

**Advanced Computer Architecture**  
**Prof. Dr. John Jose**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Guwahati, Assam.**

**Tutorial2**  
**Pipeline Hazard Analysis**

Welcome to the tutorial session of the second week. In tutorial 2, we are focusing your attention on understanding pipeline hazards. Let us move into the first question. It is basically a set of statements we have to tell whether it is true or false. Which of the following statements is or true RAW data hazard could be reduced to by operand forwarding. So that is the first 1.


**(Refer Slide Time: 00:35)**

**True/False**

❖ 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.

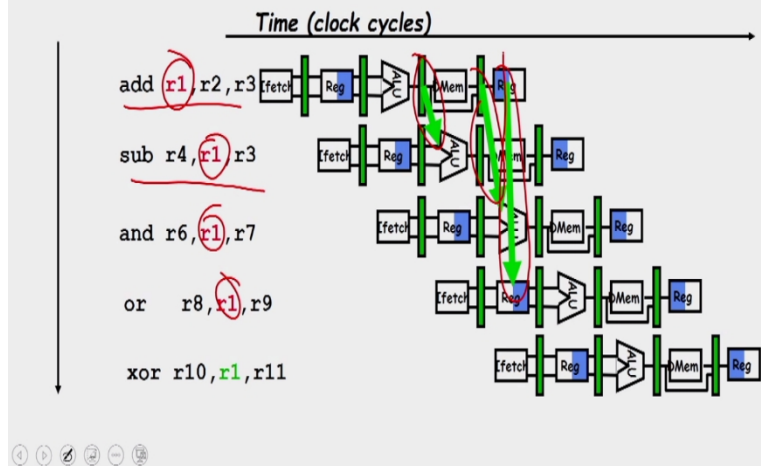
(A) I only (B) I & III only (C) II & IV only (D) III & IV only



So, consider the set of the instructions that has been given here, this ADD instruction then sub and we know that the add produces a result which has to be written in r1. And all other subsequent instructions are going to read from r1. So there exists they RAW dependency and we can see that by the concept of operand forwarding, we are forwarding the result from one functional unit to another that is from output of ALU to input of ALU and from output of mem stage to the input of ALU.

**(Refer Slide Time: 00:50)**

## Operand Forwarding to Avoid Data Hazard



So because of this, I am not having any stall. So, in the statement RAW data hazard could be reduced by operand forwarding is true the second 1 is a normal in order five stage MIPS pipeline can achieve an IPC larger than one. So this is how we visualize a pipeline in every cycle, I am going to fetch and instruction and in every cycling ideal case, we are going to complete 1 instruction.

(Refer Slide Time: 01:27)

## True/False

❖ 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

(Refer Slide Time: 01:40)

## Visualizing Pipelining

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

So on an average if you look at every cycle, I am completing 1 instruction. So the statement and normally in-order 5 stage MIPS pipeline can have an IPC instructions per cycle larger than 1 I cannot complete more than 1 instruction per cycle in a normal in order 5 stage MIPS pipeline. So on a IPC larger than 1 is impossible. So, the second statement is false.

(Refer Slide Time: 01:55)

## True/False

❖ 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

Moving on to the third statement for a MIPS Instruction

store R2, 16(R3)

some content stored in its ID/EX pipeline register will bypass the EX unit directly to the EX/MEM pipeline register. So, let us try to understand what is the store instruction, the contents in the register R2 has to be stored into memory and the address to which the storage happen is

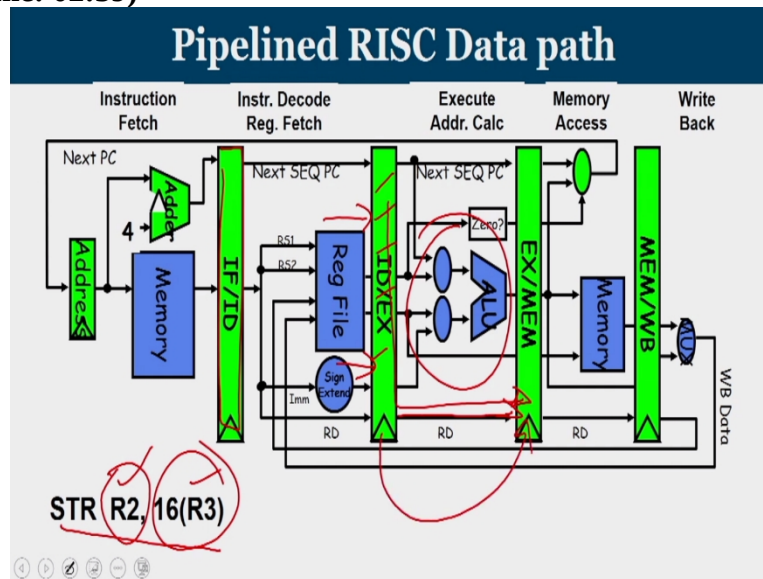
16 + content of R3

let us say how this happens to

store R2 16(R3)

so, your instruction is being fetched.

(Refer Slide Time: 02:39)



So, at the end of the fetching stage your instruction is available here, you fetch the contents of R2 and R3. So R2 and R3 is going into ID/EX register. Now 16 and R3 is added in the ALU to get effective address in the meantime, the contents of R2 which is already available in the ID/EX register, that moves to EX/MEM register.

(Refer Slide Time: 03:29)

### True/False

❖ 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, all the contents that is available in the ID EX register are not used by the ALU through this line, some of the content basically the content of R2 that is the value to be stored in the memory, which is already locating in R2, will move from ID/EX register to the EX/mem register. So, the statement for a MIPS Instruction

store R2, 16(R3)

Some contents is stored in its ID EX pipeline register will bypass the EX directly to the EX mem register is true the content of R2.

The fourth one is a normal 5 stage in order RISC pipeline without operand forwarding can have RAW and WAR hazards See, this is the general pipeline that you have seen and this is RAW hazard and here we have the WAR hazards.

(Refer Slide Time: 03:53)

Visualizing Pipelining								
Instruction number	Clock number							
	1	2	3	4	5	6	7	8
<i>i</i>	IF	ID	EX	MEM	WB			
<i>i+1</i>		IF	ID	EX	MEM	WB		
<i>i+2</i>			IF	ID	EX	MEM	WB	
<i>i+3</i>				IF	ID	EX	MEM	WB
<i>i+4</i>					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

(Refer Slide Time: 04:07)

### True/False

❖ 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 the last statement a normal 5 stage in order RISC pipeline without operand forwarding can have RAW and WAR hazards that is false. So, here which of the following statements are true the answer is 1 and 3. So now, we will look into the statements like this the next set of true or false statement,

(Refer Slide Time: 04:27)

## True/False

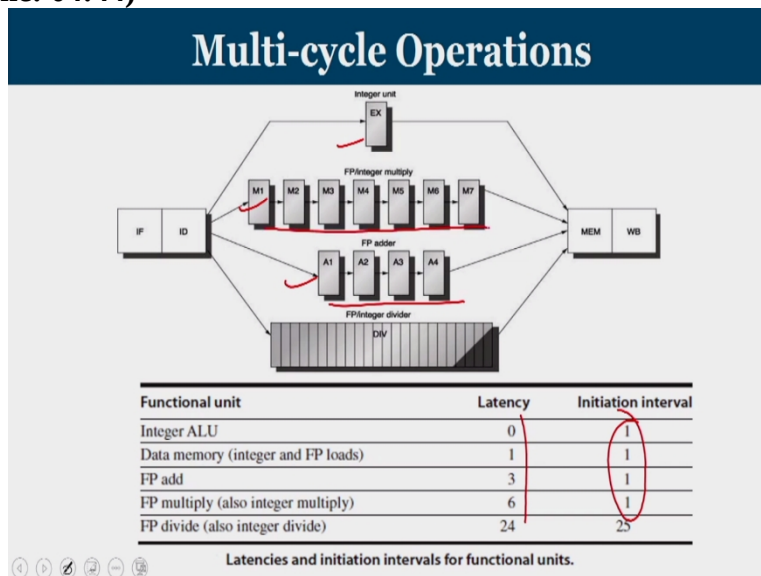
❖ Which of the following statements is/are FALSE?

- (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

Which of the following statement is/are false. For a MIPS multi cycle floating point pipeline, the initiation interval of floating point mul is larger than that are floating point add, let us see what is there. So, the initiation interval is being defined us how much cycle has to be elapsed if you wanted to use the functional unit again.

(Refer Slide Time: 04:44)



So, the integer ALU the floating point multiplier, and the floating point adder has an initiation interval of 1, but they differ in latency there are 4 stages in the FP adder and there are 7 stages in the integer or floating point multiplier. So, the statement for a MIPS multi cycle floating point pipeline the initiation interval of floating point mul is larger than that a floating point add is wrong they both have the same initiation interval, but however, a FP-mul has a larger latency than an FP-add.

(Refer Slide Time: 05:12)



## True/False

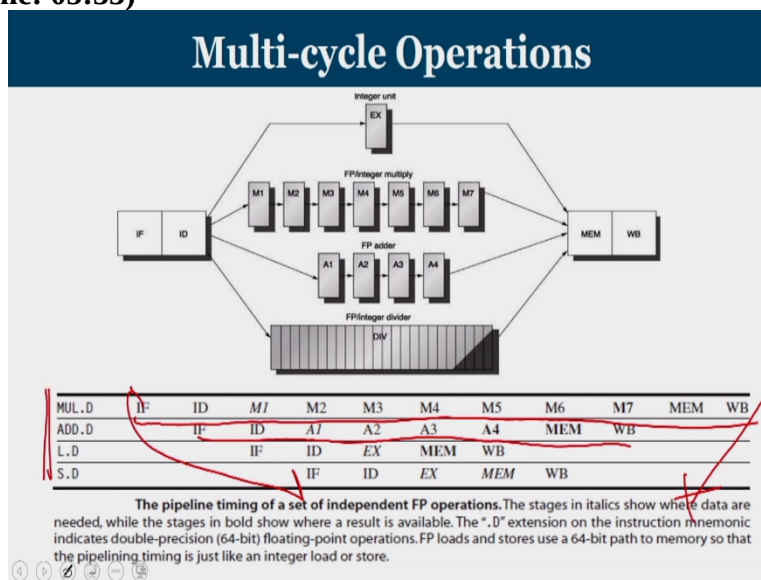
❖ Which of the following statements is/are FALSE?

- (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, WAW hazard cannot happen in a MIPS multi cycle floating point pipeline. So, WAW hazard is write after write hazards. Consider the case that when you have a MIPS multi cycle floating point pipeline, where some instructions are longer the first instruction is a multiplication. So, it takes 11 cycles to complete let us say the second one is add which will take only 8 cycles to complete. So, surely, even though we are fetching instruction in order the execution is out of order or the completion is out of order.

(Refer Slide Time: 05:33)



(Refer Slide Time: 06:08)

## True/False

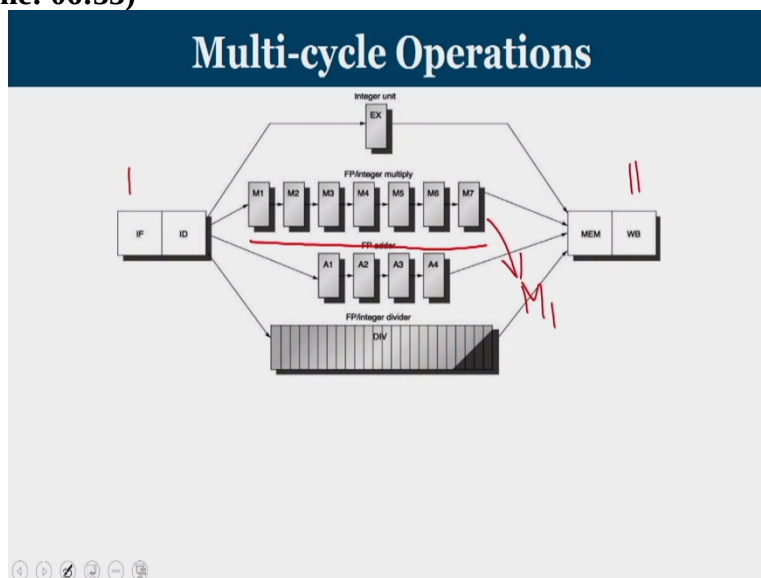
❖ Which of the following statements is/are FALSE?

- (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, when the completion is out of order, there is always a possibility that we can have write after write hazards. So, the statement in a MIPS a WAW hazards cannot happen is false WAW hazards will happen because instructions are completed out of order.

Moving on to the third one in a MIPS multi cycle floating point pipeline that supports operand forwarding there will be 7 stalls between a pair of adjacent multiplication instructions, that has a raw dependency between them.

(Refer Slide Time: 06:33)



So, this is what these the general structure of this multiplication unit, you have the first instruction which starts at 1 and then it completes only at 11. If the second instruction is going to have a RAW dependency on it, then the M1 of the second instruction can happen only after M7 of the previous instruction. So, there will not be 7 stalls, there will be only 6 stalls between them.

(Refer Slide Time: 07:03)



## True/False

❖ 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.

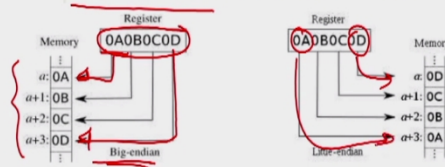
(A) III only (B) I only (C) I & IV only (D) I, II, III & IV

So, the statement that there are 7 stalls between a pair of our adjacent multiplication instruction that is a RAW dependency is wrong.

(Refer Slide Time: 07:34)

## Byte Ordering

### Big Endian vs. Little Endian



If a 32 bit values 0 x 12345678 is stored in memory byte addresses 2000, 2001, 2002 and 2003 in big endian format, then the location 2001 holds the value 56. Let us try to understand what you mean by Big Endian Little Endian in Big Endian when you store a 4 bite data into 4 continuous memory location, If MSB is stored in the lower address and LSB stored in the higher address, then that is known as big endian. If the LSB is stored in lower address and MSB stored in the higher address that is called Little endian. So the statement when you have a 32 bit value 12345678 is going to be stored in locations 2000. So these are the locations 2000, 2001, 2002 and 2003. So this is 2000, 2001, 2002 and 2003. And our value is 0 x 12345678.

(Refer Slide Time: 08:03)

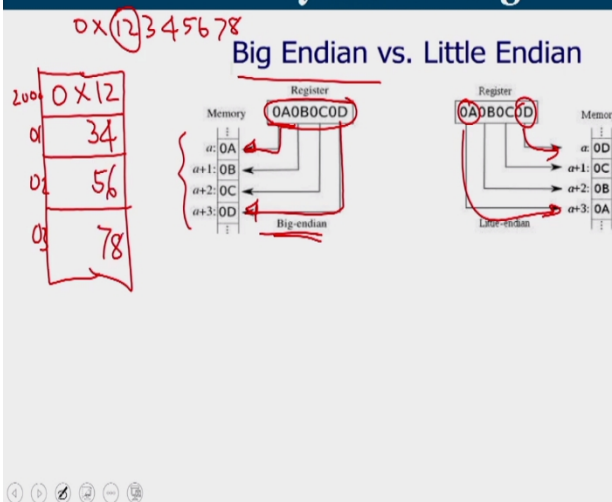
## True/False

❖ 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: 08:46)

## Byte Ordering



Then we are going to store them in big endian format. So once you store and big endian format, that MSB is stored in the lower address. So 0x12 gets stored here, 3, 4 gets stored here, 5, 6 gets stored here and 7,8 will get stored here if it is in big endian format. So what they are telling is 2001 holds 56. So 2001 is holding 34. So the statement that 2001 holds 56 is wrong. So all the 4 statements are wrong.

(Refer Slide Time: 09:12)

## True/False

- ❖ 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 answer would be 123 and 4 they all are false. So, in the first 2 questions in today is tutorial, there are 4 statements given and we were ask to check whether those statements are true or false. And we have no seen by correlating with the theory concept that we learned in the lectures to ascertain whether these statements are true or false. So to answer these kinds of questions, you require a thorough reading of the material, the video that has been covered and supplementary reading of the textbook material in which these topics are discussed.

(Refer Slide Time: 09:51)

## Pipeline Hazards

Given a non-pipelined architecture running at 1.5 GHz, that takes 5 cycles to finish an instruction. You want to make it pipelined with 5 stages. Due to hardware overhead the pipelined design will operate only at 1 GHz. 5% of memory instructions cause a stall of 50 cycles, 30% of branch instruction cause a stall of 2 cycles and load-ALU combinations cause a stall of 1 cycle. Assume that in a given program, there exist 20% of branch instructions and 30% of memory instructions. 10% of instructions are load-ALU combinations. What is the speedup of pipelined design over the non-pipelined design?

Let us know move into pipeline hazards. Let us try to understand this question given a non-pipeline architecture running at 1.5 gigahertz which will take 5 cycles to finish an instruction you want to make it pipeline with 5 stages. When I am having pipeline interface register due

to the hardware overhead, the pipeline design will operate only at 1 gigahertz. So, you have to understand your non pipelined design will operate at 1.5 gigahertz, so the

$$\text{clock} = (1 / 1.5) \text{ gigahertz}$$

whereas the pipeline design is operating at one gigahertz.

Now 5% of the memory instruction cause a stall of 50 cycles 30 percent of branch instruction cause a stall of 2 cycles and load ALU combinations we have seen that if there is an ALU operation immediately after the load operation, they cause a stall of one cycle assume that in a given program, their access 20% of branch instructions and 30% of memory instructions 10% of instructions are load-ALU combinations, what is the speedup of pipeline to design over the non-pipelined design and it is a question that has been given.

So, let us try to understand what the question is all about. You have an un-pipelined design where in the timing is given 1.5 gigahertz of clock we have a pipelined design which can operate only at 1 gigahertz but idea of pipeline is the base CPI is 1. Every cycle 1-1 instruction is getting completed in the case of a unpipelined version and it takes five cycles to complete an instruction.

Now in the pipeline, we will be having hazard, the first kind of hazard is called the memory issue where in you are trying to fetch something from an the instruction from memory are not getting. So, for certain percentage of memory instructions, it will be hit in the memory for certain percentage it may be miss in the memory. So, whenever you miss you are going to have 50 cycles of stall. Similarly, when you come to branches, there are also there are hazards for certain fraction of branches.

And there is a data hazard load-ALU combination that is for certain percentage. Now, we have to find out what is the overall execution time in this case. So, the CPI of unpipelined is 5. So it takes 5 cycles to complete an instruction the unpipelined version is working at 1.5 gigahertz roughly 0.667 nanosecond and the pipeline the version is running into one nanosecond time. The execution time of the unpipelined is CPI in the clock cycle the execution time for 1 instruction.

**(Refer Slide Time: 12:15)**

## Pipeline Hazards

(a)  $CPI_{up} = 5$

$1.5\text{GHz} \rightarrow 0.67\text{ ns} : 1\text{ GHz} \rightarrow 1\text{ ns}$

$$Ex_{T\ up} = CPI \times CCT = 5 \times 0.66\text{ ns} = 3.33\text{ ns / instruction}$$

(b) Effective  $CPI_p = \text{Base CPI} + \text{stall CPI}$

{stalls= Memory stalls+ Branch stalls+ Load -ALU stalls}

$$= 1 + (0.3 \times 0.05 \times 50) + (0.2 \times 0.3 \times 2) + (0.1 \times 1) = 1 + 0.75 + 0.12 + 0.1 = 1.97$$

$$Ex_p = CPI \times CCT = 1.97 \times 1\text{ ns} = 1.97\text{ ns / instruction}$$

$$\text{Speedup} = Ex_{up} / Ex_p = 3.33 / 1.97 = 1.69$$

So, it takes 5 cycles to complete 1 instruction and each cycle is 0.6667 nanoseconds. So roughly it is 3.33 nanosecond it will take to complete one instruction in the unpipelined design. So, the

$$\text{effective CPI (pipelined instruction)} = \text{base CPI}$$

that is one in every cycle be complete 1 instruction + whatever the stall that is happening. Now what are the kinds of stall we have memory stalls, we have branch stalls and we have Load-ALU stalls

The memory stalls what is the base CPI. Base CPI is 1 that is every cycle 1-1 instruction is getting over memory, you have 30% of instruction that are memory but 5% of memory instruction will incur 50 cycle stalls. So it is

$$0.3 * 0.05 * 50$$

.Coming into branch we have 20% of the instruction that are branch but 30% of the branch instruction only will create hazards it will create two cycle of stalls coming into load-ALU, 10% of load-ALU instruction combinations are there and they are going to create one cycle start.

So it is

$$1 + 0.75 + 0.12 + 0.1 = 1.97 \text{ (roughly)}$$

is the effective CPI in the case of pipelined design. So execution time in the case of pipelined CPI. In the clock cycle time CPI is 1.97. And it is going to operate the pipelined version is

going to operate at one nanosecond clock, so its 1.97 nanoseconds per instruction, so what is the speed up. Speed up is execution time of unpipelined, divided by

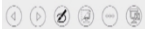
$$\text{execution time of pipeline} = 3.33/1.97=1.69.$$

The pipeline hazards next question, a program has 2000 instruction in the sequence, Load add, Load add like that. So the first, third, fifth, seventh, all odd number, instructions are load, odd and all even number instruction are add the add instruction depends on the Load instruction right before it, the Load instruction depends on the add instruction right before it. If the program is executed on a 5 stage pipeline, what would be the actual CPI with and without operand forwarding.

**(Refer Slide Time: 14:24)**

**Pipeline Hazards**

A program has 2000 instructions in the sequence L.D, ADD.D, L.D, ADD.D,..... L.D, ADD.D. The ADD.D instruction depends on the L.D instruction right before it. The L.D instruction depends on the ADD.D instruction right before it. If the program is executed on the 5-stage pipeline what would be the actual CPI with and without operand forwarding technique?



So, without operand forwarding technically let us try to understand how the instruction will flow. So this is the sequence of instruction you have Load add, Load add like that, the peculiarities is this add is dependent on the load. So the result of load will happen only at the write backstage. Until there is no operand forwarding the add can get its data only after the previous Load has written the result because there is a dependency between them.

**(Refer Slide Time: 15:04)**

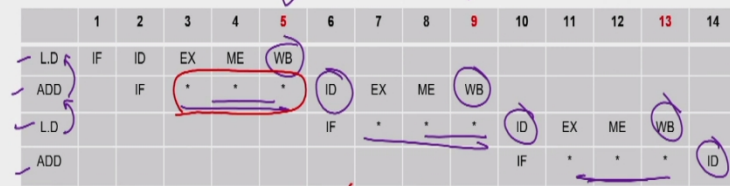


## Pipeline Hazards

**Without operand forwarding.**

ID of nth instruction can be only after WB of n-1th instruction.

3 stalls in each instruction.



Instructions reach WB at clock cycles 5, 9, 13, 17, 21, 25, 29, ....

Last instruction (ADD) reaches WB in  $5 + (1999 \times 4) = 8001$  cycles.

$CPI = 8001/2000 = 4.0005$

$$\frac{8001}{2000} = 4$$

That means the ID of an ADD statement can happen only after the right backstage of previous load. So this add is going to write the result here. And this load is dependent on the previous add. So the ID stage of a load can happen only after the write backstage of the previous add is over. Similarly, you can see that the ID of every instruction can happen only after the write backstage of the previous instruction.

So ID of nth instruction can be only after the WB of the n - 1 instruction. So there are three stalls you can see there are 3 stalls in each of the instruction, if you follow this pattern. So instructions reaches the right backs in the first instruction breaks right backstage at 5 second reach at 9 third reach at 13 fourth reach at 17. So it is a pattern that you can see

5 9 13

like that. So the last ADD instruction will reach that WB stage.

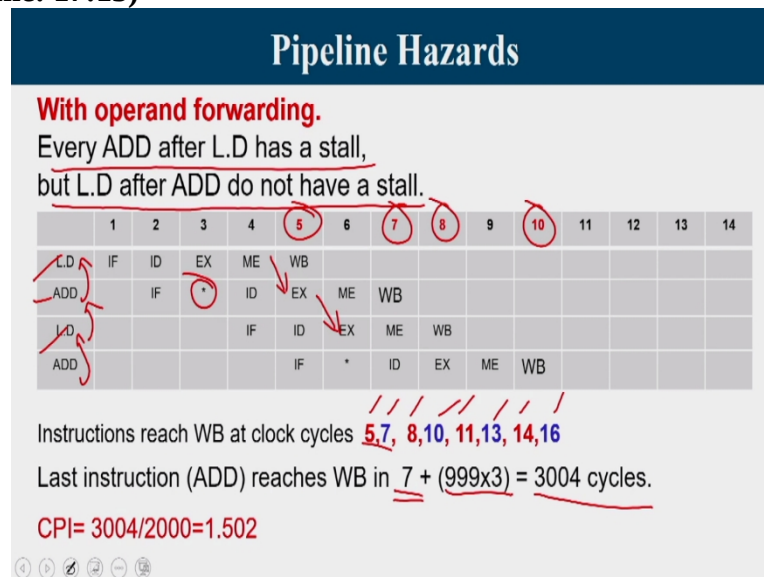
So the first instruction reaches at 5, then I have another 1999 instructions more, each will be completed at the difference for 4 clock cycles, the first instruction will writeback backstage at 5 and then

$5 + 1999 \text{ more instructions}$

which will take 4 so the answer is going to be 8001 clock cycles. So if you look at the CPI value, then it will be 8001 divided by 2000. So it is roughly going to be 4 because you know that every cycle 1-1 instruction is getting over and you have 3 stalls extra.

So the average CPI is going to be 4 in this case. Now, we will see what is with operand forwarding. Now we will see what is with the operand forwarding. So, if it is with operand forwarding, we can see that load add dependency. So, this add instruction is dependent on the load. So add instruction, will get the data only at end of MEM, so MEM is forward into EX. So that is going to incur one cycle stall here. Similarly, this load is dependent on the add. But there it is not a problem.

(Refer Slide Time: 17:13)



Again, this is dependent on the previous Load. So every add after Load has a stall. But Load after ADD do not have a stall, that is peculiarity. So the first instruction completed 5, second at 7 third at 8 fourth at 10 like that. So, if you look at the pattern instructions reach write back at clock cycles, 5, 7, so all the red color markings they are the load instruction. So, load instruction getting completed at 5, 8, 11, 14 like that, add instruction is getting completed at 7, 10, 13, 16 etc. The last instruction is add.

So, add we look into the pattern of add only we look into only the blue. So, first add instruction is getting completed at 7 we have another 999 add instructions more each will be completing at a clock cycle shift of 3 cycles. So at 3004 we have the last ADD instruction completing so what the CPI

$$\text{CPI} = 3004 / 2000 = 1.502 \text{ (roughly)}$$

So, given in this question, what we have seen is if there is a sequence of LOAD and ADD combinations, then if it is without operand forwarding, then every instruction has a stall of 3.

So, normal, an instruction will take

1 cycle to complete + 3 more cycles

So all on average the CPI is 4 and when you come to the operand forwarding mechanism, the second instruction is dependent on the first instruction every add is dependent on the previous load and there is a 1 cycle stall, but every load that is dependent on an add, since there is operand forwarding, there is no stall that happens. So, every alternate instruction will have 1 stall.

So when you have 2000 instruction, you have thousand stalls, so close to 3000 would be the total number of cycles to complete.

$$\text{So the average CPI} = 3000 / 2000 = 1.5$$

Now, let us move into another question which is a branch prediction question. Consider the last 16 actual outcomes of a single branch where T means branch is taken and N means the branch is not taken. So, this is the pattern.

(Refer Slide Time: 19:48)

### Branch Prediction

Consider the last 16 actual outcomes of a single static branch. T means branch is taken and N means not taken.  
{oldest → T T N N T N T T T N T N T T ← latest} (1,2) m n

A two level branch predictor of (1,2) type is used. Since there is only one branch in the program indexing to BHT with PC is irrelevant. Hence only last branch outcome only is used to index to the BHT. How many mis-predictions are there and which of the branches in this sequence would be mis-predicted? Fill up the table for 16 branch outcomes.

00/01 → N  
11/10 → T

So this is the statistics of what is the outcome of a branch is been collected, so T means taken so it is taken, taken, not taken, not taken like that, for the last 16 outcome, this is the latest 1 the latest 1 was taken, the first 1 was taken, second time the iteration was taken third was not taken. That is the way how we how to interpret the sequence. a 2 level brands predictor of 1-2 type is used. Since there is only one branch in the program indexing to BHT branch history tabled with the PC values are relevant.

So 1-2 predictor means, last one outcome of a branch is being used to index into the table and we are using it 2 bit predictor. So, MN predictor basically, if a branch has been mentioned as MN predictor, M stands for last m occurrences of the branch is being used to index in to the branch history table and each entry has an n bit branch predictor. So, we are asking how many mis-predictions are there and which of the branches in the sequence would be mis-predicted. Fill up a table for 16 branch outcomes.

So were basically using 2 bit predictor. So, when the value of the predictor is 00 or 01, we predict that the branch is not taken when the value is 11 or 10 we predict that the branch is taken so, we have to keep this in mind whenever the value is 00 or 01 the prediction is branch will not take and whenever the value is 11 or 10, the prediction is branch will be taken. Now, let us see a transition when we are in 00 we predict that branch is not taken, but if actually the branch is taken then from 00 state how to correct my state 01.

When I am in 01 then if branch is taken actually if it is taken then I changed my prediction value to 11. Similarly, when I am in 11, and if the actual outcome is not taken, then I am moving into 10. Similarly, when I am in 10, when the branch is actually taken, I move into stage 11. So, this is the transition diagram that has to be familiar with us.

(Refer Slide Time: 22:32)

Branch Prediction					
Sl.No	Last Outcome	BHT N/T	Prediction	Outcome	Mis-Pre Y/N ?
1	N (initial)	00 / 11	N	T	YES
2	T	01 / 11	T	T	NO
3	T	01 / 11	T	N	YES
4	N	01 / 10	N	N	NO
5	N	00 / 10	N	T	YES
6	T	01 / 10	T	N	YES
7	N	01 / 00	N	T	YES
8	T	11 / 00	N	T	YES
9	T	11 / 01	N	T	YES
10	T	11 / 11	T	N	YES
11	N	11 / 10	T	T	NO
12	T	11 / 10	T	N	YES
13	N	11 / 00	T	T	NO
14	T	11 / 00	N	T	YES
15	T	11 / 01	N	N	NO
16	N	11 / 00	T	T	NO

Now, what happens here is here it has been mentioned that what will happen to the last 16 outcomes of this branch. So, this will tell you what happened to the branch. So, the first iteration the branch was taken, second iteration branch was not taken like that the 16th iteration branch was taken. Now, let us see this table. This table looks a bit clumsy, when you

try to work out this question for the first time I make it more simple, this is your branch history table entry.

So, the branch history table has 2 columns. The first column tells when the branch entry is not taken, you have to consult the first column when the branch entry is taken, then you have to look into the second entry. So, it is basically a 2 entry table. So, this is the 2 entries, when mentioned that we are mentioning 1 2 predictor. So, you look into what is the outcome of the last branch, if the outcome of the last branch is not taken, then refer this the last branch is taken then refer the second entry.

So, to interpret into the table, if the last branch was actually not taken, refer this entry, if the last branch was actually taken, then refer the second entry. So referring into first and second entry is the most crucial thing as far as working with this branch table is concerned. Now, the initial configuration of the branch is 00/11. So, my first is one, my initial condition is N last outcome.

(Refer Slide Time: 24:10)

00/01 → N  
10/11 → T

### Branch Prediction

Sl.No	Last Outcome	BHT N/T	Prediction	Outcome	Mis-Pre Y/N ?	
1	N (initial)	00 / 11	N	T	YES	T ✓
2	T	01 / 11	T	T	NO	T ✓
3	T	01 / 11	T	N	YES	N ✓
4	N	01 / 10	N	N	NO	N ✓
5	N	00 / 10	N	T	YES	T
6	T	01 / 10	T	N	YES	T
7	N	01 / 00	N	T	YES	T
8	T	11 / 00	N	T	YES	T
9	T	11 / 01	N	T	YES	T
10	T	11 / 11	T	N	YES	N
11	N	11 / 10	T	T	NO	T
12	T	11 / 10	T	N	YES	N
13	N	11 / 00	T	T	NO	T
14	T	11 / 00	N	T	YES	T
15	T	11 / 01	N	N	NO	N
16	N	11 / 00	T	T	NO	T

Now I am referring into this one, since the last outcome for the initial case to start with let us assume that it is not taken since it is not taken I refer into the first entries 00. since the entries 00 we have to understand that if the entry in the table is 00 or 01, then the prediction is not taken if the entry is 1 0 or 1 1, then the prediction is taken. So since the entry is 00, I predict it has not taken but what actually happened with the first branch was taken.

So my prediction was not taken, but the actual outcome was taken that is a mis-prediction, I predict that branch will not take but actually the branch was taken and that is known as mis-prediction yes or no. This is a mis-predicted case. Now when we move into the next entry, I have to make some changes, my state was 00. But the prediction when the wrong when prediction went wrong from 00 I moved to 01 that is what has been the change that you see.

So at the end of the first row, the table gets itself updated to 01 / 11, why this is 11, I am not going to make any change to that entry which has not referred I have referred only the first entry 00 and first interest 00 based upon a predictor it has not taken actual outcome was taken. So it is a mis-prediction whenever there is a mis-prediction, I how to change the state. Now, the second row works with the new updated table. So, what is a new table the new table is 01/11 and this is already taken.

Now, the last outcome of the branch was taken. So this will get repeated. So, these both will be same whatever was the last outcome that is been entered here. So the last outcome was taken, since the last outcome was taken look into the second portion of the table when the last outcome is not taken look into the first portion of table, last outcome is taken look into the second portion of the table.

The second portion is 11 that means a prediction will be taken whenever the stores that whenever the entries 11 the prediction is taken, the actual outcome is also taken that means it is not a mis-prediction. So, in this case, the third row is same as this. Now, the last outcome was taken how will I get this, this was the last outcome was taken I refer in to this it is predicting that the branch will be taken, but actually the third time the branch was not taken. So that is a mis-prediction.

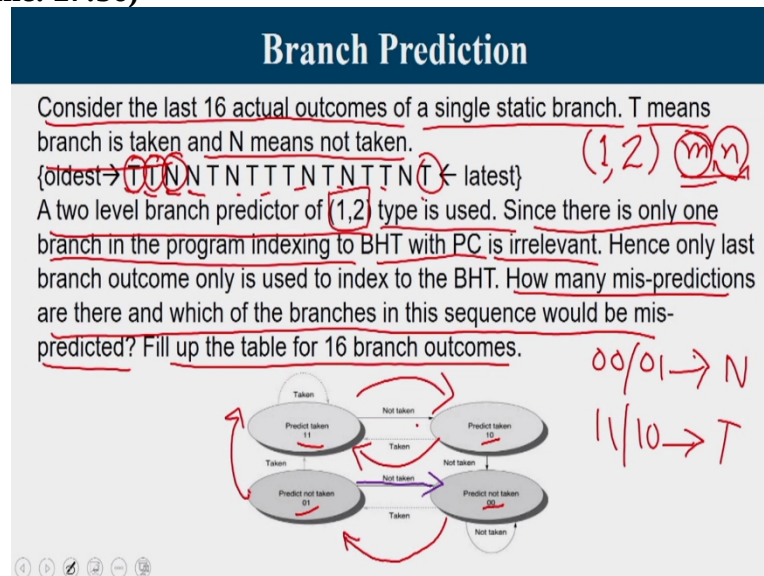
Once it is a mis-prediction from 11, when I miss predict I moved to 10 that is what is happening here. So, that is the change that is being reflected I move into 10, now I continue with the fourth iteration, the previous 1 was not taken. So, I go here this is not taken, since it is not taken I refer to the first half the first half is 01 indicates not taken actual outcome was also not taken. So, there is a perfect match there.

So, when at 01 when the branch was not taken, then we have to understand that it will go back to 00 state. So, this is our table, when you are in 01 when the branch is not taken, I



move into 00 and then the previous outcome was not taken. So I continues. So, to fill up this table, whatever is here, the same thing will get repeated in this in the next row. This is the outcome that outcome is used for the subsequent rows.

(Refer Slide Time: 27:56)



(Refer Slide Time: 28:03)

### Branch Prediction

00/01 → N  
11/10 → T

Sl.No	Last Outcome	BHT N/T	Prediction	Outcome	Mis-Pre Y/N ?
1	N (initial)	00 / 11	N	T	YES
2	T	01 / 11	T	T	NO
3	T	01 / 11	T	N	YES
4	N	01 / 10	N	N	NO
5	N	00 / 10	N	T	YES
6	T	01 / 10	T	N	YES
7	N	01 / 00	N	T	YES
8	T	11 / 00	N	T	YES
9	T	11 / 01	N	T	YES
10	T	11 / 11	T	N	YES
11	N	11 / 10	T	T	NO
12	T	11 / 10	T	N	YES
13	N	11 / 00	T	T	NO
14	T	11 / 00	N	T	YES
15	T	11 / 01	N	N	NO
16	N	11 / 00	T	T	NO

Handwritten notes on the right side of the table: T ✓, T ✓, N ✓, N ✓, T, N, T, T, T, T, N, T, N, T, N.

So, if you continue like that, we have to understand I have to update only that entry which I have referred if the entry that I have referred is correct, then I make the predictions, I look into what is the outcome based on the outcome I make appropriate changes and update the entries downward. This is the way how we distribute flourished. So, wherever this green color that has been shown, these are not these are cases where the prediction and the outcome was matching.

In all other cases, where the S is marked, these are places where I got mis- predictions. So, I got total of 10 mis-predictions from this entire setup. So, what has been given here is you have given a structure of a table this is a 1 2 predictor. So, last one occurrence only is been checked and based upon last one occurrence is the last occurrence is not taken you refer in to the first portion of the table, if the last occurrence is taken you refer into the second portion of the table.

Now the contents of this table you look into 00 or 01 then the prediction is not taken if the content of the table is 10 or 11, then the prediction will become taken. Now, you see correlate with what is actually happening this is a sequence which is telling what the outcome of the branch. If they are perfectly matching then there is no mis-prediction. If there is a mismatch, so the value that we predicted and the actual outcome is different that there is a mismatch and then we have to correlate that.

So with this we come to the end of the tutorial. In this tutorial, we have seen about how to handle with the hazards few true or false statements were there. And then we had few numerical problems in handling with the hazards and then some dependency issues with and without operand forwarding and towards the end we had branch predictor. This week also we will put up some short exercise questions also for you to work on with respect to branches. Thank you