

Multicore Computer Architecture - Storage and Interconnects

Lecture 3

Introduction to Superscalar Pipelines

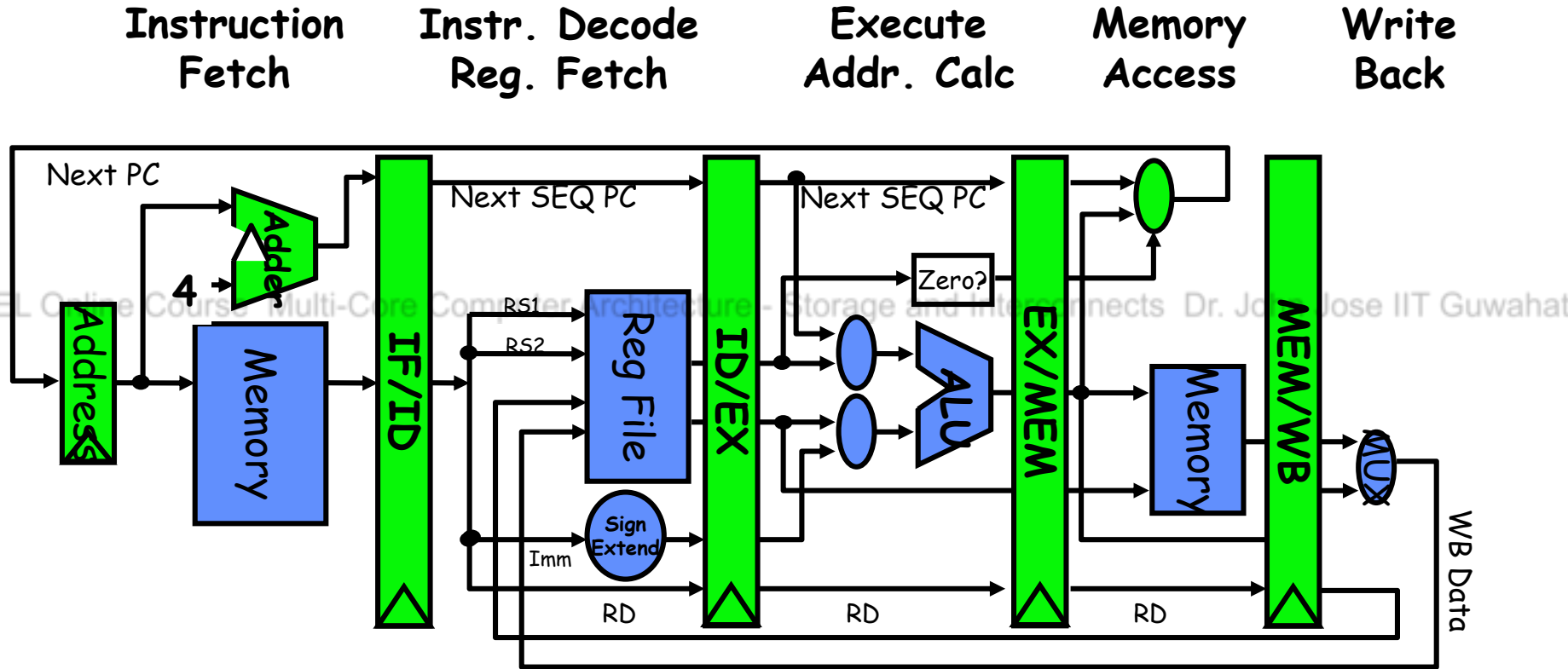


Dr. John Jose

Assistant Professor

**Department of Computer Science & Engineering
Indian Institute of Technology Guwahati, Assam.**

Pipelined RISC Data path



Multi-cycle Operations

- ❖ Some operations require more than 1 clock cycle to complete.

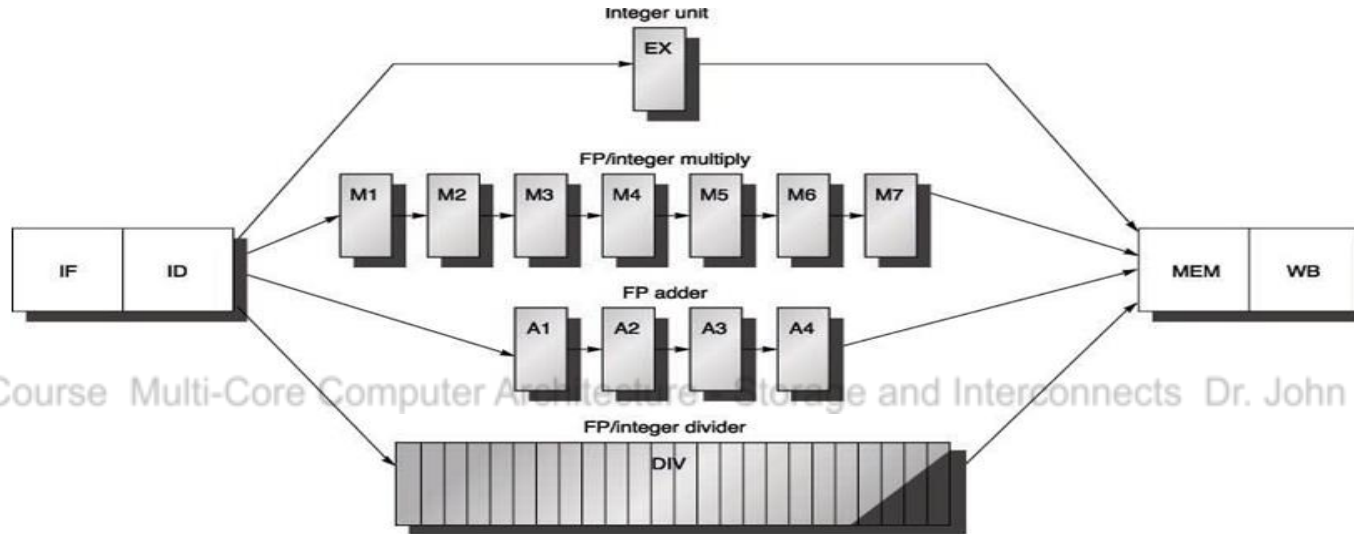
- ❖ Floating Point Multiply

- ❖ Floating Point Divide

- ❖ Floating Point Add

- ❖ Hardware is available on the processor for performing the operations.

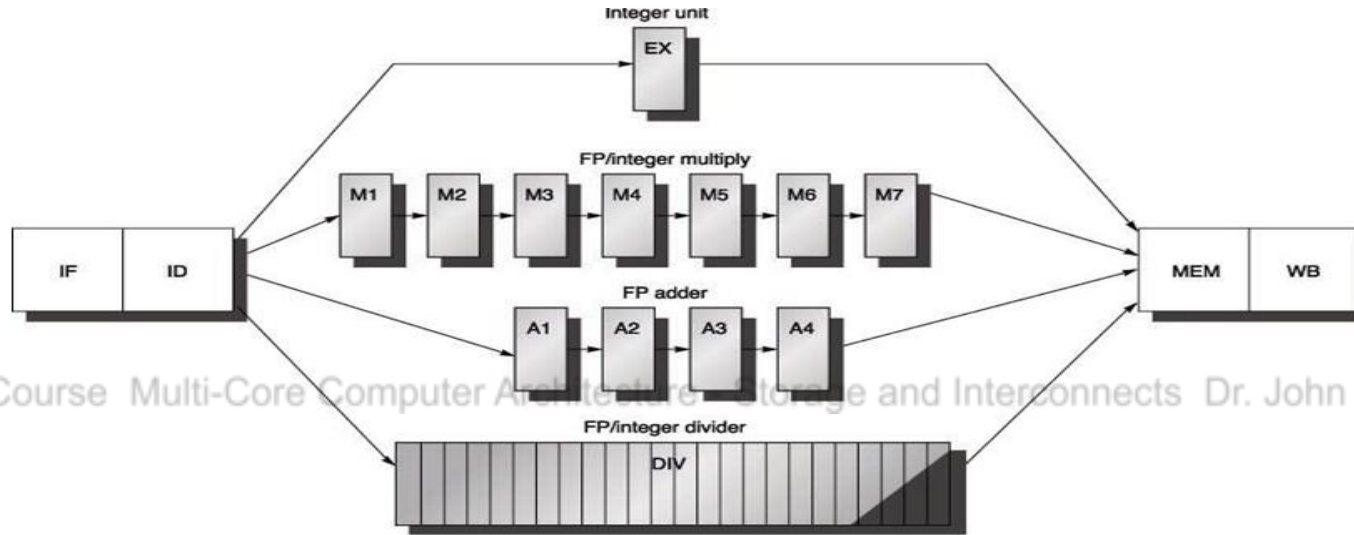
Multi-cycle Operations



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.

Multi-cycle Operations



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

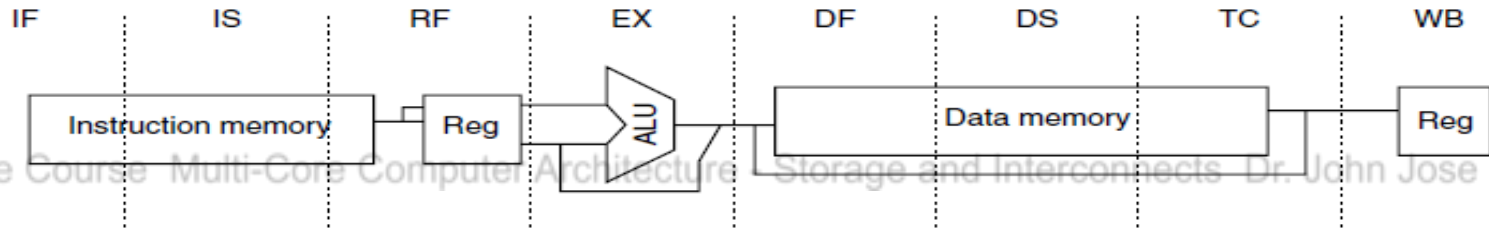
Issues in Longer Latency Pipelines

- ❖ Since divide unit is not-pipelined – structural hazard
- ❖ Instruction have varying runtimes—more register writes/cycle
- ❖ Resolve write port conflicts in ID stage
- ❖ WAW hazards possible
- ❖ Out of order completion
- ❖ Stalls for RAW hazards will be more

Instruction	Clock cycle number																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
L.D F4,0(R2)	IF	ID	EX	MEM	WB												
MUL.D F0,F4,F6		IF	ID	stall	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
ADD.D F2,F0,F8			IF	stall	ID	stall	stall	stall	stall	stall	stall	A1	A2	A3	A4	MEM	WB
S.D F2,0(R2)					IF	stall	stall	stall	stall	stall	stall	ID	EX	stall	stall	stall	MEM

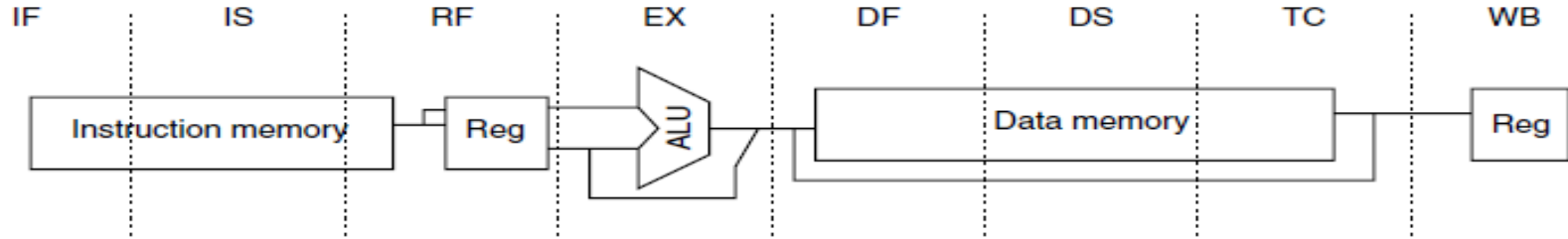
Deeper Pipelines

- ❖ MIPS 4000: 8 stage pipeline, Memory stages span across multiple clock cycles.(super pipelining)



- ❖ IF—First half of instruction fetch; PC selection actually happens here, together with initiation of instruction cache access.
- ❖ IS—Second half of instruction fetch, instruction cache access.
- ❖ RF—Instruction decode and register fetch, hazard checking, and also instruction cache hit detection.

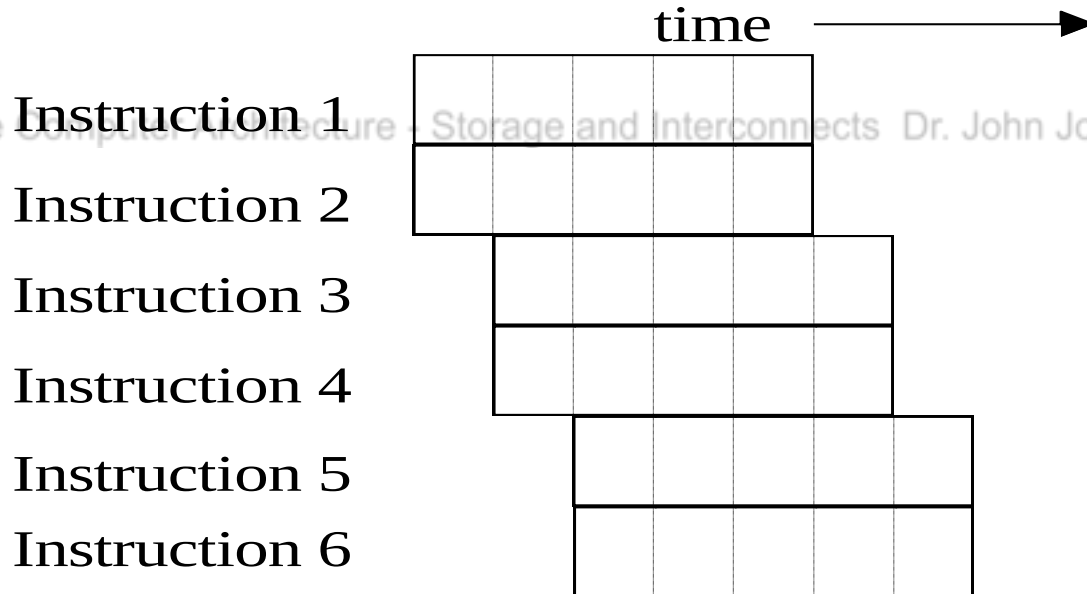
Deeper Pipelines



- ❖ EX—Execution, which includes effective address calculation, ALU operation, branch-target computation and condition evaluation.
- ❖ DF—Data fetch, first half of data cache access.
- ❖ DS—Second half of data fetch, completion of data cache access.
- ❖ TC—Tag check, determine whether the data cache access hit.
- ❖ WB—Write back for loads and register-register operations.

Superscalar – multiple-issue

- ❖ Fetch (and execute) more than 1 instructions at one time
- ❖ Expand every stage to accommodate multiple instructions



Compiler Techniques for Exposing ILP

- ❖ Find and overlap sequence of unrelated instruction
- ❖ Compiler scheduling
 - ❖ Separate dependent instruction from the source instruction by pipeline latency of the source instruction

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Pipeline Stalls

Loop: L.D F0,0(R1)

stall

ADD.D F4,F0,F2

stall

stall

S.D F4,0(R1)

DADDUI R1,R1,#-8

stall (assume integer load latency is 1)

BNE R1,R2,Loop

for (i=999; i>=0; i=i-1)

$x[i] = x[i] + s;$

NPTTEL Online Course - Computer Architecture - Storage and Interconnects Dr. John Jose IIT Guwahati

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Pipeline Scheduling

Loop: L.D F0,0(R1)

stall

ADD.D F4,F0,F2

stall

stall

S.D F4,0(R1)

DADDUI R1,R1,#-8

stall (assume integer load latency is 1)

BNE R1,R2,Loop

Scheduled code:

Loop: L.D F0,0(R1)

DADDUI R1,R1,#-8

ADD.D F4,F0,F2

stall

stall

S.D F4,8(R1)

BNE R1,R2,Loop

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Loop Unrolling

Loop: L.D F0,0(R1)

stall

ADD.D F4,F0,F2

stall

stall

S.D F4,0(R1)

DADDUI R1,R1,#-8

stall

BNE R1,R2,Loop

Loop unrolling

Loop: L.D F0,0(R1)

ADD.D F4,F0,F2

S.D F4,0(R1) ; // drop DADDUI & BNE

L.D F6,-8(R1)

ADD.D F8,F6,F2

S.D F8,-8(R1) ; // drop DADDUI & BNE

L.D F10,-16(R1)

ADD.D F12,F10,F2

S.D F12,-16(R1) ; // drop DADDUI & BNE

L.D F14,-24(R1)

ADD.D F16,F14,F2

S.D F16,-24(R1)

DADDUI R1,R1,#-32

BNE R1,R2,Loop

- ❖ Unroll by a factor of 4
- ❖ Eliminate unnecessary instructions

Loop Unrolling/Pipeline Scheduling

❖ Loop unrolling

Loop: L.D F0,0(R1)
 ADD.D F4,F0,F2
 S.D F4,0(R1)
 L.D F6,-8(R1)
 ADD.D F8,F6,F2
 S.D F8,-8(R1)
 L.D F10,-16(R1)
 ADD.D F12,F10,F2
 S.D F12,-16(R1)
 L.D F14,-24(R1)
 ADD.D F16,F14,F2
 S.D F16,-24(R1)
 DADDUI R1,R1,#-32
 BNE R1,R2,Loop

❖ Scheduled, Unrolled loop

Loop: L.D F0,0(R1)
 L.D F6,-8(R1)
 L.D F10,-16(R1)
 L.D F14,-24(R1)
 ADD.D F4,F0,F2
 ADD.D F8,F6,F2
 ADD.D F12,F10,F2
 ADD.D F16,F14,F2
 S.D F4,0(R1)
 S.D F8,-8(R1)
 DADDUI R1,R1,#-32
 S.D F12,16(R1)
 S.D F16,8(R1)
 BNE R1,R2,Loop

Dynamic Scheduling

- ❖ Rearrange execution order of instructions to reduce stalls while maintaining data flow
- ❖ Advantages:
 - ❖ Compiler doesn't need to have knowledge of micro-architecture
 - ❖ Handles cases where dependencies are unknown at compile time
- ❖ Disadvantage:
 - ❖ Substantial increase in hardware complexity

How dynamic scheduling works ?

- ❖ Limitation of simple pipelining.
 - ❖ In-order instruction issue and execution.
 - ❖ Instructions are issued in program order.
 - ❖ If an instruction is stalled in the pipeline, no later instructions can proceed.
- ❖ If instruction j depends on a long-running instruction i, currently in execution in the pipeline, then all instructions after j must be stalled until i is finished and j can execute.

DIV.D

F0, F2, F4

ADD.D

F10, F0, F8

SUB.D

F12, F8, F14

How dynamic scheduling works ?

DIV.D	F0, F2, F4
ADD.D	F10, F0, F8
SUB.D	F12, F8, F14

- ❖ Separate the issue process into two parts:
 - ❖ checking for any structural hazards.
 - ❖ waiting for the absence of a data hazard.
- ❖ Use in-order instruction issue but we want an instruction to begin execution as soon as its data operands are available.
- ❖ out-of-order execution → out-of-order completion.
- ❖ OOO execution introduces the possibility of WAR and WAW hazards

How dynamic scheduling works ?

- ❖ To allow out-of-order execution, **split the ID stage into two**
 - ❖ **Issue**—Decode instructions, check for structural hazards.
 - ❖ **Read operands**—Wait until no data hazards, then read operands.
- ❖ In a dynamically scheduled pipeline, all instructions pass through the **issue stage in order (in-order issue)**; however, they can be stalled or bypass each other in the second stage (read operands) and thus enter **execution out of order**.
- ❖ **Done by - score boarding technique**
- ❖ **Approach used - Tomasulo's algorithm**

Dynamic Scheduling - Summary

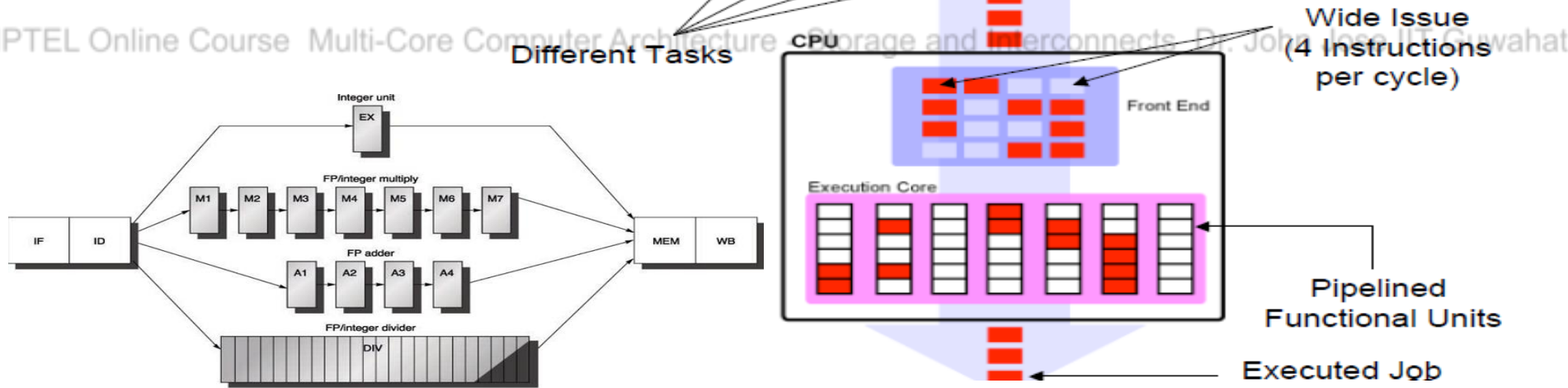
- ❖ Dynamic scheduling implies:
 - ❖ Out-of-order execution
 - ❖ Out-of-order completion
- ❖ Creates the possibility for WAR and WAW hazards

- ❖ **Solution : Tomasulo's Approach**

- ❖ Tracks when operands are available
 - ❖ Introduces register renaming in hardware
 - ❖ Minimizes WAW and WAR hazards

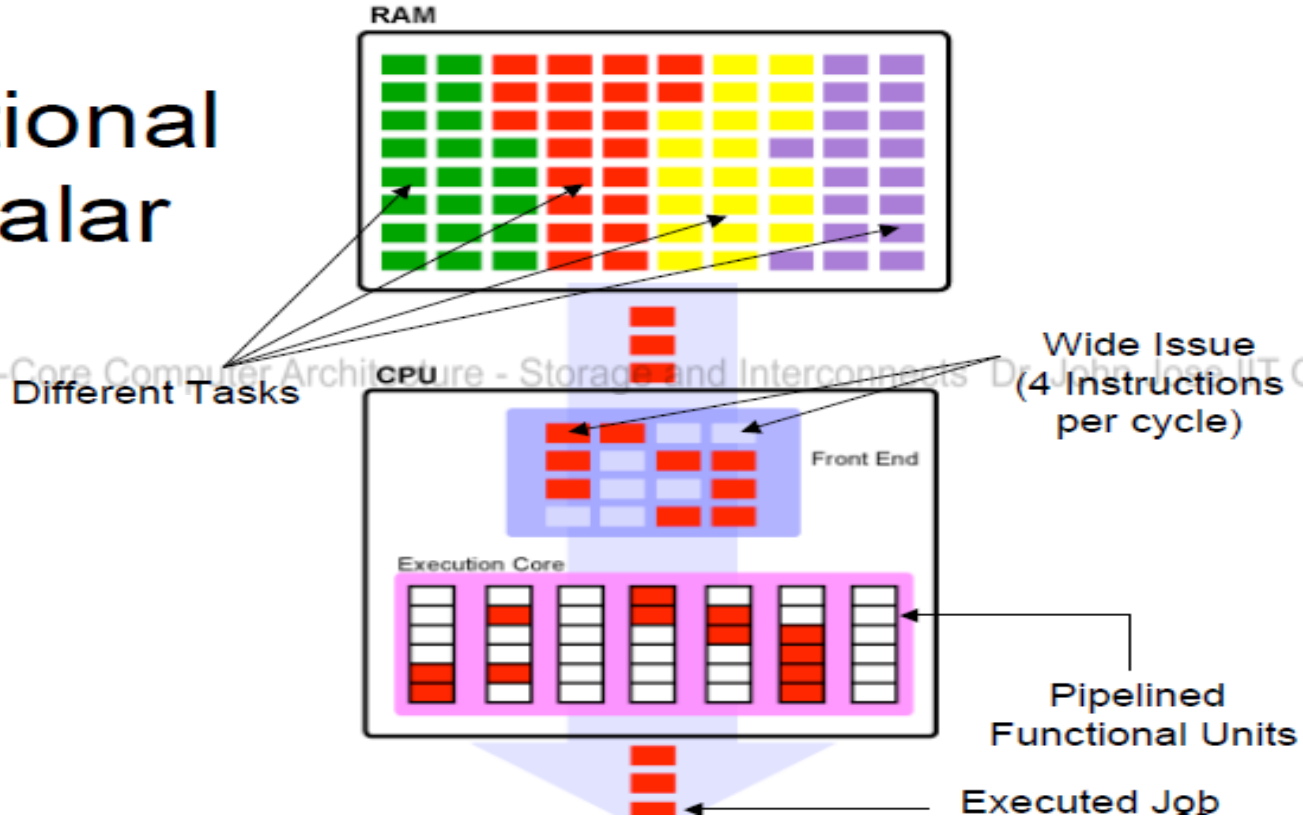
Superscalar Processor

Conventional Superscalar



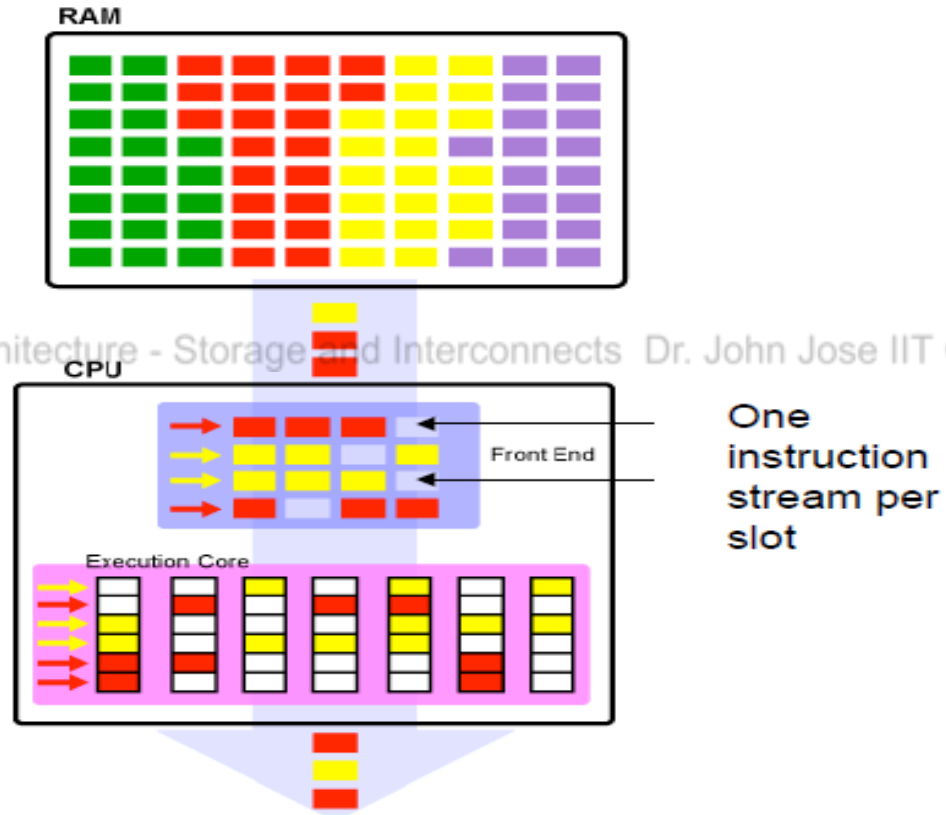
Superscalar Processor

Conventional Superscalar



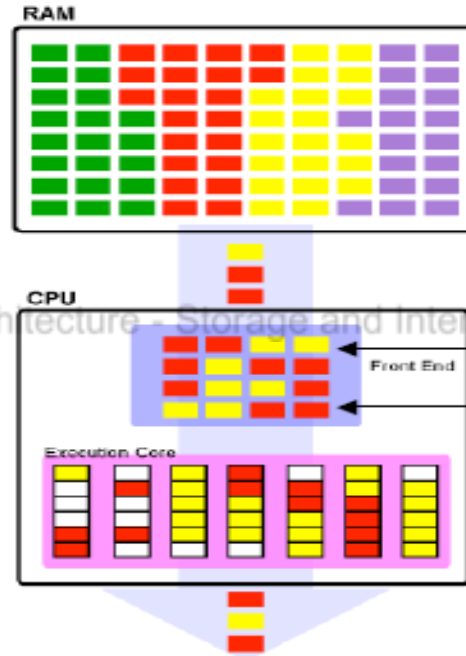
Multithreading

Multithreading



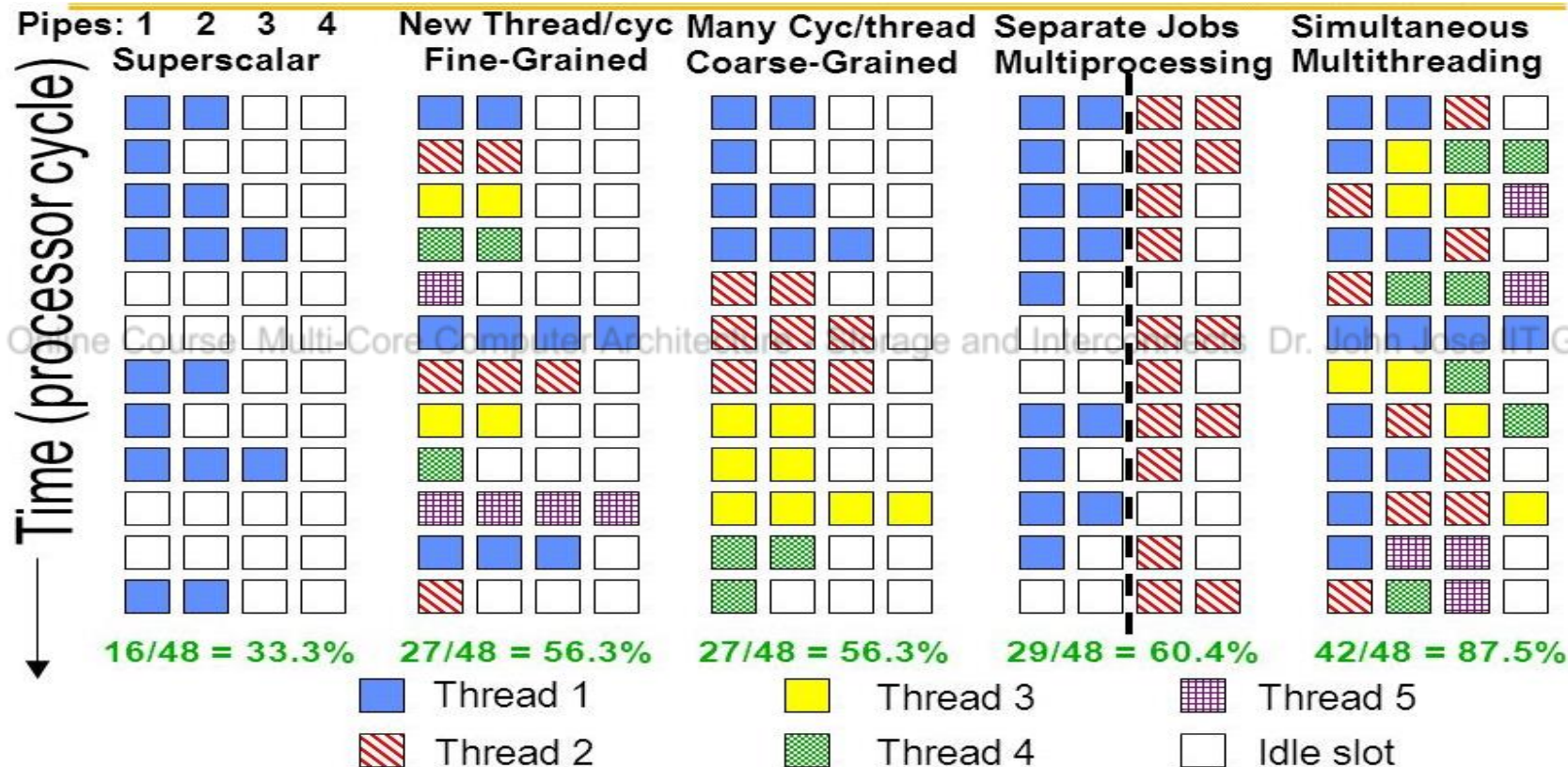
Hyperthreading /SMT

Hyperthreading



More than one
instruction stream
per slot

Comparison of Resource Utilization



**THANK
YOU!**

**FOR
YOUR
PRESENCE**

johnjose@iitg.ac.in

<http://www.iitg.ac.in/johnjose/>