

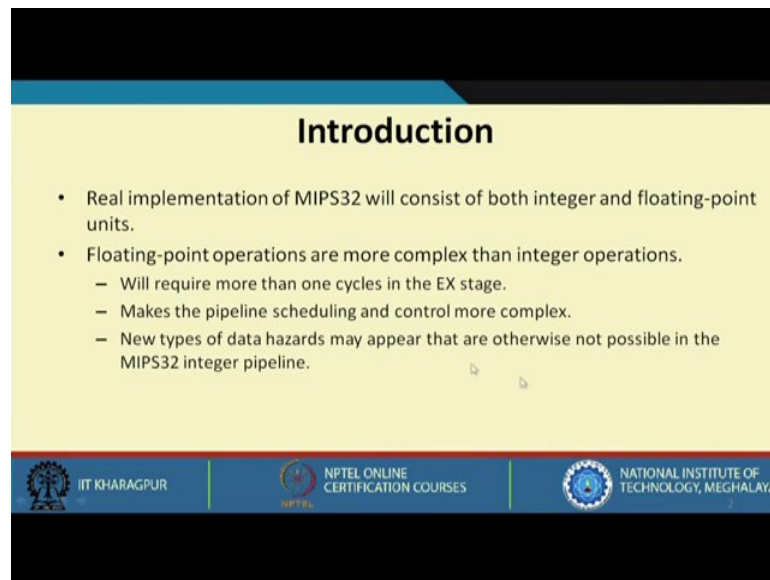
**Computer Architecture and Organization**  
**Prof. Indranil Sengupta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 59**  
**Multicycle Operations in MIPS32**

In our earlier discussions on the MIPS32 pipeline, we basically concentrated on integer instructions. We had seen how the pipeline works, we had seen the effects of the hazards, the interrupts and various ways to handle these problems and to improve the performance. But now let us come to the real scenario. In a real computer, we do not only have integer operations or instructions, we need also to process scientific data that are in the form of floating point numbers. So, we need floating point arithmetic. Floating point processing are done in the computer system itself now. So far we have assumed our EX stage will be able to finish computation in a single cycle, but that is a little too optimistic realistically speaking. Can you finish a multiplication or a division operation in one cycle?

Well yes, provided you make the clock sufficiently slow, but that is not a good idea. If you do that everything else will also become slow,, moreover we have seen that for floating point operations, you need multiple cycles to carry out the floating point operation itself which means essentially your EX cycle itself may required multiple clocks. So, we are talking about something called multi cycle operations. The topic of our lecture today is multicycle operation in MIPS32.

(Refer Slide Time: 02:24)



**Introduction**

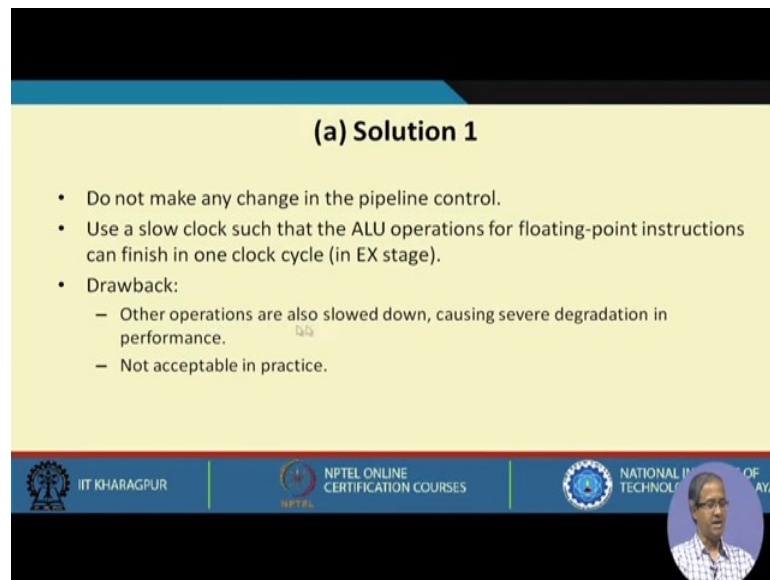
- Real implementation of MIPS32 will consist of both integer and floating-point units.
- Floating-point operations are more complex than integer operations.
  - Will require more than one cycles in the EX stage.
  - Makes the pipeline scheduling and control more complex.
  - New types of data hazards may appear that are otherwise not possible in the MIPS32 integer pipeline.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

As I said, real implementation of MIPS32, where you can run all kind of applications, will contain both integer and floating-point units. Floating-point operations are more complex than integer operations; not only that, even within integer operations multiply and divide can be more complex than add and subtract.

These kind of operations are slower in general and may require more than one cycle to finish during the EX stage. This obviously makes the pipeline scheduling and controlling much more complex and we shall see more interesting kinds of hazards can appear because of this. Earlier what we assumed was that that all instructions are of the same size, they take same time. But now when we say instructions can vary in time in terms of execution we are landing ourselves into a big trouble. May be the earlier instruction will be finishing later, next instruction will be finishing earlier; lots of such problems can occur.

(Refer Slide Time: 03:55)



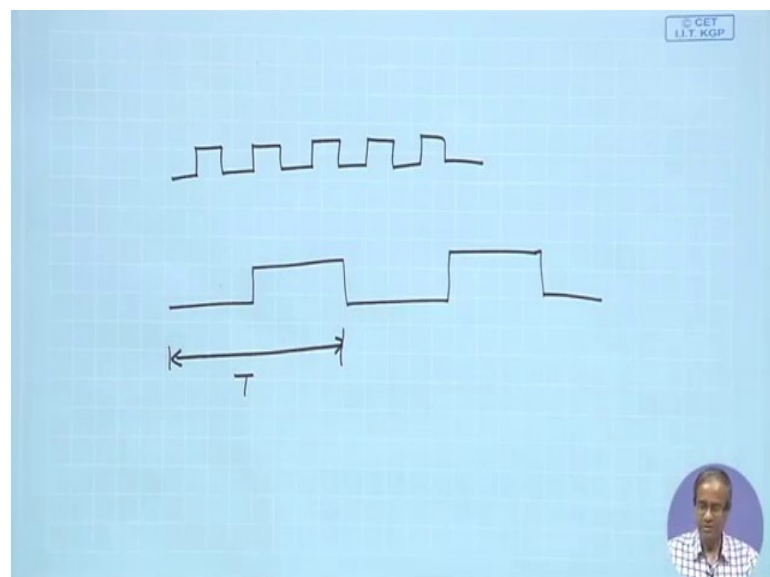
**(a) Solution 1**

- Do not make any change in the pipeline control.
- Use a slow clock such that the ALU operations for floating-point instructions can finish in one clock cycle (in EX stage).
- Drawback:
  - Other operations are also slowed down, causing severe degradation in performance.
  - Not acceptable in practice.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY KHARAGPUR

The first and very naive solution is not to make any change in the control, just slow down the clock. Your operation may be more time consuming, but you give sufficient time within a clock period to complete it. So, maybe earlier your clock was like this, this was your clock signal.

(Refer Slide Time: 04:22)



© CET IIT KGP

The image shows two hand-drawn clock signals on a grid background. The top signal is a high-frequency square wave. The bottom signal is a lower-frequency square wave. A horizontal double-headed arrow below the bottom signal is labeled with the letter 'T', representing its period.

But now you make the clock like this. Obviously, this is not a good idea because making clock period slow means your all your stages, not only EX; IF, ID, EX, MEM, WB, they will all be running at this speed  $T$ . So, if you slow down the clock everything will slow

down. This will cause a severe degradation in performance and; obviously, we cannot use it in practice.

(Refer Slide Time: 05:11)

**(b) Solution 2**

- We allow the floating-point arithmetic pipeline to have a longer latency.
  - EX cycle is considered to be repeated several times.
  - The number of repetitions can vary depending on the operation.

IF ID EX EX EX EX MEM WB

- The EX stage will have multiple floating-point functional units.
  - For example, one for addition/subtraction (pipelined), one for multiplication (pipelined), and one for division (non-pipelined).
  - A stall will occur in the pipeline if the instruction to be issued will either cause a structural hazard for the functional unit, or a data hazard.
    - Pipelining the functional units can avoid the structural hazard.

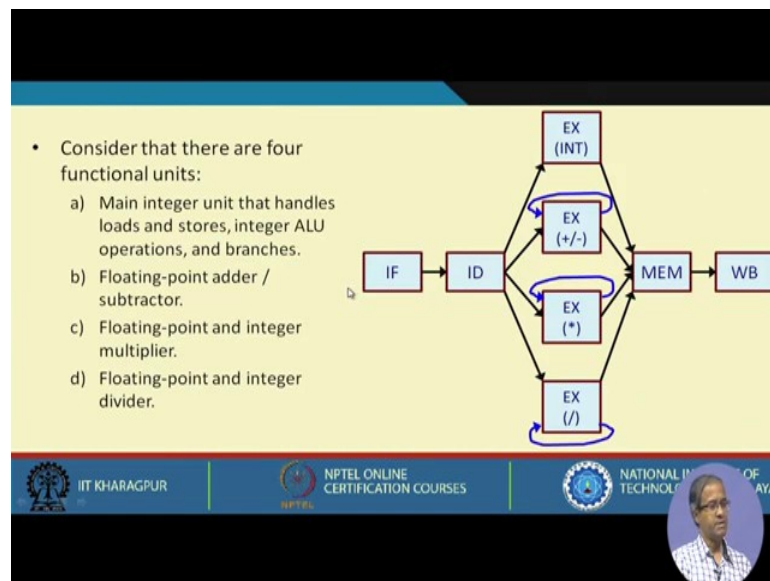
IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES NATIONAL INSTITUTE OF TECHNOLOGY KANPUR

So, what is the more practical solution? We do not change or modify IF, ID, MEM, WB because they do not change, they remain as it is. The only concern is the EX stage, where the arithmetic operations are carried out, which here we are saying can vary in complexity. Some are integer, some are floating point, even within integer, they can be of various complexities. Let us assume integer and floating point for the time being, we allow floating point arithmetic pipeline to have a longer latency. Logically speaking what we are saying is that in the instruction execution cycle, the EX cycle is repeated for more than one clock cycles.

So, we are repeating the EX cycle several times. How many times we are repeating depends on the complexity of the operation. In a real computing platform, the EX stage will actual be having multiple functional units; not just one. May be there will be one functional unit that will be handling additional subtraction, there will be one functional unit for multiplication, one functional unit for division. Now there can be stalls due to structural hazards, also there is a possibility of a stall if you are trying to issue an instruction (means your instruction moves from ID to EX), it may so happen that the earlier instruction which was there in the pipeline was using the same functional unit and it was still somewhere in EX.

So, the next instruction cannot enter EX, unless the EX or the arithmetic circuitry is itself pipelined. You recall, we discussed pipeline implementation of some of the floating point operations, this is why we need that. If they are not pipelined, then while a previous instruction is doing multiplication the next instruction has to wait for the multiplication till the earlier one completes. But if the multiply operation itself is pipelined, when the first instruction is in the second stage of multiplication, the next instruction can enter the first stage. So, they can proceed in overlapped fashion.

(Refer Slide Time: 08:38)



So, if you have pipelining in the functional unit, you can avoid structural hazard. Otherwise, there will be a structural hazard in the EX stage. This is a logical picture where we are assuming that the EX stage has logically 4 functional units. The first one is or conventional integer unit which we have seen earlier. For that the sequence will be IF, ID, EX, MEM, WB. The main integer unit will be handling all load and store operations from memory, integer ALU operations and also branches. There is a floating point adder, subtractor, another functional unit, this blue arrow indicates that you may have to iterate it several times. This is a multicycle EX stage. There can be a floating point multiplier and also integer multiplier. And this is a division (floating point or integer) unit.

As you can see, other than the integer EX unit, other EX units are multicycle, they will need multiple cycles of EX.

(Refer Slide Time: 10:01)

**MIPS32 Floating-Point Extension**

- In the floating-point extension of MIPS32, there are 32 floating-point registers F0 to F31, each of size 32 bits.
- For double-precision operations, register pairs can be used to store the data:
  - Register pair <F0, F1> → referred to as F0
  - Register pair <F2, F3> → referred to as F2
  - Register pair <F30, F31> → referred to as F30
- Some examples of double-precision floating-point instructions are shown in the next slide.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY KHARAGPUR

Earlier we briefly looked at the MIPS32 floating point registers and some of the instructions there. In the MIPS32, there are 32 floating point registers which are named F0 to F31, they are all of 32 bit size. When we work with double precision numbers, that means 64 bits, then we can take register pairs; like F0 and F1, you can take together F2 and F3 and so on, similarly F30 and F31. Like this, you can use register pairs for double precision operations.

(Refer Slide Time: 11:06)

**Floating-Point Instruction Examples**

a) Load into a floating-point register pair:  
`L.D F6, 200(R2) // F6 = Mem[R2+200]; F7 = Mem[R2+204];`

b) Store from a floating-point register pair:  
`S.D F4, 40(R5) // Mem[R5+40] = F4; Mem[R5+44] = F5;`

a) Arithmetic operations on floating-point register pairs:  
`ADD.D F0, F4, F6  
SUB.D F12, F8, F20  
MUL.D F4, F6, F8  
DIV.D F8, F8, F10`

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY KHARAGPUR

Some floating-point instruction examples are shown in this slide.

(Refer Slide Time: 12:26)

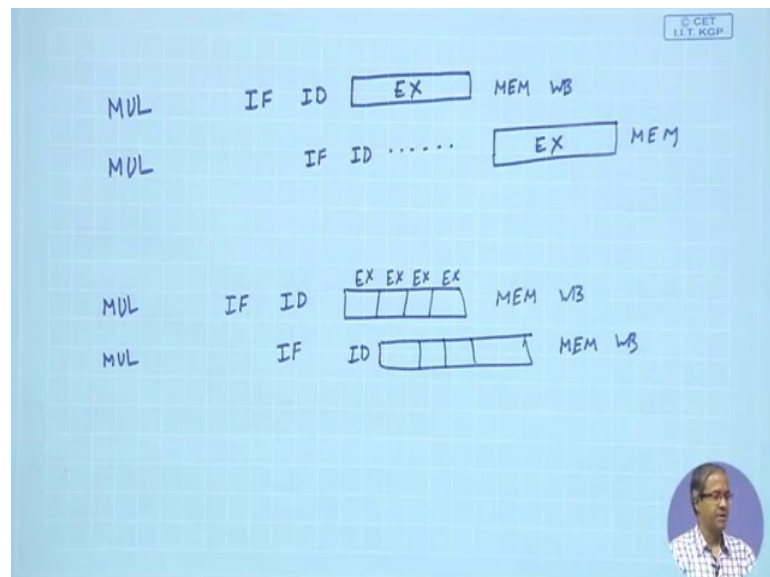
**Latency and Initiation Interval**

- The multi-cycle arithmetic units are often pipelined to allow overlapped operation and hence improved performance.
- Definitions:
  - a) **Latency:** The number of cycles between an instruction producing a result and another instruction using it.
  - b) **Initiation Interval:** The number of cycles that must elapse between issuing two operations of the same type.

Logos: IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, NATIONAL INSTITUTE OF TECHNOLOGY Kharagpur

We talk about latency and initiation intervals in this kind of multicycle pipelines. I mentioned that arithmetic units are often pipelined wherever possible because unless you do a pipeline, you cannot overlap the operations. I am giving a small example. Suppose there is one MUL instructions followed by another MUL instruction.

(Refer Slide Time: 13:06)



The way the MUL instructions can proceed in overlapped fashion in the EX stage is shown.

Now latency means we are talking about the data dependencies; there is one instruction that is producing a result and another instruction which is using it. So, how many minimum number of clock cycles will be required between them? We saw earlier that a load instruction followed by its use in the integer case requires 1 stall cycle, which means latency is 1. But for floating point operations latency can be more.

And initiation interval means how frequently we can issue instructions of the same type. If it is fully pipelined, we can issue it every cycle, like in the previous example of multiplication I have shown. These are the typical values that we will assume; these values are very realistic.

(Refer Slide Time: 15:22)

**Typical Values Assumed**





Functional Unit	Latency	Initiation Interval
Integer ALU	0	1
Data Memory (integer / FLP load)	1	1
FP add / subtract	3	1
FP multiply	6	1
FP divide	24	25

Assumptions on number of EX stages:

- FP add/subtract: 4
- FP multiply: 7
- FP divide: 1 (not pipelined)

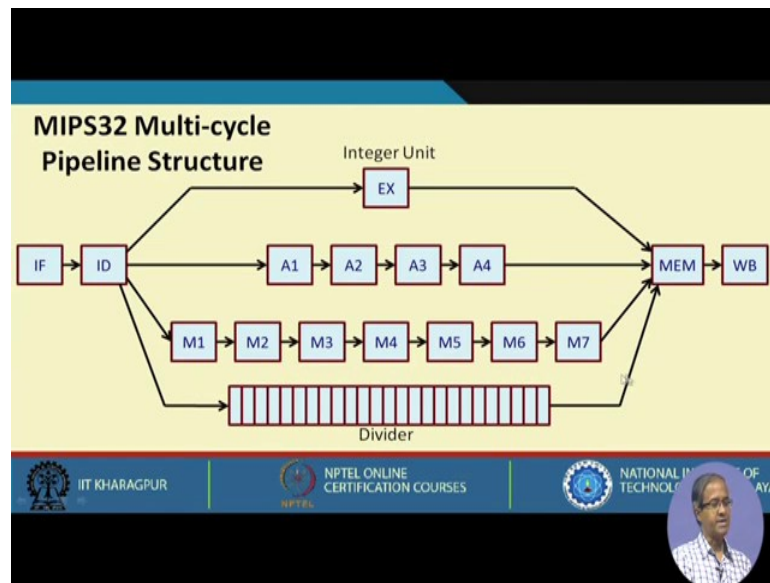
It is possible to have up to:

- 4 outstanding FP add/subtract
- 7 outstanding FP multiply
- 1 FP divide.



(Refer Slide Time: 17:35)



So, our architecture looks like this. These are the pipeline stages, IF ID MEM and WB, here these are the EX stages. The integer unit has a single EX, for floating point addition there are 4 stages, for floating point multiplication there are 7 stages, and division is a single stage, but with delay of 25. It is a non pipelined, this is how typically you see division. The reasons is the cost of pipelining a divider is much more, but otherwise the frequency of division is much less as compared to other operation like addition, subtraction, multiplication.

(Refer Slide Time: 18:51)

**Some Scenarios**

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

**Out of order completion of instructions**

L.D F8,0(R5)	IF	ID	EX	MEM	WB												
MUL.D F4,F8,F10		IF	ID	-	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
ADD.D F6,F4,F12			IF	-	ID	-	-	-	-	-	A1	A2	A3	A4	MEM	WB	
S.D F6,0(R5)					IF	-	-	-	-	-	ID	EX	-	-	-	MEM	WB

**Stalls arising due to RAW hazards**





There are some RISC architectures where division is not pipelined. Some interesting scenarios you can see here; I have shown one multiply followed by add; these are all double load and store. MUL will need 7 cycles in the execution stage M1 to M7; ADD will require 4 cycles. Now you see out of order completion of instructions. The third instruction finishes first, fourth instruction finishes next, then the second instruction, then the first instruction. There will be out of order completion of instructions; then second instruction illustrates some read after write hazards. The required stall cycles are also shown.

We can see that there are so many stalls because of RAW hazards in floating point operations. The number of stall cycles is significantly higher for multicycle operations. So, the compiler needs to take care of these kind of things; some of these techniques will be discussed.

(Refer Slide Time: 21:53)

MUL.D F4,F8,F10	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
---		IF	ID	EX	MEM	WB					
---			IF	ID	EX	MEM	WB				
SUB.D F6,F8,F10				IF	ID	A1	A2	A3	A4	MEM	WB
---					IF	ID	EX	MEM	WB		
---						IF	ID	EX	MEM	WB	
L.D F6,0(R5)							IF	ID	EX	MEM	WB

- Three instructions are trying to write into the FP register bank simultaneously.
  - WAW hazard for the last two conflicting instructions (both writing F6).
- No conflict in MEM as only the last instruction accesses memory.

Now another interesting example is here. This results in a write-after-write hazard that was never possible in an integer pipeline. See there are 3 instructions. They are the independent instructions, but the point is that they are writing into some registers. Multiply will require 7 EX stages, subtract will require 4 EX stages, load will require 1, but interestingly we see that all reach the WB stages together. So, there can be structural hazard in the WB stage, first one is trying to write into F4, this one is trying to write into F6, this is also trying to write into F6. So, there is a conflict here that needs to be

handled; these instructions should not be allowed to reach WB stage at the same time. Some stall cycles in WB also needs to be inserted here, but for MEM in this example, there is no conflict because only load is accessing memory.

The conflict in WB is important here; you need to insert stalls. With this, we come to the end of this lecture, where we tried to appraise you about the problems that arise in muticycle operations; what are the kind of issues and hazards that can show up and of course, in a real machine there has to be a very sophisticated mechanism to handle all these kind of problems.

Thank you.