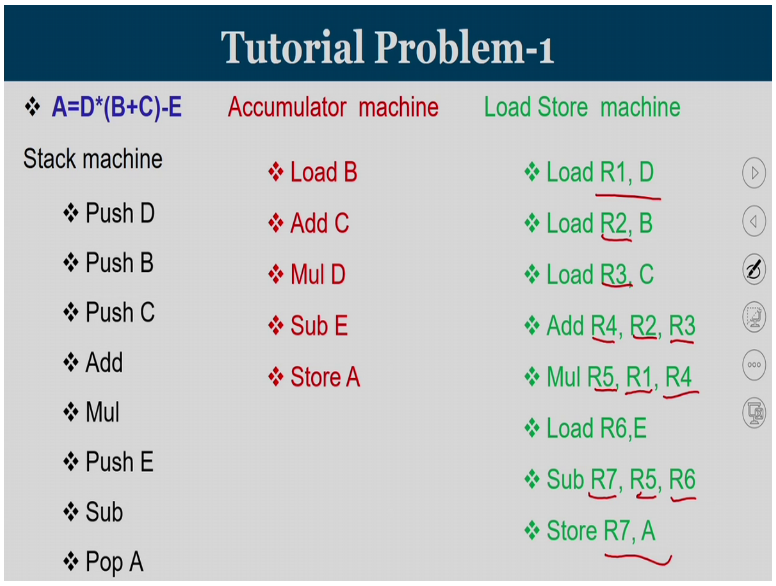
And now since this portion is already done, I can perform a multiplication operation with D and that is what is going to happen. So, with this we are going to have accumulator is equal to accumulator star D that is going to be the operation in multiplication. So, we get now D star B plus C in the accumulator and now we are going to subtract E, so, then accumulator C equal to accumulator minus E. So, the resultant value is now in accumulator and towards the end we are going to store the value. So, then; that means, A is assigned to the value of Accumulator.

So, in this way first you load the first operand into the accumulator that is what we see with the help of load B instruction the content of B is getting loaded to accumulator and when you perform add C whatever is there in accumulator that is added with the condense of memory location C. So, any operation let say it is add multiplication or subtraction whatever the operand that we have it is a memory operand.

So, one of the operand is an accumulator the other one is in memory location and their name of the memory location is specified in the instruction. So, once you perform add and C then you perform the subtraction operation and the final result is going to be stored into A. So, this is how the code there is been generated for the accumulator machine.

Now, the third category of instruction type that we have to generate is for Load Store machine.

(Refer Slide Time: 10:07)



Tutorial Problem-1

❖ A=D*(B+C)-E

Stack machine	Accumulator machine	Load Store machine
❖ Push D	❖ Load B	❖ Load R1, D
❖ Push B	❖ Add C	❖ Load R2, B
❖ Push C	❖ Mul D	❖ Load R3, C
❖ Add	❖ Sub E	❖ Add R4, R2, R3
❖ Mul	❖ Store A	❖ Mul R5, R1, R4
❖ Push E		❖ Load R6,E
❖ Sub		❖ Sub R7, R5, R6
❖ Pop A		❖ Store R7, A

We have to understand that in the case of a Load Store machine, arithmetic operations can be performed only between two registers. So, in the case of stack architecture both the operands are in the Stack, in the case of accumulator architecture one of the operand is an accumulator and the other one is in memory. In the case of Load Store architecture both the operands, both the source operand should be there in the register and then only you can perform operation. So, no operation is allowed with a memory operand in it.

So, let us try to see what happens, In this case we have to define few registers. So, one of the one that we are going to use is R 1 R 2 R 3 are going to be kind of the register naming that we use in this example. So, the content of D is loaded to R 1,content of B is loaded to R 2 and C is loaded to R 3. So, now, I have these three operand values already available in the registers.

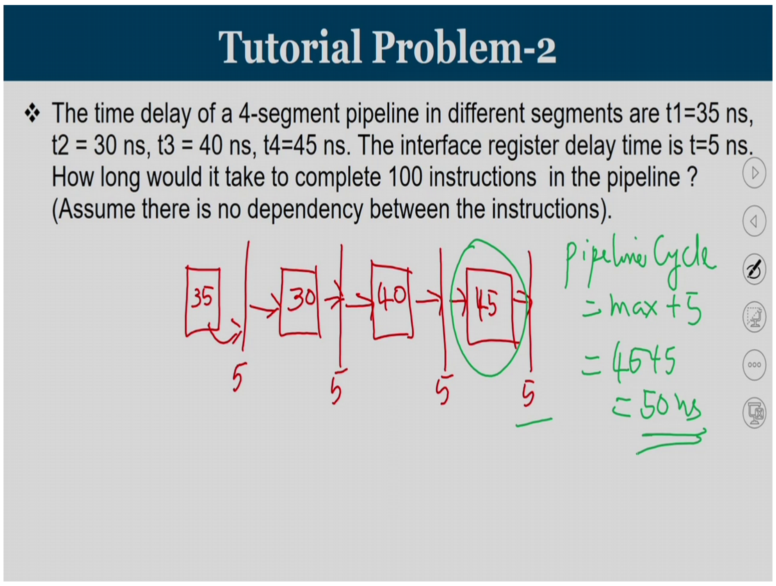
Now, the first operation we need to carry out is B plus C, we know that B is available in R 2 and C is available in R 3. So, an add operation is performed on R 2 and R 3 and the resultant value is now available in R 4. So, this whole value is now available in R 4. Now R 4 value should be multiplied with R 1 value, we know that R 1 is having the value of D. So, multiplication of R 1 and R 4 and that will give you the complete, product of these things in R 5.

So, now we have this much component available inside R 5. Now we have to load the value of E into one more register. So, load R 6 into that is value of E is loaded to R 6 and once we have the result then we have to subtract the value of R 5 and R 6 and the resultant is stored in R 7 and then the last property is store the value of R 7 into A.

So, when you look at this sequence of instructions it is easy for us to understand that all the operands has to be loaded into registers. In this case we are using all new registers in every context for better reuse of registers. If you know that value of a register is no longer required then I can reuse the register. So, depending on the underlying architecture let us say it is a stark architecture the code that the compiler has to generate is different. If it is an accumulator architecture the code that compiler is generating is different and for a Load Store architecture it is yet another code.

So, given the same sequence of instruction what we have seen A is equal to D star B plus C for the same instruction the kind of code that is generated by the compiler is different because hardware understands different language. So, this is an initial problem with gives you a deeper understanding about in what way the code that a stack architecture and accumulator architecture and a Load Store architecture is working with.

(Refer Slide Time: 13:20)



Tutorial Problem-2

❖ The time delay of a 4-segment pipeline in different segments are t1=35 ns, t2 = 30 ns, t3 = 40 ns, t4=45 ns. The interface register delay time is t=5 ns. How long would it take to complete 100 instructions in the pipeline ? (Assume there is no dependency between the instructions).

35
30
40
45
5
5
5
5
Pipeline Cycle
= max + 5
= 45 + 5
= 50 ns

Now, I request you to consider the second problem. This is an pipeline associated problem, time delay of a four segment pipeline in different segments are t 1 equal to 35 nanoseconds, t 2 is equal to 30 nanoseconds, t 3 is equal to 40 nanosecond then the t 4 is equal to 45 nanosecond. The interface register delay time is t equal to 5 nanosecond. How long would you take to complete 100 instructions in the pipeline? Assume there is no dependency between the instructions.

So, this is a case where we are going to understand about how to design a pipeline. So, to get better clarity let me try to explain, these are the 4 segments inside your pipeline and the first one is taking 35 nanosecond, the second one is taking 30 nanosecond, third is taking 40 nanosecond and the last one is taking 45 nanosecond. We have learned in the pipeline that even if different segments in a pipeline is going to take different amount of time pipeline can advance only together.

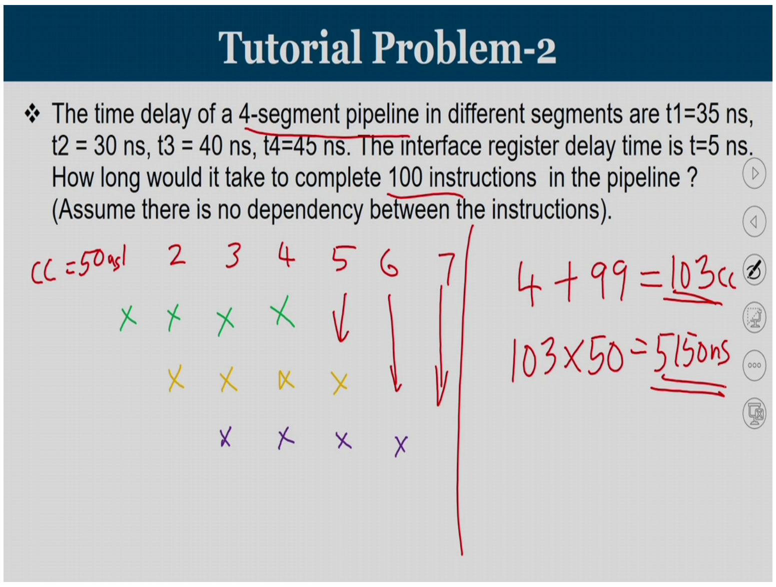
So, we are going to add the interface registers between them and interface register is going to consume 5 nanoseconds time. So, the content of this will be returned to the interface register and that is going to be used by the second one and that is a way how the pipeline is going to proceed, but we have to make sure that all the values are taken at the same time since the delay associated with each of the segment this varying we have to find out which is the one that is dominating.

So, out of this four segments this is the segment which is having the largest time unit. So, my pipeline time is going to be governed by there is pipeline cycle time is maximum among or the cases plus the interface time that is 5. So, it is 45 plus 5; that is 50 nanosecond is the timing at which this pipeline is going to operate.

So, when we get different time units for different segments we have to pick that one which is the maximum or which is a dominating one and then add the interface time. So, at every 50 nanoseconds you are going to read from the interface register perform the operation in the segment and going to write the result but you may write the result early because the first unit may write the result in 35 nanosecond where a second will write in 30 nanosecond whatever time you are going to write the next triggering will happen only at 50 because a last unit will be able to complete writing by 45 nanoseconds. So, 50 nanosecond is the time at which we are going to operate the cycle.

Now, we will do about how will you take care of the pipeline aspect of 100 instructions 100 independent instruction by what time these instructions are going to be completed? So, we will try to understand what is the context here.

(Refer Slide Time: 16:47)



Tutorial Problem-2

- The time delay of a 4-segment pipeline in different segments are t1=35 ns, t2 = 30 ns, t3 = 40 ns, t4=45 ns. The interface register delay time is t=5 ns. How long would it take to complete 100 instructions in the pipeline ? (Assume there is no dependency between the instructions).

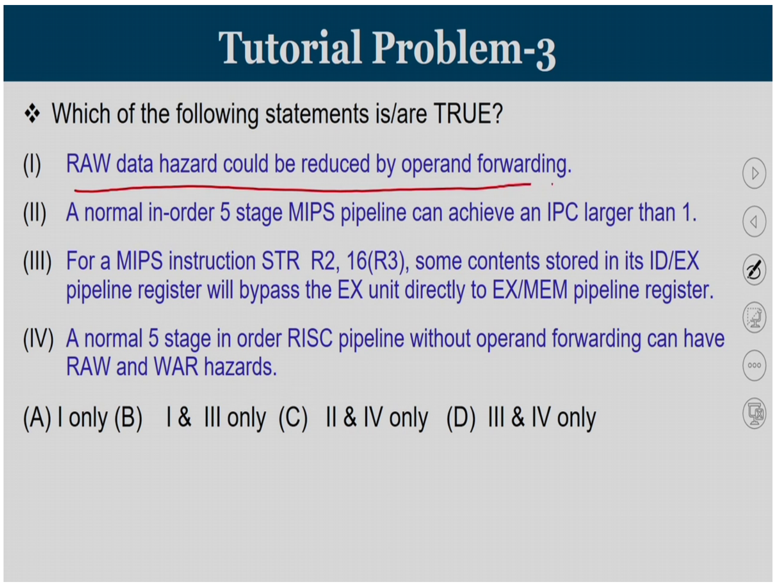
Let us say clock cycle number 1, 2, 3, 4, 5, 6, 7 like that it is going. It is a 4 stage pipeline and we know that every clock cycle, one clock cycle is equal to 50 nanosecond that is what we have found out now.

The first instruction because it is a 4 segment pipeline first instruction get complete at the forth clock cycle. So, it goes through the 4 stages and it gets complete in the forth clock cycle, the second instructions in this pipeline it is going to start at the second clock cycle it get over at the fifth clock cycle, the third instruction is going to start at the third clock cycle and it gets over at the 6th clock cycle. This is going to be the sequence in which the instructions are going to progress.

So, if you try to generalize this we have 100 instructions, the first instruction is getting over at the forth clock cycle and thereafter everyone clock cycle one more instruction is getting over. So, we have another 99 instructions more out of the 100 instruction first instruction is getting over at the fourth clock cycle because it is a 4 segment pipeline and thereafter for every clock cycle one more instruction is getting over. So, 100 instructions get over in 103 clock cycles and we know that one clock cycle is going to be 50 nanosecond then 103 instructions will get over at 5150 nanosecond.

So, it will take 5150 nanoseconds to complete this operation. So, that is a way how we are going to deal with pipelines of varying time segments.

(Refer Slide Time: 18:47)



Tutorial Problem-3

❖ Which of the following statements is/are TRUE?

(I) RAW data hazard could be reduced by operand forwarding.

(II) A normal in-order 5 stage MIPS pipeline can achieve an IPC larger than 1.

(III) For a MIPS instruction STR R2, 16(R3), some contents stored in its ID/EX pipeline register will bypass the EX unit directly to EX/MEM pipeline register.

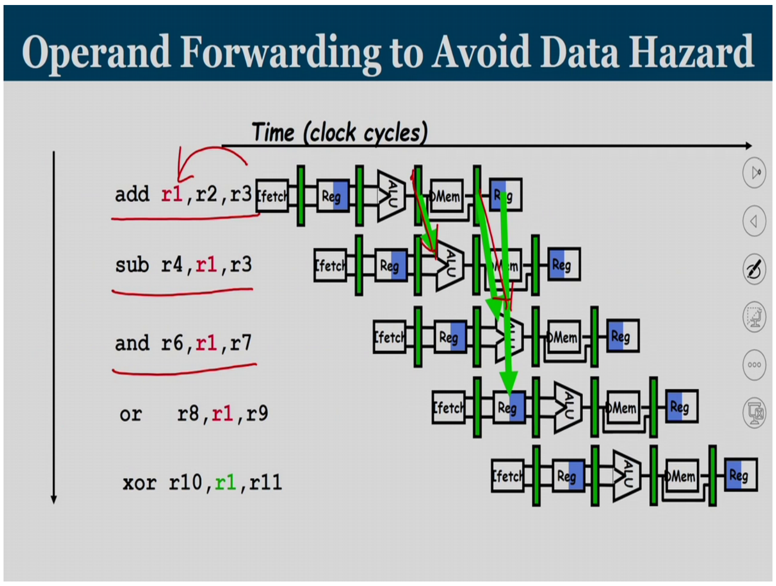
(IV) A normal 5 stage in order RISC pipeline without operand forwarding can have RAW and WAR hazards.

(A) I only (B) I & III only (C) II & IV only (D) III & IV only

.

Now, this is a case of a typical True or False question. Which of the following statements is or are True? There are four examples given. So, only if you clearly understand the pipeline concept you may be able to tell this. Let us try to understand each one of them. Which of the following statements are True? There are four statements given, the first one is RAW hazard, RAW data hazard could be reduced by operand forwarding that is a first statement that we have.

(Refer Slide Time: 19:19)

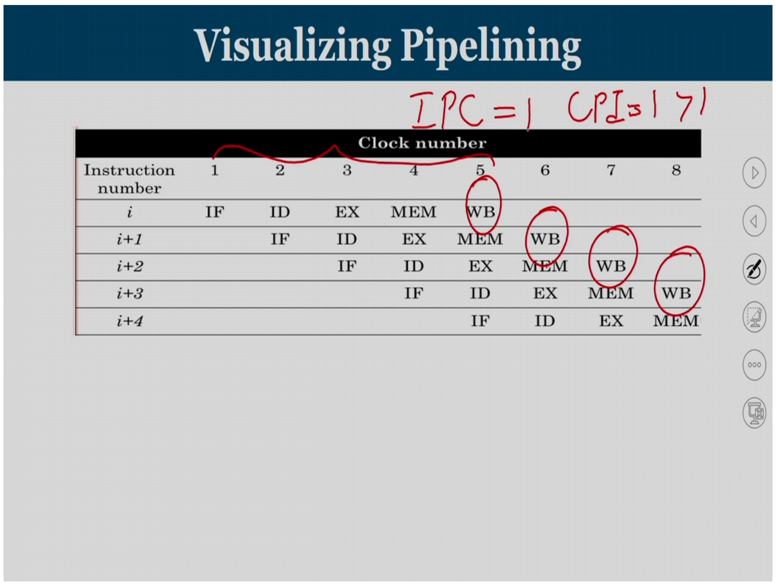


Operand Forwarding to Avoid Data Hazard

So, if you look at that this is the concept of operand forwarding whenever we have a data hazard let us say in this case we are going to write the result into r 1. So, r 2 and r 3 context are going to be added into and stored in r 1 and that value of r 1 is needed in the subtraction instruction and the add instruction. So, operand forwarding is something where whenever the result is ready the content of r 2 plus r 3 is ready here that is forwarded into ALU such that the subtraction instruction can progress.

So, in this way whenever we do operand forwarding the stalls are eliminated. So, what we see here is a RAW data hazard could be reduced by operand forwarding. So, that is a true statement. A normal in order 5 stage MIPS pipeline can achieve an IPC larger than 1. So, what do we, we have to understand what do you mean by normal in order 5 stage pipeline and what is the meaning of IPC? So,

(Refer Slide Time: 20:21)



Visualizing Pipelining

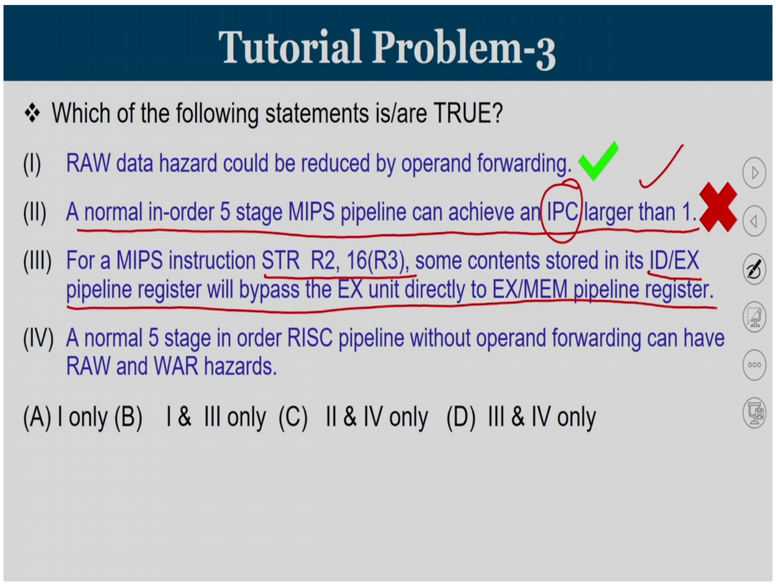
IPC = 1 CPI = 1 7/

Instruction number	Clock number 1	2	3	4	5	6	7	8
i	IF	ID	EX	MEM	WB			
i+1		IF	ID	EX	MEM	WB		
i+2			IF	ID	EX	MEM	WB	
i+3				IF	ID	EX	MEM	WB
i+4					IF	ID	EX	MEM

If you look at an in order pipeline it is the 5 stage in order pipeline. So, each instruction will go through 5 different stages IF, ID, EX, MEM and write back. Since it is an in order pipeline, the second instruction will start only after the first instruction is fetched issued. So, the sequence is never violated.

Now, how many instructions are getting complete, we can see that in the fifth clock cycle one instruction is getting over in 6, one instruction is getting over in 7, one instruction is getting over. So, in every cycle one instruction is getting over leading to a CPI value of one. If there are hazards then CPI cycles per instruction will be slightly larger than one. So, IPC is the value which is the reciprocal of CPI instructions per cycle. So, number of instructions completed per cycle is equal to 1 in the case of a normal in order ideal pipeline.

(Refer Slide Time: 21:20)



Tutorial Problem-3

❖ Which of the following statements is/are TRUE?

(I) RAW data hazard could be reduced by operand forwarding. ✓

(II) A normal in-order 5 stage MIPS pipeline can achieve an IPC larger than 1. ✗

(III) For a MIPS instruction STR R2, 16(R3), some contents stored in its ID/EX pipeline register will bypass the EX unit directly to EX/MEM pipeline register.

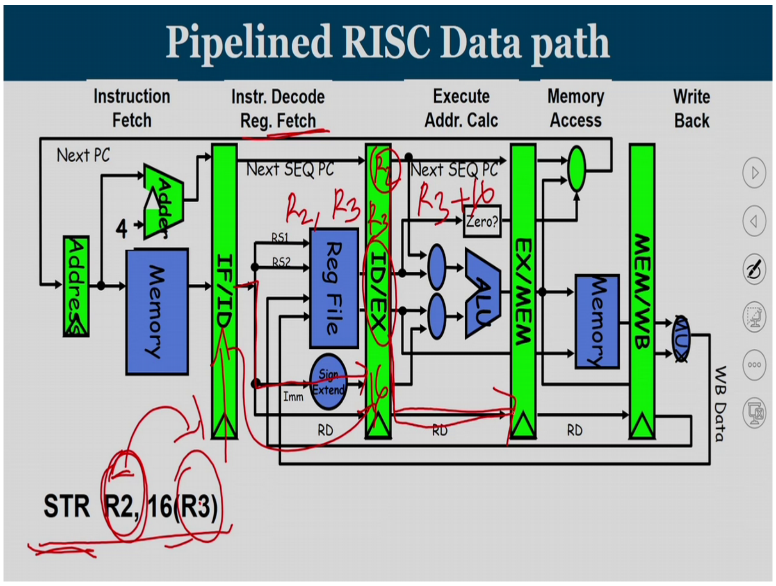
(IV) A normal 5 stage in order RISC pipeline without operand forwarding can have RAW and WAR hazards.

(A) I only (B) I & III only (C) II & IV only (D) III & IV only

So, the statement is mentioning that a normal in order 5 stage MIPS pipeline can achieve an IPC larger than 1, Instruction Per Cycle larger than 1. Can we complete more than 1 instructions per clock cycle? That is not possible in the case of an in order 5 stage pipeline, such cases are possible only if it is an out of order superscalar pipeline. So, the second statement is faults.

Now, we look at the third statement for a MIPS instruction store R2 16 of R 3, some condense stored in its ID EX pipeline register will bypass the EX unit directly to the EX MEM pipeline to register.

(Refer Slide Time: 22:04)

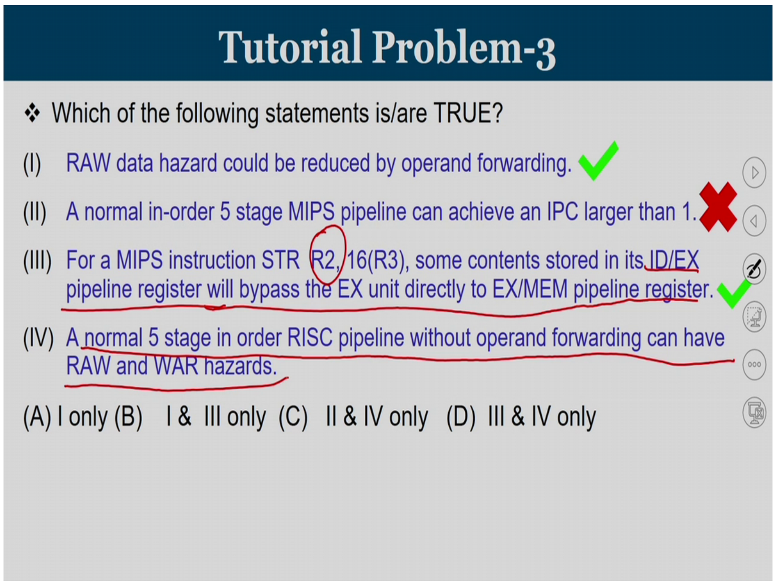


To understand that let us revisit the pipeline diagram, we have a store instruction R 2 16 of R 3. Let us try to understand what happens in each of the pipeline stages in the execution of this instruction. So, in the instruction fetch stage this entire instruction the binary value of the entire instruction is loaded into the IF ID register.

Now, any reading from the register has to be performed in the register fetch stage that is a second stage. So, in this case we have two registers R 2 and R 3. The meaning of the instruction is the content of register R 2 should be written to a memory address that is pointed by 16 of R 3. So, content of R 2 is red similarly in order to compute address content of R 3 also has to be red. So, values of R 2 and R 3 are now in the ID register and this value 16 will also be there 16 will come through this. 16 would not go through the register file this is the path by which 16 is going to travel.

Now, R 2 value is not needed in the ALU stage that is needed in order to write into the memory. So, R 2 value is directly going to come into this stage. Whatever is the value of R 3 it is added with 16. So, R 3 plus 16 is carried out here whereas, R 2 is going to bypass the ALU for in the case of a store instruction the value to be stored is going to bypass the ALU that will reach the ID EX register. So, some contents of the ID EX register is going to bypass the ALU into the EX MEM register.

(Refer Slide Time: 23:59)



Tutorial Problem-3

❖ Which of the following statements is/are TRUE?

(I) RAW data hazard could be reduced by operand forwarding. ✓

(II) A normal in-order 5 stage MIPS pipeline can achieve an IPC larger than 1. ✗

(III) For a MIPS instruction STR R2, 16(R3), some contents stored in its ID/EX pipeline register will bypass the EX unit directly to EX/MEM pipeline register. ✓

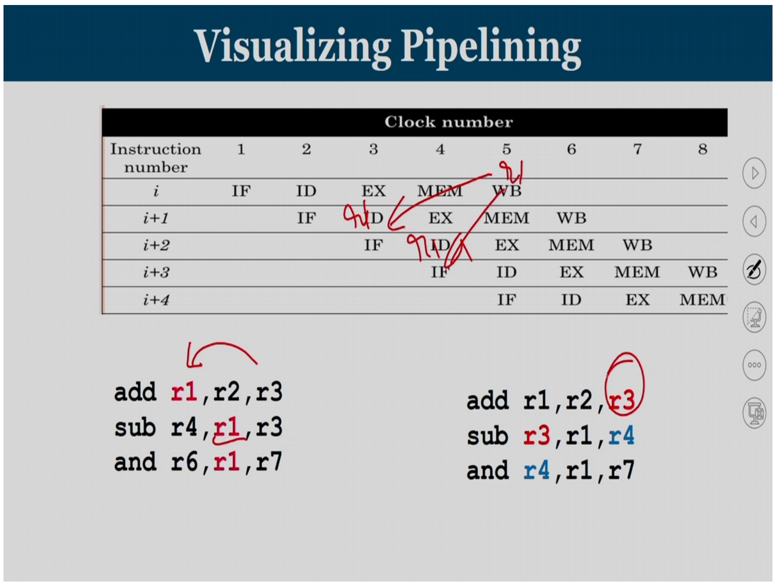
(IV) A normal 5 stage in order RISC pipeline without operand forwarding can have RAW and WAR hazards.

(A) I only (B) I & III only (C) II & IV only (D) III & IV only

Now, if you look at, will bypass the EX unit directly to the EX MEM. So, from the ID EX pipeline register, the value of R 2 is going to bypass the EX unit, the value of R 2 will not go to the EX unit that is what we are going to see. So, this is true.

Now let us try to understand what is the fourth instruction.

(Refer Slide Time: 24:25)



Visualizing Pipelining

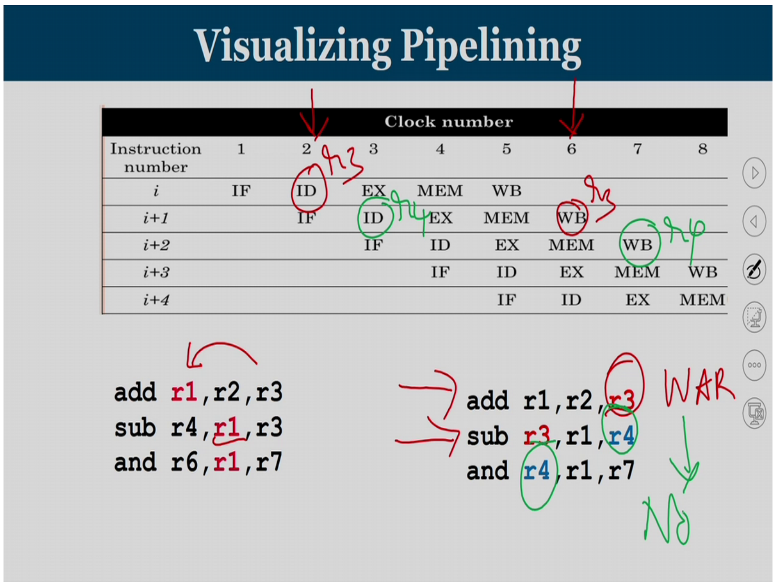
	Clock number							
Instruction number	1	2	3	4	5	6	7	8
i	IF	ID	EX	MEM	WB			
i+1		IF	ID	EX	MEM	WB		
i+2			IF	ID	EX	MEM	WB	
i+3				IF	ID	EX	MEM	WB
i+4					IF	ID	EX	MEM

```
add r1,r2,r3
sub r4,r1,r3
and r6,r1,r7
```

```
add r1,r2,r3
sub r3,r1,r4
and r4,r1,r7
```

A normal 5 stage in order RISC pipeline without operand forwarding can have RAW and WAR hazards. So, what do you will see here is we have a there is no operand forwarding here so can we have RAW hazards? If you look at this case RAW hazard means you are going to write the value into r 1 and that value 1 is going to be red. So, writing into r 1 happens here and we need the value of r 1 at these stages. So, you have to forward this value. So, you are going to have a hazard here in the case of RAW because we are not using operand forwarding. Whereas, on the right side we see it is a WAR hazard you are going to read a value from r 3 and the second instruction is going to write a value into the r 3.

(Refer Slide Time: 25:20)


r3
r4
r3
r4
WAR
No

Visualizing Pipelining

Instruction number	Clock number							
	1	2	3	4	5	6	7	8
i	IF	ID	EX	MEM	WB			
i+1		IF	ID	EX	MEM	WB		
i+2			IF	ID	EX	MEM	WB	
i+3				IF	ID	EX	MEM	WB
i+4					IF	ID	EX	MEM

```
add r1,r2,r3
sub r4,r1,r3
and r6,r1,r7
```

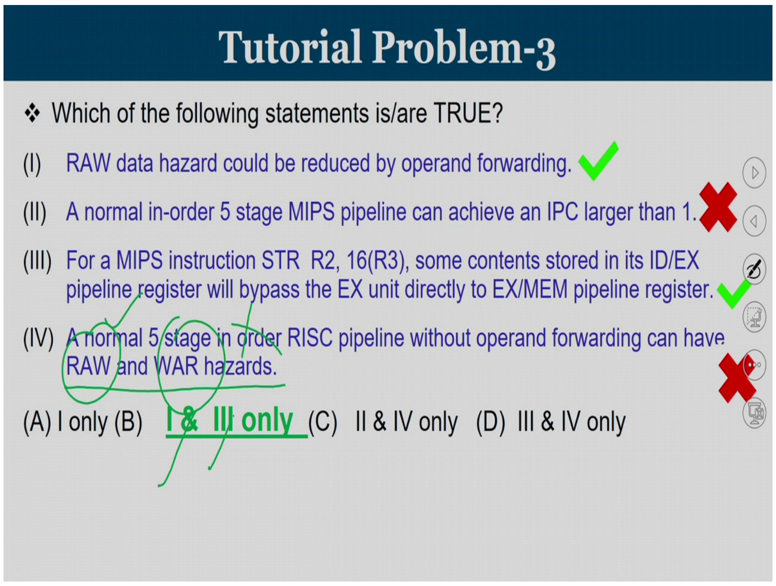
```
add r1,r2,r3
sub r3,r1,r4
and r4,r1,r7
```

So, when we work on WAR hazard, these are typical cases of WAR hazards right after read, you are going to read from r 3 that happens here in the second clock cycle and you are going to write the value into r 3 and that happens here.

So, in the normal 5 stage in order pipeline reading from r 3 for the first add instruction happens in clock cycle number 2 and writing into r 3 by the subtraction instruction happen in clock cycle number 6. So, it is not going to have any kind of violation. So, this would not create any issue as of now. Now try to understand what happens in the case of r 4. So, r 4 you are going to write the value, you are going to read the value of r 4 here and we are going to write into r 4 by the and instruction. So, that is a place where you are going to r.

So, WAR, hazards are not there, there is no WAR hazards,, but there can be RAW hazards.

(Refer Slide Time: 26:17)



Tutorial Problem-3

❖ Which of the following statements is/are TRUE?

(I) RAW data hazard could be reduced by operand forwarding. ✓

(II) A normal in-order 5 stage MIPS pipeline can achieve an IPC larger than 1. ✗

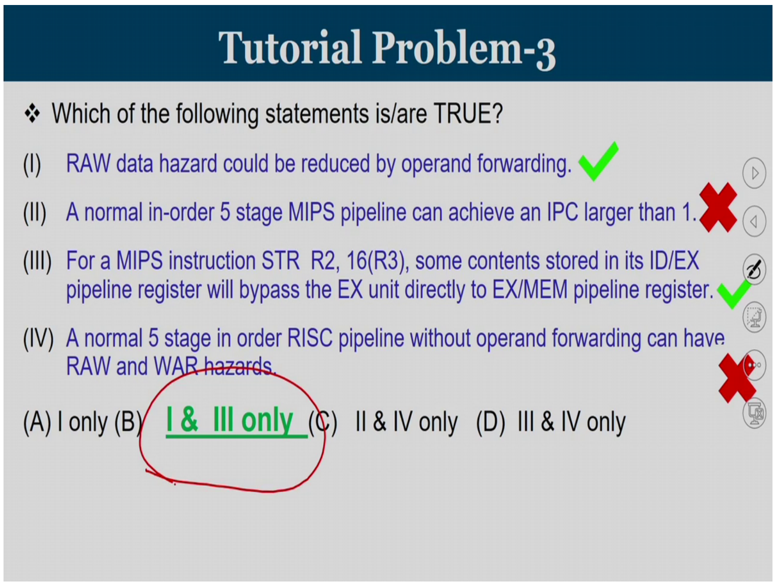
(III) For a MIPS instruction STR R2, 16(R3), some contents stored in its ID/EX pipeline register will bypass the EX unit directly to EX/MEM pipeline register. ✓

(IV) A normal 5 stage in order RISC pipeline without operand forwarding can have RAW and WAR hazards. ✗

(A) I only (B) **I & III only** (C) II & IV only (D) III & IV only

So, a normal 5 stage in order RISC pipeline without operand forwarding can have RAW and WAR hazards, this statement is faults you can have RAW hazards but you will never have WAR hazards, so, altogether if you look at statement number 4, that is wrong. So, what is correct is question number 1 and three option, these are the two options that are correct in this scenario. So, the correct answer is 1 and 3 that is already mentioned.

(Refer Slide Time: 26:51)



Tutorial Problem-3

❖ Which of the following statements is/are TRUE?

(I) RAW data hazard could be reduced by operand forwarding. ✓

(II) A normal in-order 5 stage MIPS pipeline can achieve an IPC larger than 1. ✗

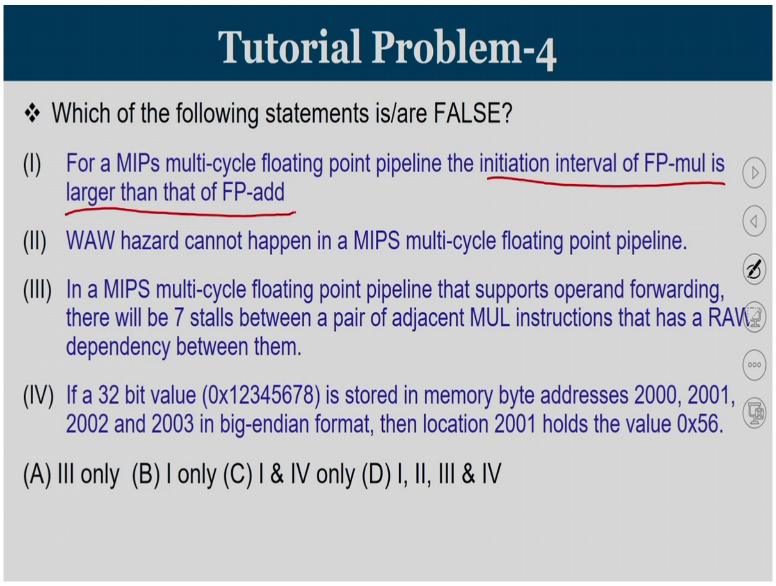
(III) For a MIPS instruction STR R2, 16(R3), some contents stored in its ID/EX pipeline register will bypass the EX unit directly to EX/MEM pipeline register. ✓

(IV) A normal 5 stage in order RISC pipeline without operand forwarding can have RAW and WAR hazards. ✗

(A) I only [✓ (B) **I & III only**] (C) II & IV only (D) III & IV only

So, only if you have a deeper understanding about the various operations that is happening inside an instruction pipeline then only you will be able to solve this.

(Refer Slide Time: 27:05)



Tutorial Problem-4

❖ Which of the following statements is/are FALSE?

(I) For a MIPs multi-cycle floating point pipeline the initiation interval of FP-mul is larger than that of FP-add

(II) WAW hazard cannot happen in a MIPS multi-cycle floating point pipeline.

(III) In a MIPS multi-cycle floating point pipeline that supports operand forwarding, there will be 7 stalls between a pair of adjacent MUL instructions that has a RAW dependency between them.

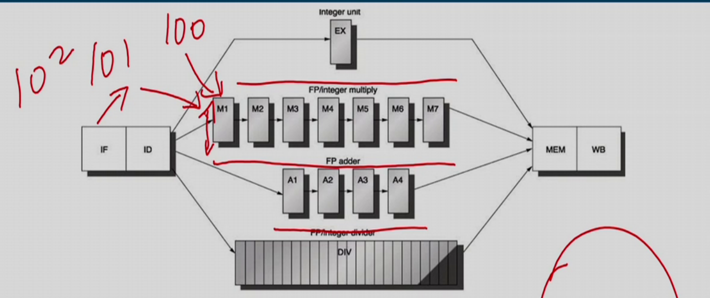
(IV) If a 32 bit value (0x12345678) is stored in memory byte addresses 2000, 2001, 2002 and 2003 in big-endian format, then location 2001 holds the value 0x56.

(A) III only (B) I only (C) I & IV only (D) I, II, III & IV

Now, we have one more true or false case, which of the following statements are False? For a MIPS multi cycle floating point pipeline the initiation interval of floating point multiplication is larger than floating point addition. So, how will you tackle this statement? There is a floating point multiplication which we have seen that it has 7 pipeline stages; M 1, M 2, M 3, M 4, M 5 like that these is the floating point pipeline, M 1 to M 7 and floating point adding as A 1, A 2, A 3, A 4.

(Refer Slide Time: 27:39)

Multi-cycle Operations



Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1 ✓
FP multiply (also integer multiply)	6	1 ✓
FP divide (also integer divide)	24	25

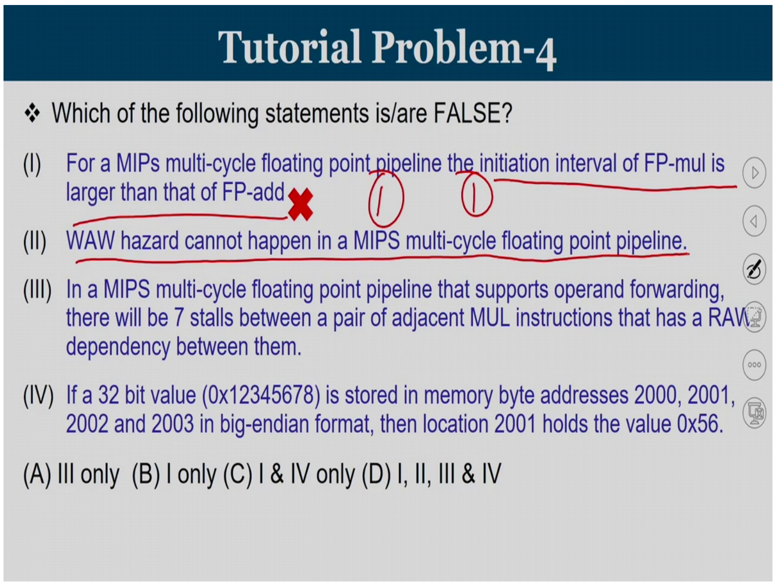
Latencies and initiation intervals for functional units.

The initiation interval of floating point multiplication is it larger than floating point addition for that we have to understand. What do you mean by the term initiation interval? Initiation interval means what should be the minimum gap between two operations; that is going to use the same functional unit, let say I am giving an operation in floating point multiplication and clock cycle 100, in clock cycle 100 a pair of floating point numbers are multiplied are we are going to start multiplication.

Initiation one interval of floating point multiply is given as 1, the meaning is at 101, another no pair of numbers can be added into the unit, at 102 another no pair of numbers can be given in the multiplier unit. Since the multiplication unit is pipelined as long as there is no data dependency between the numbers that are going to be multiplied and the numbers that are there already in the multiplication unit; that means, any new pair of numbers can be given to the multiplier unit in adjacent clock cycles that is what the initiation interval of one.

Whereas if you look at the division unit you have an initiation interval of 25; that means, you can give the next number to be divided only after the 25th clock cycle similarly when you look at adding also since adding unit is also a pipeline that every cycle new pair of numbers, new pair of floating point numbers can be given in the floating point adder. So, initiation interval of floating point add also is 1 and floating point multiplication is also 1.

(Refer Slide Time: 29:34)



Tutorial Problem-4

❖ Which of the following statements is/are FALSE?

(I) For a MIPs multi-cycle floating point pipeline the initiation interval of FP-mul is larger than that of FP-add ✖ 1 1

(II) WAW hazard cannot happen in a MIPS multi-cycle floating point pipeline.

(III) In a MIPS multi-cycle floating point pipeline that supports operand forwarding, there will be 7 stalls between a pair of adjacent MUL instructions that has a RAW dependency between them.

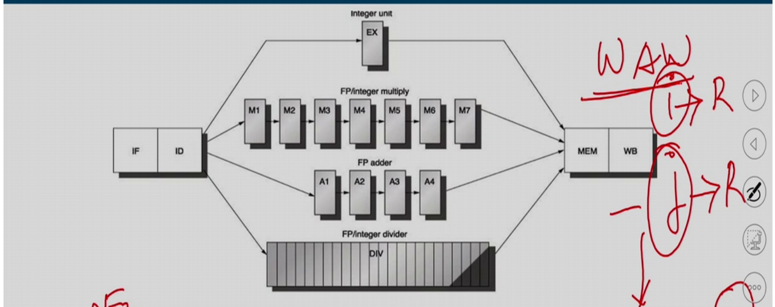
(IV) If a 32 bit value (0x12345678) is stored in memory byte addresses 2000, 2001, 2002 and 2003 in big-endian format, then location 2001 holds the value 0x56.

(A) III only (B) I only (C) I & IV only (D) I, II, III & IV

So, the statement for a MIPS multi cycle floating point pipeline initiation interval of floating point MUL is larger than that of floating point add is wrong because both has initiation interval of one. So, statement number 1 is false. Look at statement number 2, WAW hazard cannot happen in MIPS multicycle floating point pipeline, what you mean by WAW hazard write after write hazard if the second instruction is going to write on a value where the first instruction is also going to write.

(Refer Slide Time: 30:09)

Multi-cycle Operations



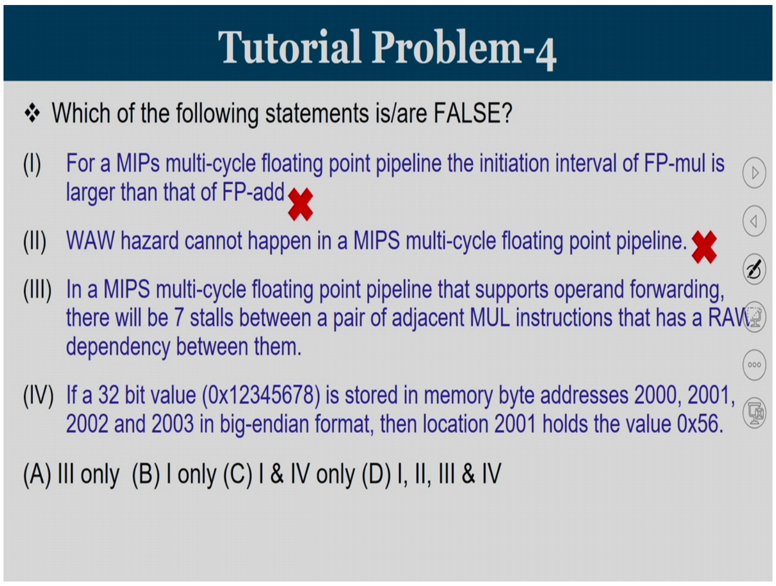
MUL.D F2	IF	ID	*M1*	M2	M3	M4	M5	M6	M7	MEM	WB
ADD.D F2		IF	ID	*A1*	A2	A3	A4	**MEM**	WB		
L.D			IF	ID	*EX*	**MEM**	WB				
S.D				IF	ID	*EX*	*MEM*	WB			

The pipeline timing of a set of independent FP operations. The stages in italics show where data are needed, while the stages in bold show where a result is available. The ".D" extension on the instruction mnemonic indicates double-precision (64-bit) floating-point operations. FP loads and stores use a 64-bit path to memory so that the pipelining timing is just like an integer load or store.

So, consider the case that let us say you are going to write on a number add is going to, let say add is going to write on the register F 2 and multiplication is also going to write on a register F 2, add is issued after the multiplication add comes after multiplication because of long multiplication stage multiplication get over only here whereas, adding gets only will be done before that.

So, we are able to complete the adding operation before the multiplication operation is over that is exactly WAW hazard. When you have an instruction J and instruction let say instruction I is going to write into a register R instruction J is also going to write into a register R. If j happens after i then if j writes before I is going to write into R that is called WAW hazard. So, in a multi cycle floating point operation WAW hazard cannot happen that is false because WAW hazard will surely happen.

(Refer Slide Time: 31:11)



Tutorial Problem-4

❖ Which of the following statements is/are FALSE?

(I) For a MIPs multi-cycle floating point pipeline the initiation interval of FP-mul is larger than that of FP-add

(II) WAW hazard cannot happen in a MIPS multi-cycle floating point pipeline.

(III) In a MIPS multi-cycle floating point pipeline that supports operand forwarding, there will be 7 stalls between a pair of adjacent MUL instructions that has a RAW dependency between them.

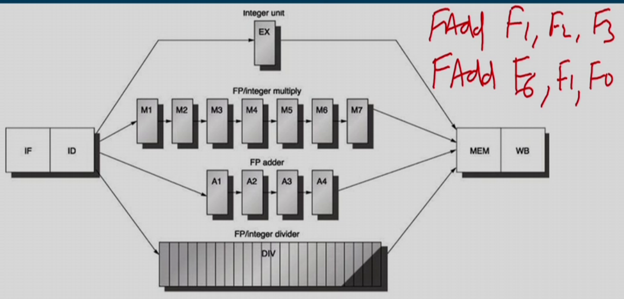
(IV) If a 32 bit value (0x12345678) is stored in memory byte addresses 2000, 2001, 2002 and 2003 in big-endian format, then location 2001 holds the value 0x56.

(A) III only (B) I only (C) I & IV only (D) I, II, III & IV

Now, the third statement in a MIPS multicycle floating point pipeline that supports operand forwarding there will be 7 stalls between a pair of adjacent multiplication instruction that has a raw dependency between them.

(Refer Slide Time: 31:33)

Multi-cycle Operations

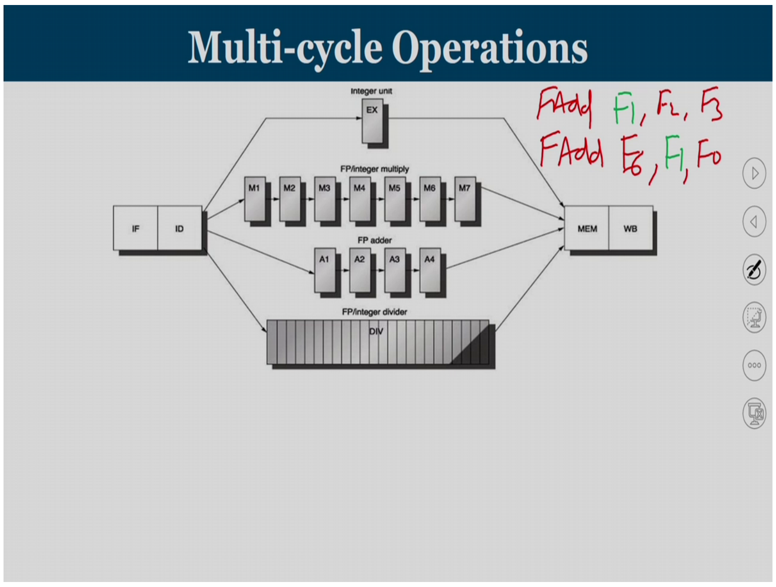


FAdd F1, F2, F3
FAdd F6, F1, F0

So, this is the floating point the pipeline. Now we are going to talk about a case where two adjacent adding operation let us say we are going to add two floating point numbers F 1, F 2 and F 3 are going to be add. So, the result is going to be obtained in F 1.

Similarly, there is one more add where there is a raw dependency; that means, F 1 is one of the operand and let say this is F 6. So, if you try to correct the value of F 1 with a for easy grasping.

(Refer Slide Time: 32:15)



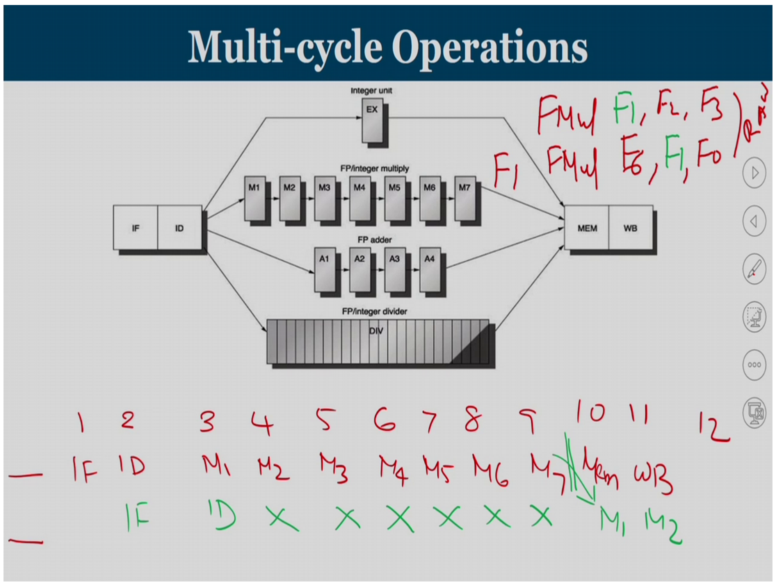
Multi-cycle Operations

```
FAdd F1, F2, F3
FAdd F6, F1, F0
```

So, the first instruction is going to write into F 1, second instruction is going to read from F 1, now if you look at in a MIPS multicycle floating point pipeline that is supports operand forwarding there will be 7 stalls between a pair of adjacent multiplication instruction that has a RAW dependency between them.

So, let me correct rather than this adding operation they are basically multiplication operation floating point multiplication operation. So, F MUL is the operation.

(Refer Slide Time: 32:41)



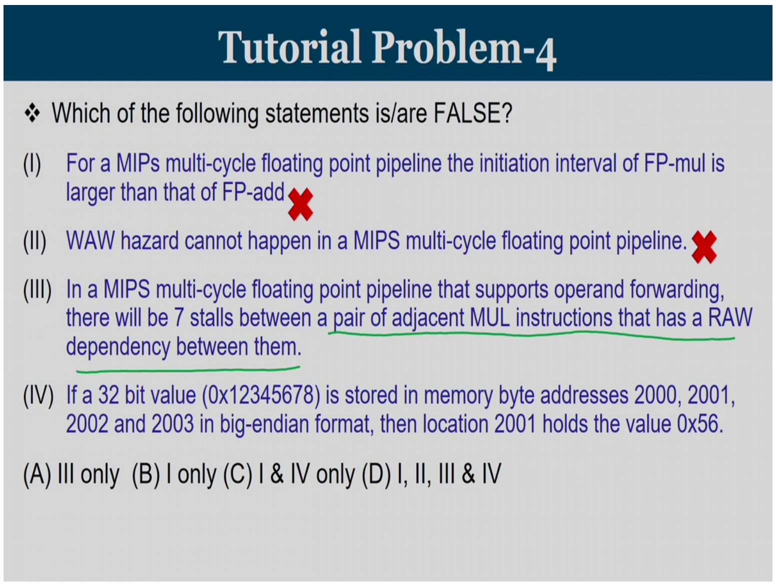
So, two multiplication between them there is a dependency, there is a RAW dependency read after write dependency. So, only if the first one produces the result when will I get produce a result F 1 will be written only at this point, now the value of that is to be used by the second instruction.

Now, if you try to re visit the instructions we are going to see what happens here. Let us take the first clock cycle, this is clock cycle 1,2,3,4,5,6, 7,8,9,10, 11 and you are in the instruction fetch state that is IF ID then it is m 1, m 2, m 3, m 4, m 6, m 7 then this is going to be your normal MEM stage and this is the write back. So, this is your first instruction, the second instruction is dependent on the first one.

So, when you go to the second instruction, we know that the second instructions, since it is dependent on the first one this is a second instruction IF ID. Now I cannot start the multiplication because the multiplication can be started only after getting the result and the result is obtained only at this point. So, M 1 can start only here even if operand forwarding is there. So, this is the point of M 1, M 2. So, how many stalls I have to put 1,2,3,4, 5, 6.

So, I am going to introduce only 6 stalls in this case. So, even if I have a dependency between them IF and ID lines will be completed like in the previous case. So, this is what we have seen.

(Refer Slide Time: 34:34)



Tutorial Problem-4

❖ Which of the following statements is/are FALSE?

(I) For a MIPs multi-cycle floating point pipeline the initiation interval of FP-mul is larger than that of FP-add ✖

(II) WAW hazard cannot happen in a MIPS multi-cycle floating point pipeline. ✖

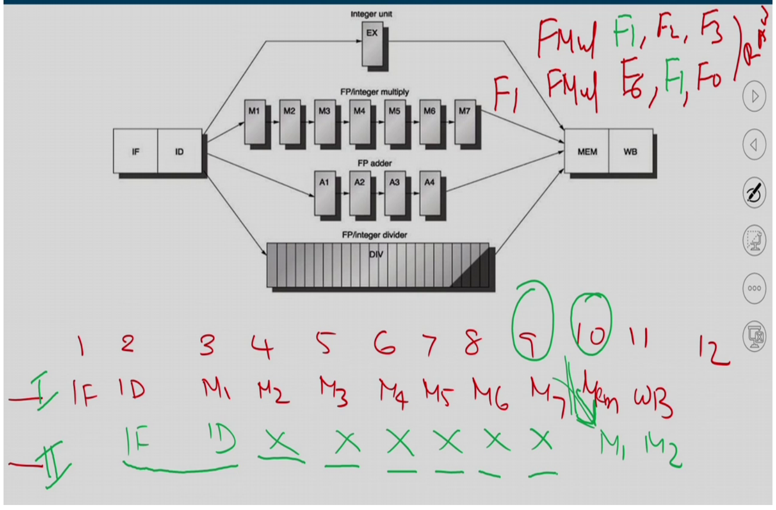
(III) In a MIPS multi-cycle floating point pipeline that supports operand forwarding, there will be 7 stalls between a pair of adjacent MUL instructions that has a RAW dependency between them.

(IV) If a 32 bit value (0x12345678) is stored in memory byte addresses 2000, 2001, 2002 and 2003 in big-endian format, then location 2001 holds the value 0x56.

(A) III only (B) I only (C) I & IV only (D) I, II, III & IV

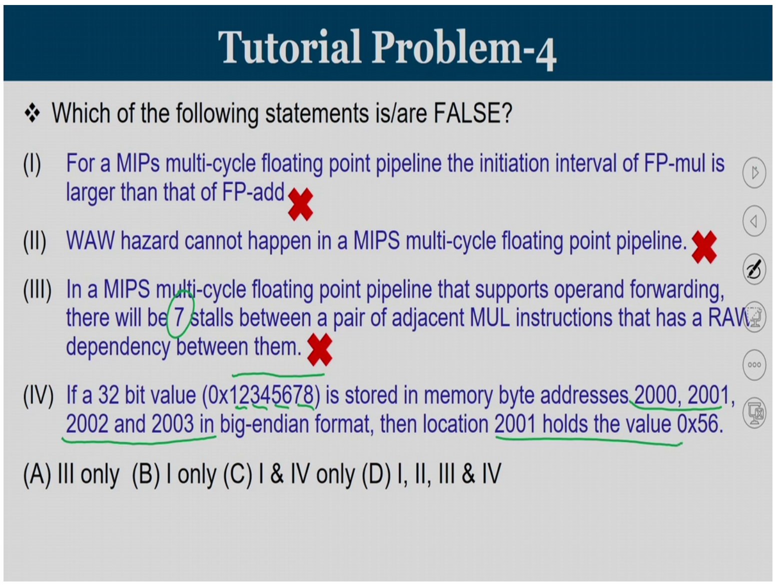
(Refer Slide Time: 34:38)

Multi-cycle Operations



IF and ID is going to be completed since I cannot start multiplication because this second line, this is line number 1 and line number 2, line number 2 has a dependency with the result that is produced by line number 1. So, the result is obtained only at the end of the ninth clock cycle, it is 7 stage of floating point multiplication operation. So, I can get the data only at this point only at the clock cycle 10 I can start my M 1. So, till then I have to include stall. So, stall is there at clock cycle 4, 5, 6, 7, 8, 9. So, only there are 6 stalls that are going to happen.

(Refer Slide Time: 35:15)



Tutorial Problem-4

❖ Which of the following statements is/are FALSE?

(I) For a MIPs multi-cycle floating point pipeline the initiation interval of FP-mul is larger than that of FP-add ✖

(II) WAW hazard cannot happen in a MIPS multi-cycle floating point pipeline. ✖

(III) In a MIPS multi-cycle floating point pipeline that supports operand forwarding, there will be 7 stalls between a pair of adjacent MUL instructions that has a RAW dependency between them. ✖

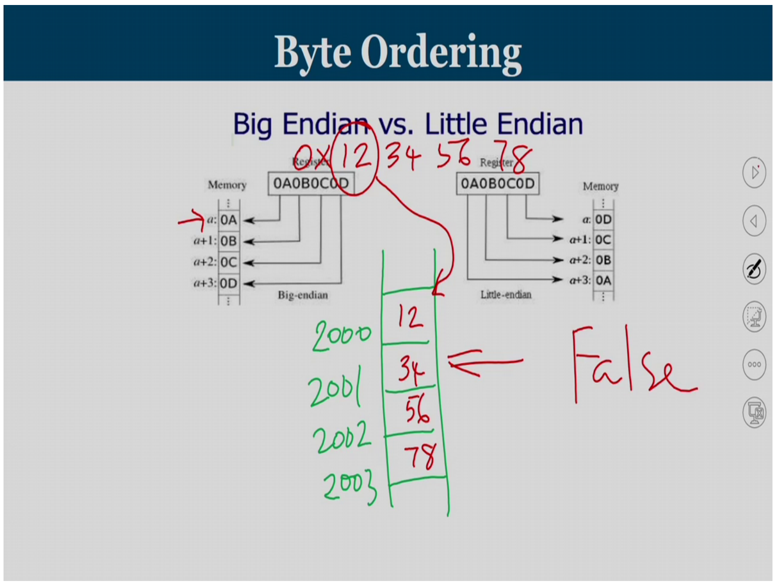
(IV) If a 32 bit value (0x12345678) is stored in memory byte addresses 2000, 2001, 2002 and 2003 in big-endian format, then location 2001 holds the value 0x56.

(A) III only (B) I only (C) I & IV only (D) I, II, III & IV

So, in the statement it is mentioned that there should be 7 stalls between them that is wrong. So, the third statement is also wrong. Now, we go to the forth statement if 32 bit value which is represented in hexadecimal zero x 1 2 3 4 5 6 7 8 is stored in memory byte addresses. So, there are the byte addresses in which we are going to store this value. So, each of this memory location 2000, 2001, 2002 and 2003 can store 1 byte of information. So, totally I have 4 bytes of information where two of this hexadecimal number will represent a byte. So, 4 byte value is going to be stored in 4 adjacent memory locations in a Big Endian format. Then location 2001 holds the value 56.

(Refer Slide Time: 36:05)

Byte Ordering

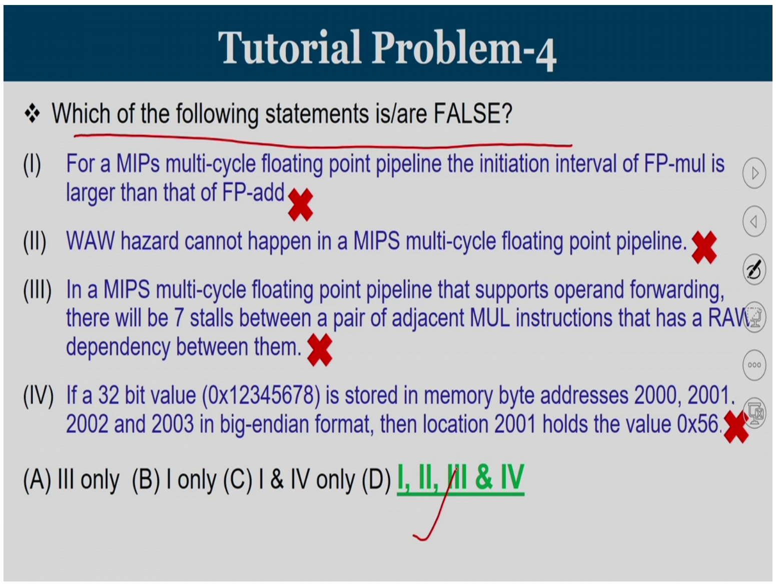


Let see whether it is true or false will try to illustrate the concept of Big Endian and Little Endian. So, in little Endian format let see this is the number what you are going to store from a register. So, these are the 4 bytes the least significant byte is going to be stored in the memory address a then the more significant address will be stored in a plus 3. So, the least significant byte is stored with the lower address, more significant byte is stored in the higher address. When you come to Big Endian format the more significant byte is stored in the lower address.

So, in this case of an example we have memory location 2000, 2001, 2002 and 2003, these are the 4 memory locations that we have mentioned and we have the value that is the represented by zero x 1 2 3 4 5 6 7 8 that is the number what we are going to operate.

So, it is zero x 1 2 3 4 5 6 7 8. Now it is in which format what we are trying to understand it is in Big Endian format. So, in Big Endian format the most significant byte is going to be stored at the lower address A. So, in this case 1 2 to get stored here 3 4 get stored here 5 6 will be storing it here and 7 8 is stored here. Now the question is, asking whether location 2001s holds 56? So, 2001 is holding 34. So, the statement 2001 holding 0 x 5 6 is faults.

(Refer Slide Time: 38:09)



Tutorial Problem-4

❖ Which of the following statements is/are FALSE?

(I) For a MIPs multi-cycle floating point pipeline the initiation interval of FP-mul is larger than that of FP-add ✗

(II) WAW hazard cannot happen in a MIPS multi-cycle floating point pipeline. ✗

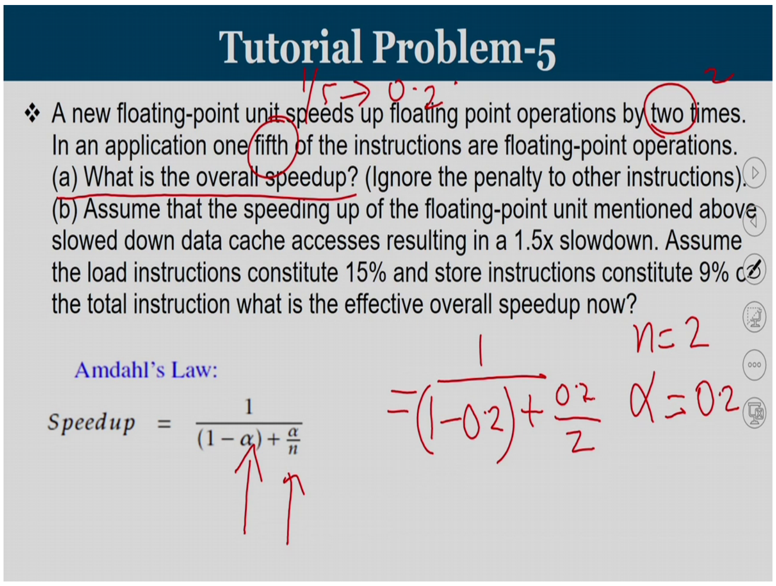
(III) In a MIPS multi-cycle floating point pipeline that supports operand forwarding, there will be 7 stalls between a pair of adjacent MUL instructions that has a RAW dependency between them. ✗

(IV) If a 32 bit value (0x12345678) is stored in memory byte addresses 2000, 2001, 2002 and 2003 in big-endian format, then location 2001 holds the value 0x56. ✗

(A) III only (B) I only (C) I & IV only [✓ (D) I, II, III & IV]

So, the last statement this also faults. So, the question was which of the following statement this faults? All the four statements1, 2, 3 and 4 are faults. So, that completes our problem number 4.

(Refer Slide Time: 38:21)



Tutorial Problem-5

- A new floating-point unit speeds up floating point operations by two times. In an application one fifth of the instructions are floating-point operations. (a) What is the overall speedup? (Ignore the penalty to other instructions). (b) Assume that the speeding up of the floating-point unit mentioned above slowed down data cache accesses resulting in a 1.5x slowdown. Assume the load instructions constitute 15% and store instructions constitute 9% of the total instruction what is the effective overall speedup now?

Amdahl's Law:

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

$1/5 \rightarrow 0.2$

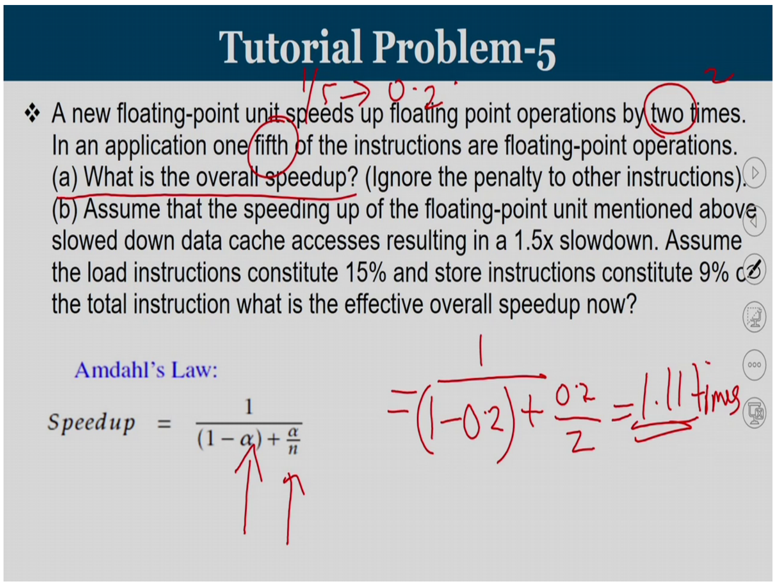
$= \frac{1}{(1-0.2)+\frac{0.2}{2}}$ $n=2$ $\alpha = 0.2$

Now we are moving into the fifth problem, this problem has two sub divisions, a new floating point units speeds up floating point operations by two times in an application one fifth of the instructions are floating point operation what is the overall speed up. So, we have to understand that your floating point unit is going to improve your performance by 2 times.

So, every floating point operation gets is speeding of two factor 2 and there are one fifth of the instructions that are floating point; that means, 0.2 percent was 0.2 is the fraction of the instructions that are to be used by the floating point unit what is a overall speed up, this is governed by Amdahl's law which tells that the speed up is defined as 1 by 1 minus alpha plus alpha by n where alpha is the fraction enhanced and n is the speed up of enhancement for that fraction.

Let us try to define the number. So, here n is equal to 2,value of alpha is the percentage of fraction that gets improvement. So, only floating point instructions are getting the benefit. So, 0.2 is the fraction. So, once we substitute it 1 by 1 minus 0.2 plus 0.2 divided by 2. So, when we solved is we are going to get the answer as 1.11.

(Refer Slide Time: 40:07)



Tutorial Problem-5

- A new floating-point unit speeds up floating point operations by two times. In an application one fifth of the instructions are floating-point operations. (a) What is the overall speedup? (Ignore the penalty to other instructions). (b) Assume that the speeding up of the floating-point unit mentioned above slowed down data cache accesses resulting in a 1.5x slowdown. Assume the load instructions constitute 15% and store instructions constitute 9% of the total instruction what is the effective overall speedup now?

Amdahl's Law:

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

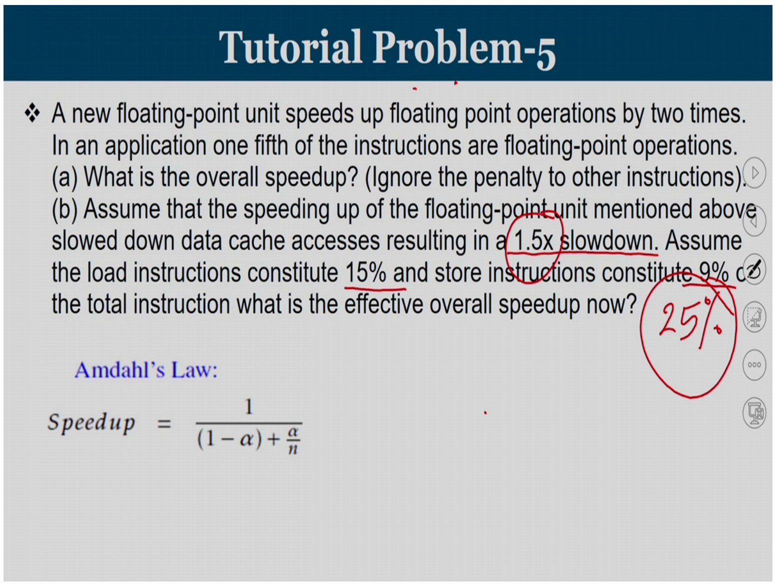
$1/5 \rightarrow 0.2$

$$= \frac{1}{(1-0.2)+\frac{0.2}{2}} = \underline{1.11 \text{ times}}$$

So, when you use a floating point unit that is going to improve the performance by two times for floating point instructions, let us say the floating point instruction is only 1 by 5th of the total instruction then what is a overall speed up, the overall speed up is 1.11 times.

Now, the question has a second component. So, we will try to revisit the second component what are we going to get from this. So, second component is an extension to the first problem. So, whatever applicable on the first problem that should be considered in the second problem also. Assume that the speeding up of floating point unit mentioned above slowed down data cache access resulting in 1.5 times slowdown.

(Refer Slide Time: 41:03)



Tutorial Problem-5

- A new floating-point unit speeds up floating point operations by two times. In an application one fifth of the instructions are floating-point operations. (a) What is the overall speedup? (Ignore the penalty to other instructions). (b) Assume that the speeding up of the floating-point unit mentioned above slowed down data cache accesses resulting in a $\boxed{1.5\text{x}}$ slowdown. Assume the load instructions constitute 15% and store instructions constitute 9% of the total instruction what is the effective overall speedup now? $\boxed{25\%}$

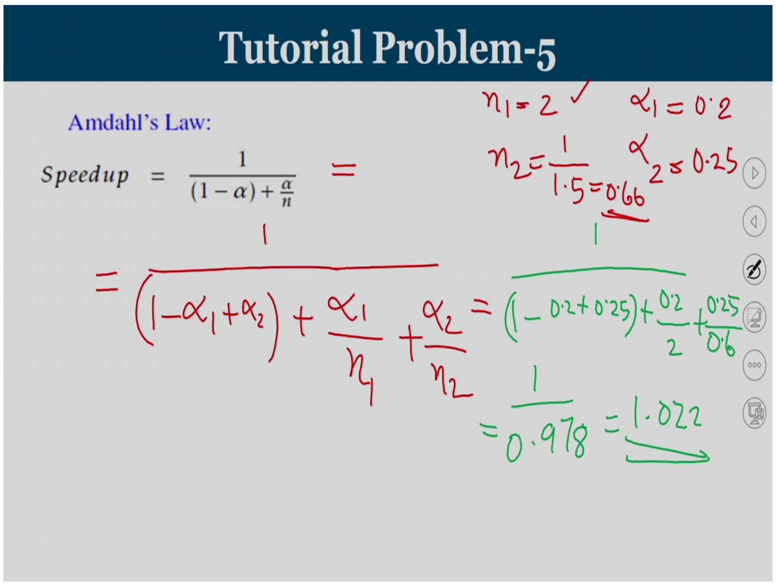
Amdahl's Law:

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

So, your data cache is going to be slowed down by 1.5 times. Assume the load instructions constitute 15 percent and store instructions constitute 9 percent of the total instruction.

So, when you go to data cache whenever there is a Load instruction whenever there is a Store instruction we are going to go to data cache, so 25 percent or data cache instructions. These 25 percent instructions are getting slowdown by 1.5 times. So, what is the overall speed up that we are going to get.

(Refer Slide Time: 41:39)



Tutorial Problem-5

Amdahl's Law:

$n_1 = 2$ ✓ $\quad \alpha_1 = 0.2$

$n_2 = \frac{1}{1.5} = \underline{0.66} \quad \alpha_2 = 0.25$

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

$$= \frac{1}{(1-\alpha_1+\alpha_2)+\frac{\alpha_1}{n_1}+\frac{\alpha_2}{n_2}} = \frac{1}{(1-0.2+0.25)+\frac{0.2}{2}+\frac{0.25}{0.6}}$$

$$= \frac{1}{0.978} = \underline{1.022}$$

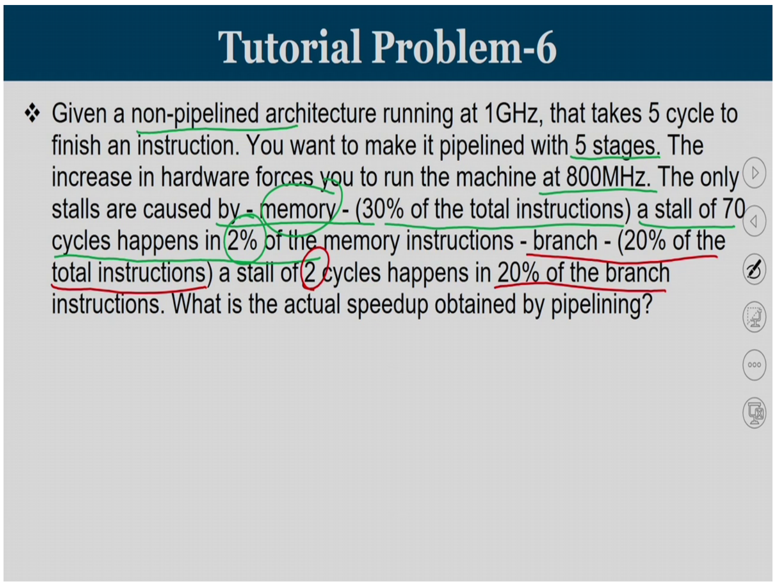
So, we know that we have to slightly rephrase because there are two factors that is coming, one is for floating point operations and one is for data cache instructions. So, floating point operations are 1 by 5th and data cache instructions are 25 percent and if you look at the value of n 1 you get the speed up of 2 and n 2 it is 1 by 1.5, because it is getting slowed down by 1.5 times.

So, the previous example we have seen that floating point instructions are speeded up by 2 whereas, data cache access is slowed down by 1.5,slowdown by 1.5 times means it getting a speed up of 1 by 1.5,speeding up and slowdown are this reciprocal to each other. So, the speed up that you are going to get 0.66. Now because we have two elements that is coming 1 by 1 minus alpha 1 plus alpha 2, these are the percentage fraction of instructions that are impacted and alpha 1 time that is a percentage improvement is n 1 and alpha 2 times the percentage improvement is n 2.

So, if you substitute the value 1 by 1 minus 0.2 plus 0.25, alpha 1 value is 0.2 divided by 2 plus 0.25 divided by 0.66. Once you solve this you are going to get 1 by 0. 978; that means, a speed up I am going to get is 0. 022. So, this is a classical problem where due to a change in one of the unit you are getting certain instructions getting speeded up.

But incorporating that unit how slowdown some other component. So, in this example the floating unit that we newly added is improving the performance of floating point instruction by two times whereas, it is making your data cache work a bit slow. So, data cache is going to be slowed down by1.5 times. So, what is overall speed up that you get in that way if you substitute the values properly in Amdahl’s law you are going to get this solution.

(Refer Slide Time: 44:23)



Tutorial Problem-6

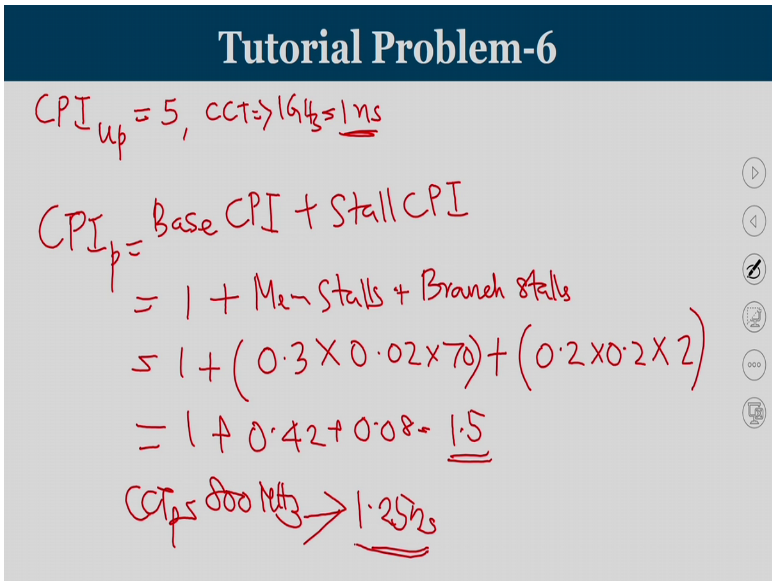
❖ Given a non-pipelined architecture running at 1GHz, that takes 5 cycle to finish an instruction. You want to make it pipelined with 5 stages. The increase in hardware forces you to run the machine at 800MHz. The only stalls are caused by - memory - (30% of the total instructions) a stall of 70 cycles happens in 2% of the memory instructions - branch - (20% of the total instructions) a stall of 2 cycles happens in 20% of the branch instructions. What is the actual speedup obtained by pipelining?

Now, we move into problem number 6. So, in problem number 6 you are given the comparison between a non pipelined architecture as well as a pipelined architecture. So, given a non pipelined architecture running at 1 Gigahertz that takes 5 cycle to finish an instruction you want to make it pipelined with 5 stages the increase in hardware forces you to run the machine at 800 Megahertz.

So, the only stalls are caused by, so you will get a memory stall for what 30 percent of total instructions are memory and you get 70 cycles stall for 2 percent of total memory instructions.

Similarly, for branch conditions which are roughly 20 percent of total instructions are branch and out of the branch 20 percent of the branch instruction you incur a two cycle stall. So, what is the speed up that you are going to get. So, we have to understand that in the case of an unpipelined design. So, for unpipelined design of CPI cycles per instruction for unpipeline.

(Refer Slide Time: 45:45)



Tutorial Problem-6

$CPI_{up} = 5$, $CCT \Rightarrow 1GHz = \underline{1ns}$

$CPI_p =$ Base CPI + Stall CPI

$= 1 +$ Mem Stalls + Branch Stalls

$= 1 + (0.3 \times 0.02 \times 70) + (0.2 \times 0.2 \times 2)$

$= 1 + 0.42 + 0.08 = \underline{1.5}$

$CCT_p = 800\ MHz \rightarrow \underline{1.25ns}$

So, we can put it UPS, UP is equal to 5 because a non pipelined architecture is going to take 5 cycles to finish an instruction. So, you take 5 clock cycles per instruction.

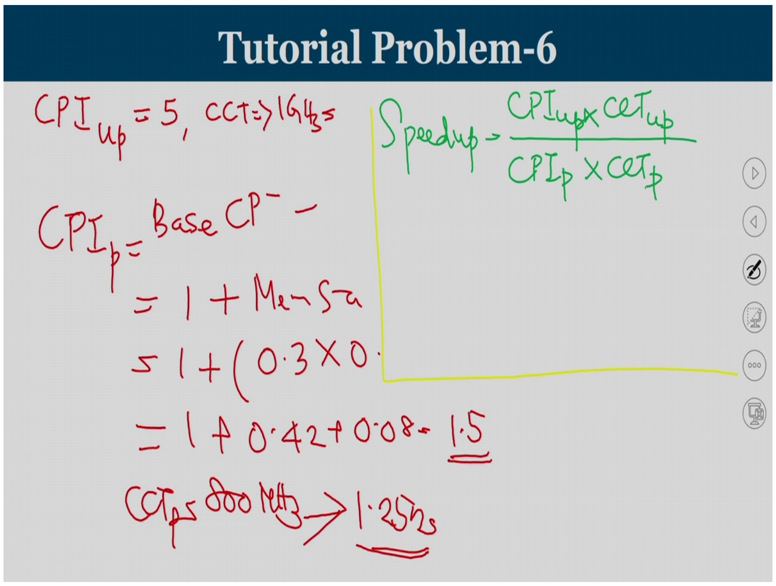
So, CPI of that is 1 and clock cycle time is equal to the 1 Gigahertz is going to 1 as 1 nanosecond. Now when you go to the pipelined architecture our basic assumption is CPI is going to be 1. So, in the case of a pipelined architecture for every cycle one more instruction is getting over in our pipeline. So, the CPI of the pipelined architecture is base CPI; that means, every instruction if there is now dependency or a now stalls between them that is called base CPI called ideal CPI plus stall CPI.

You are going to stall for certain instruction. Base CPI is always 1 because every pipeline assume that the clock cycles per instruction is 1 and now we have stalls here they are memory stalls as well as branch stalls. Now memory stalls are happening for the memory instructions alone and it is mentioned that you have 30 percent of your total instructions are memory instructions, out of that 2 percent of the memory instruction only will result in a stall. So, typically this memory instruction stall happen whenever it is a cache MIS.

So, 30 percent of the instructions are memory instructions and out of the memory instruction 2 percent of them will stall for how many cycles 70 cycles. Now branch, 20 percent of them are branch instruction that is what is being mentioned and out of the branch instructions only 20 percent of them will result in stall and the stall we are going to get is only 2. So, this we will solve how to 0.42 plus 0.08 the value is 1.5. So, the CPI under the pipelined version is 1.5.Now let us try to understand what is the clock cycle time of the pipelined version? It is going to work under 800 Megahertz. So, that 800 Megahertz corresponds to 1.25 nanosecond.

So, we have obtained what is the CPI value in the case of an unpipelined design and what is the CPI value in the case of a pipelined design? Now, while solving this what is the speed up, the overall speed up we are going to get.

(Refer Slide Time: 49:15)



Tutorial Problem-6

$CPI_{up} = 5$, $CCT \Rightarrow 1GHz$

$CPI_p = $ Base CP‾ –

$= 1 +$ Mem Sta

$= 1 + (0.3 \times 0.$

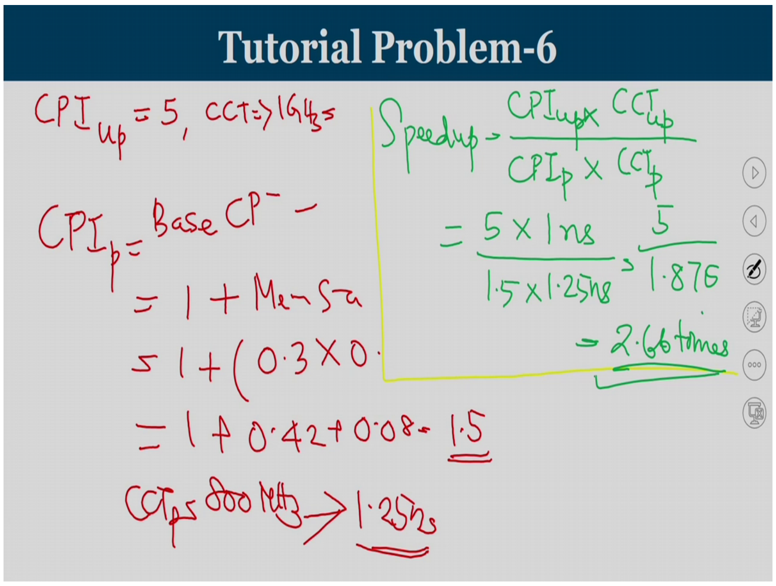
$= 1 + 0.42 + 0.08 = \underline{\underline{1.5}}$

$CCT_p = 800\ MHz \rightarrow \underline{\underline{1.25ns}}$

$$\text{Speedup} = \frac{CPI_{up} \times CCT_{up}}{CPI_p \times CCT_p}$$

So, speed up can be defined as CPI of the unpipelined into clock cycle time of unpipelined divided by CPI of the pipelined processor into clock cycle time of the pipelined. So, if the clock cycle time of the pipeline is going to be larger because for implementing a pipeline you have to add the interface registers.

(Refer Slide Time: 49:47)



Tutorial Problem-6

$CPI_{up} = 5$, $CCT \Rightarrow 1GHz =$

$CPI_p = \text{Base } CP^{-} -$

$= 1 + \text{Mem Sta}$

$= 1 + (0.3 \times 0.$

$= 1 + 0.42 + 0.08 = \underline{1.5}$

$CCT_p = 800\text{ MHz} \rightarrow \underline{1.25ns}$

$$\text{Speedup} = \frac{CPI_{up} \times CCT_{up}}{CPI_p \times CCT_p} = \frac{5 \times 1ns}{1.5 \times 1.25ns} = \frac{5}{1.875} = \underline{2.66 \text{ times}}$$

So, the CPI of unpipelined is equal to 5 and clock cycle time is 1 nanosecond divided by CPI of the pipelined is 1.5 into clock cycle time is 1.2 nanosecond and you are able to get 5 divided by 1.875. So, the overall speedup is 2.66 times. So, when you make this 5 stage pipeline which takes 5 clock cycle to complete an instruction which is operating at 1 Gigahertz when you make into pipelined version it is going to improve your performance by 2.66 times including the overhead for the pipeline.

So, in this way we are trying to understand what happens in a pipeline what is impact of stalls. So, on that completes today's the tutorial lecture. So, I hope this session was useful for you to understand then grasp about the subject. So, I request you to go through similar problems; that is there at the end of the textbook, the prescribed reference text book. We will have more such tutorials which we will give you more grasp in solving the questions.

Thank you.