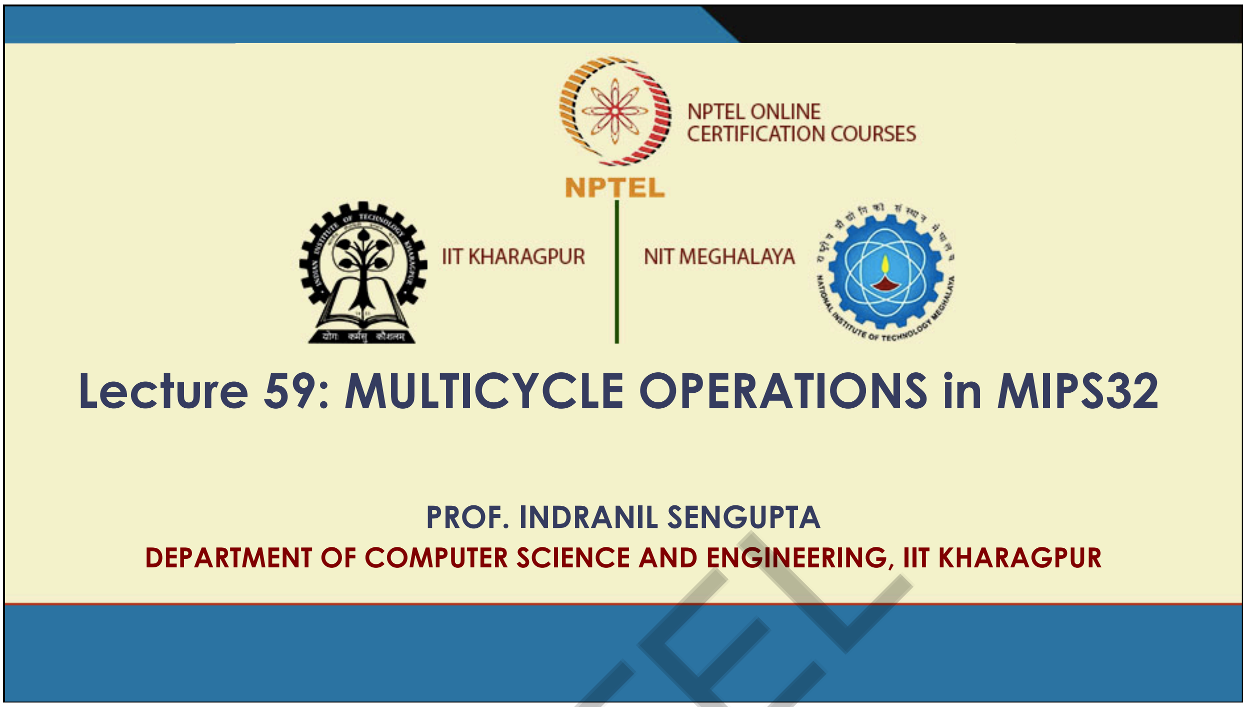


NPTEL ONLINE CERTIFICATION COURSES

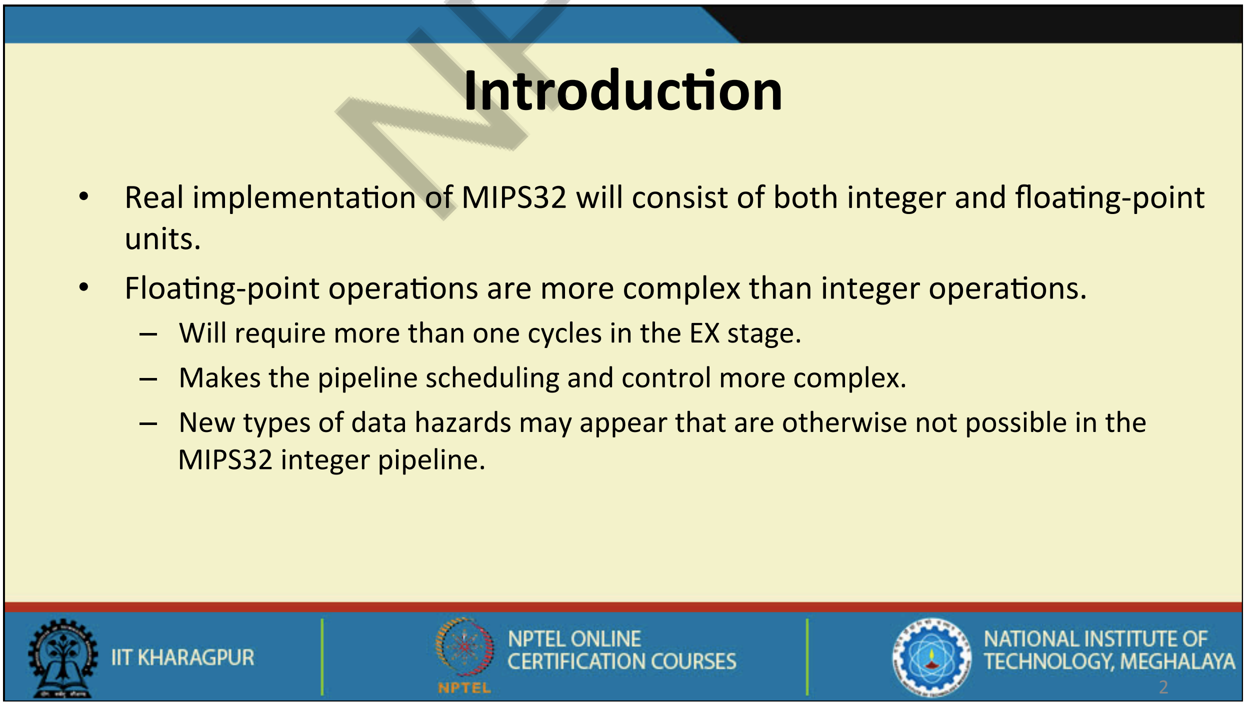
NPTEL

IIT KHARAGPUR

NIT MEGHALAYA

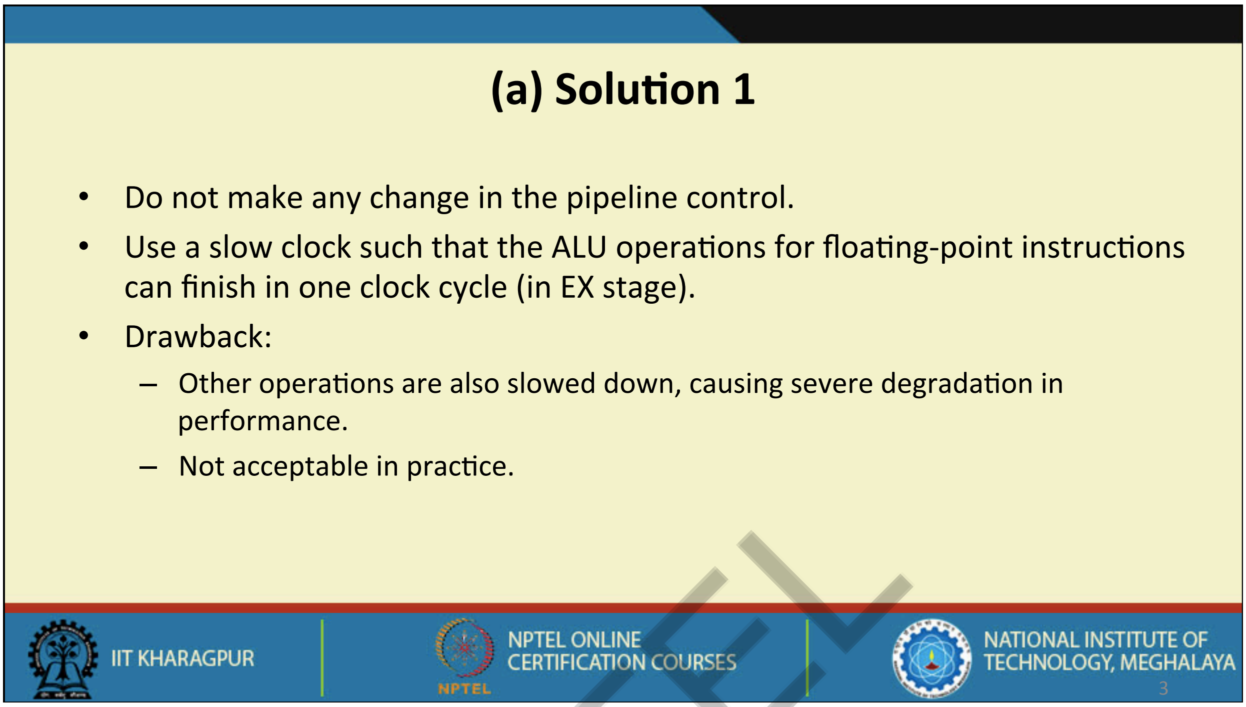
Lecture 59: MULTICYCLE OPERATIONS in MIPS32

PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, IIT KHARAGPUR



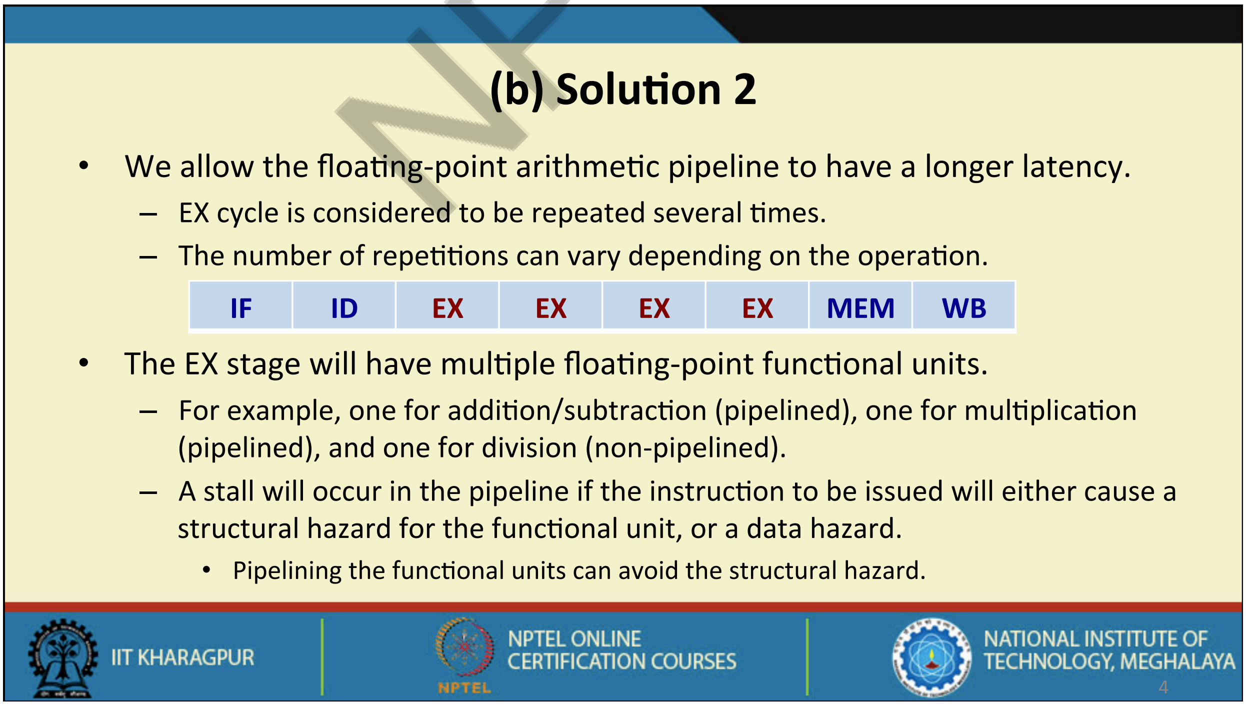
Introduction

- 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.



(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.



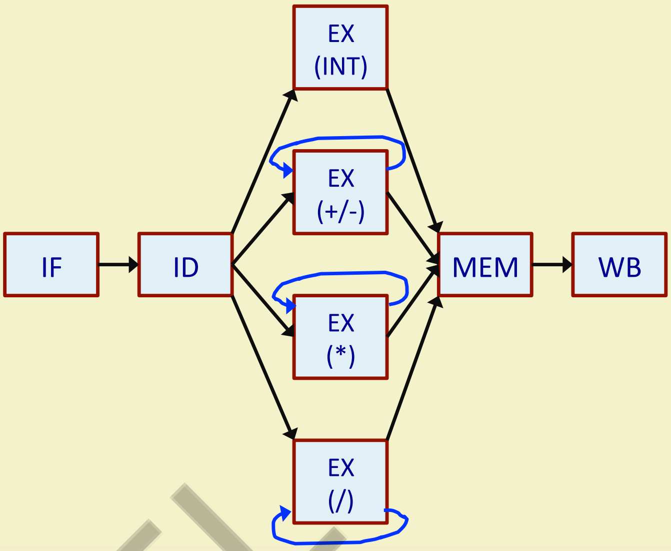
(b) Solution 2

- 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

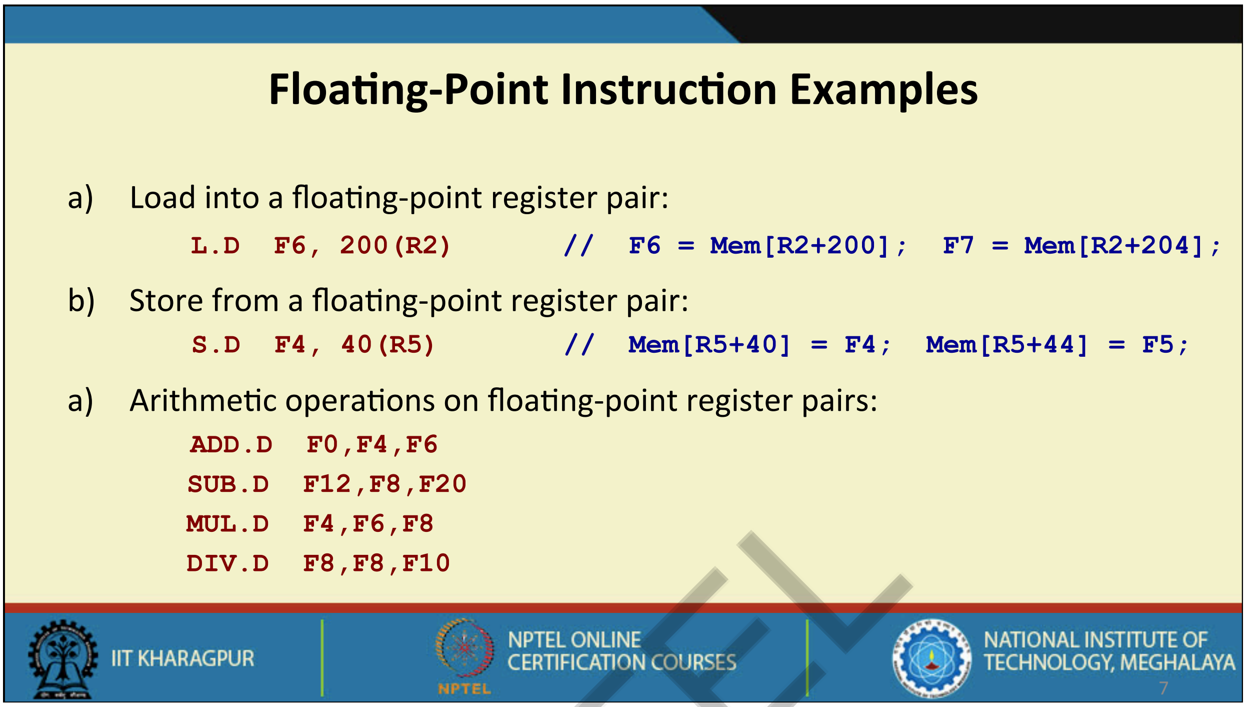
- The EX stage will have multiple floating-point functional units.
  - For example, one for addition/subtraction (pipelined), one for multiplication (pipelined), and one for division (non-pipelined).
  - A stall will occur in the pipeline if the instruction to be issued will either cause a structural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

- Consider that there are four functional units:
  a) Main integer unit that handles loads and stores, integer ALU operations, and branches.
  b) Floating-point adder / subtractor.
  c) Floating-point and integer multiplier.
  d) Floating-point and integer divider.



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.



Floating-Point Instruction Examples

a) Load into a floating-point register pair:

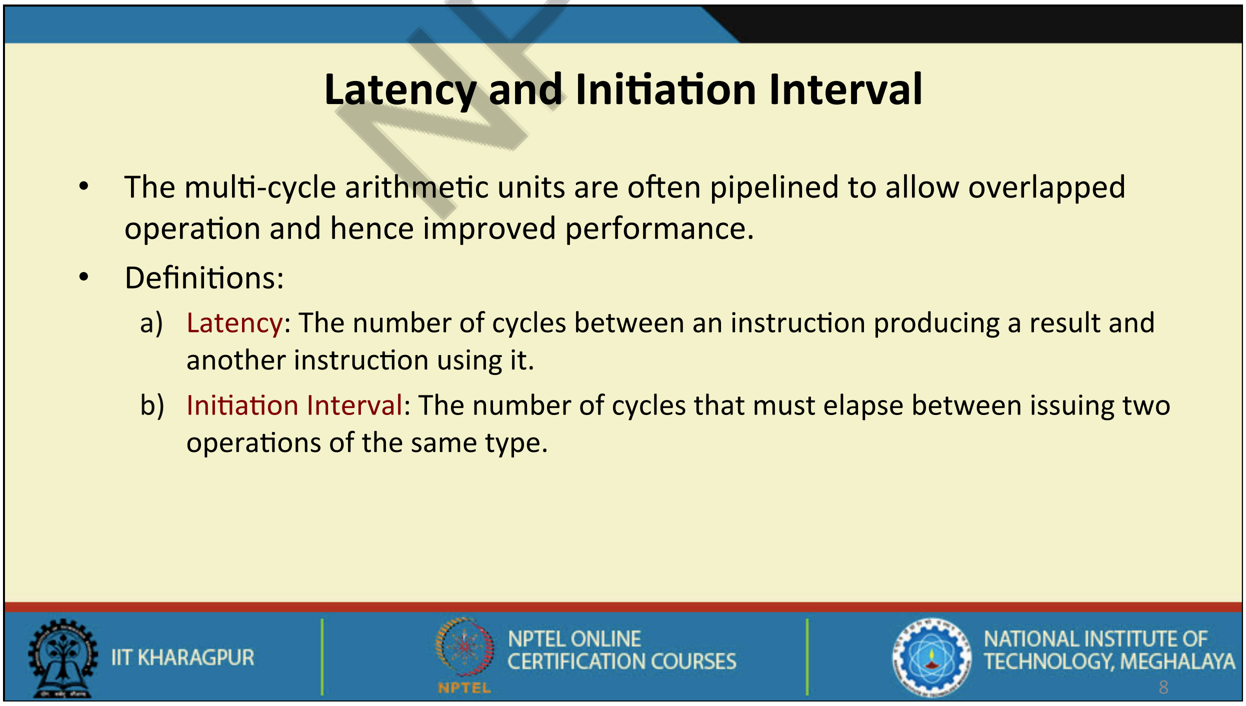
```
L.D  F6, 200(R2)      //  F6 = Mem[R2+200];  F7 = Mem[R2+204];
```

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
```



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.

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

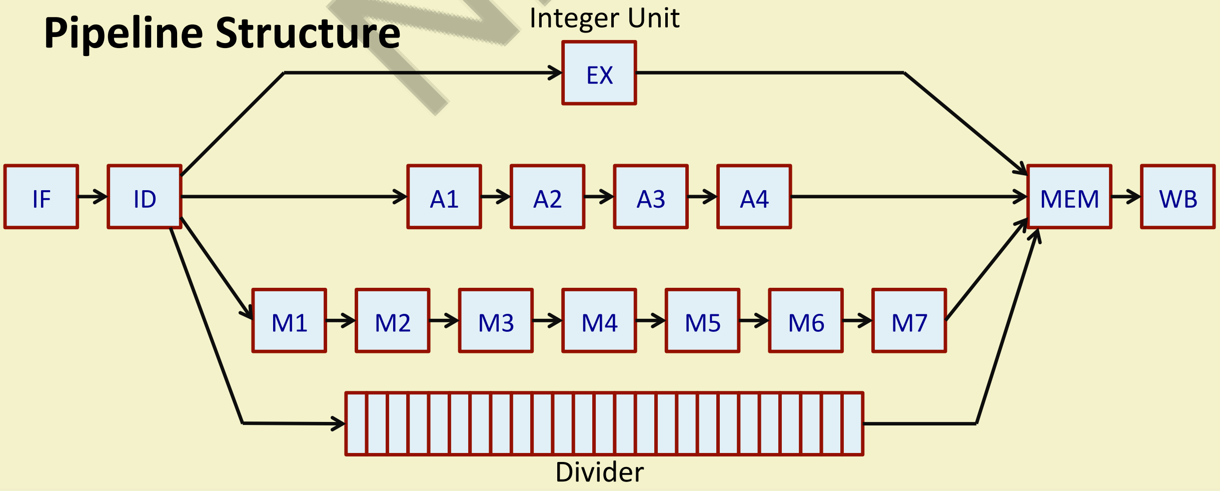
Assumptions on number of EX stages:

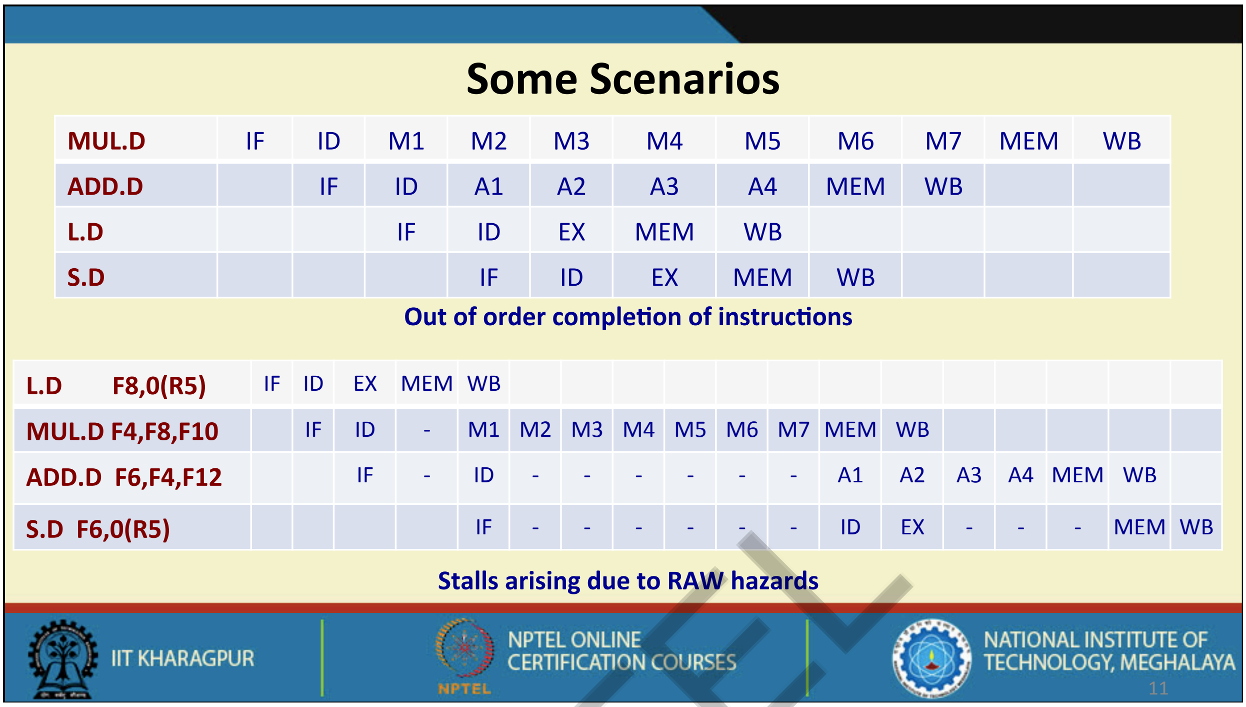
- FP add/subtract: 4
- FP multiply: 7
- FP divide: 1 (not pipelined)

It is possible to have up to:

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

MIPS32 Multi-cycle Pipeline Structure





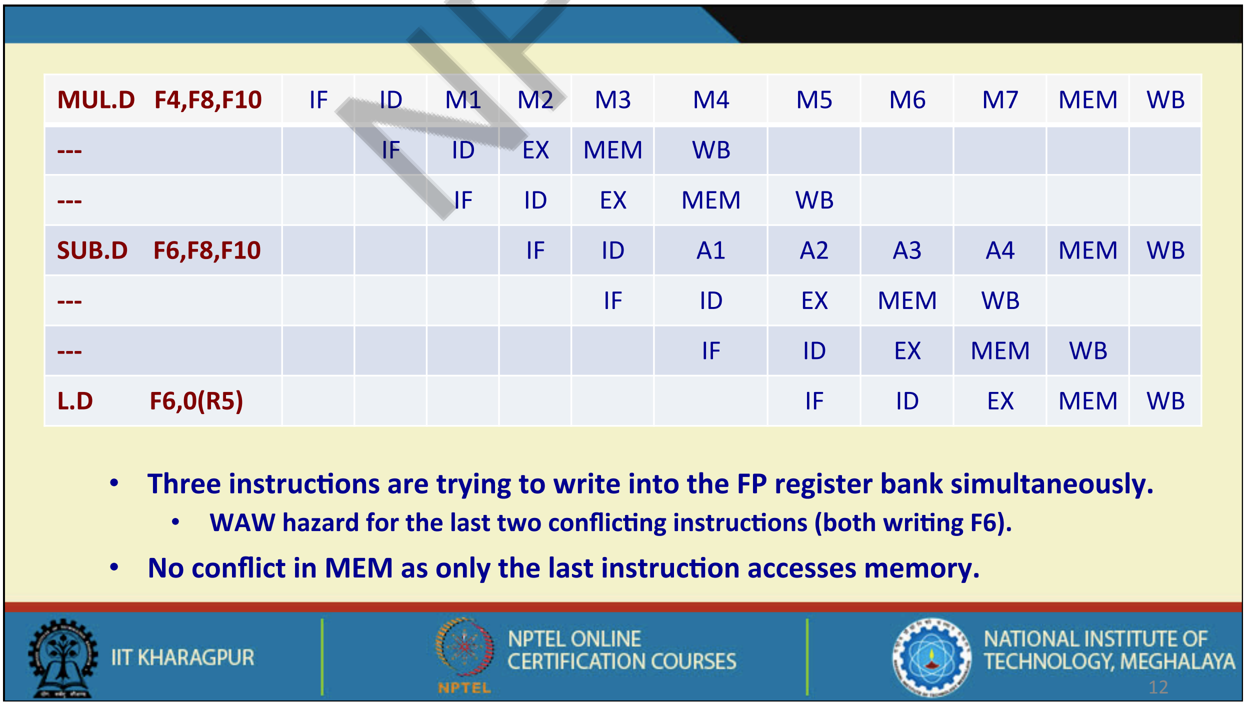
Some Scenarios

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

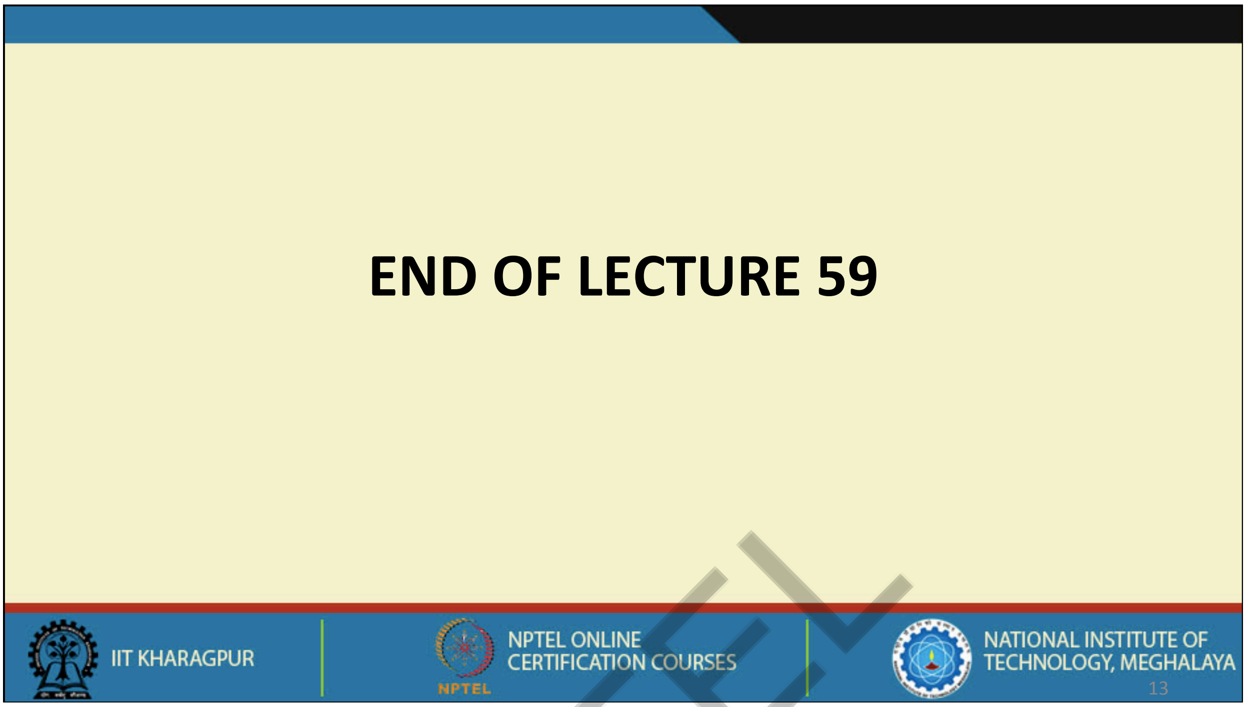
L.D F8,0(R5)	IF	ID	EX	MEM	WB													
MUL.D F4,F8,F10		IF	ID	-	M1	M2	M3	M4	M5	M6	M7	MEM	WB					
ADD.D F6,F4,F12			IF	-	ID	-	-	-	-	-	-	A1	A2	A3	A4	MEM	WB	
S.D F6,0(R5)					IF	-	-	-	-	-	-	ID	EX	-	-	-	MEM	WB

Stalls arising due to RAW hazards



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.



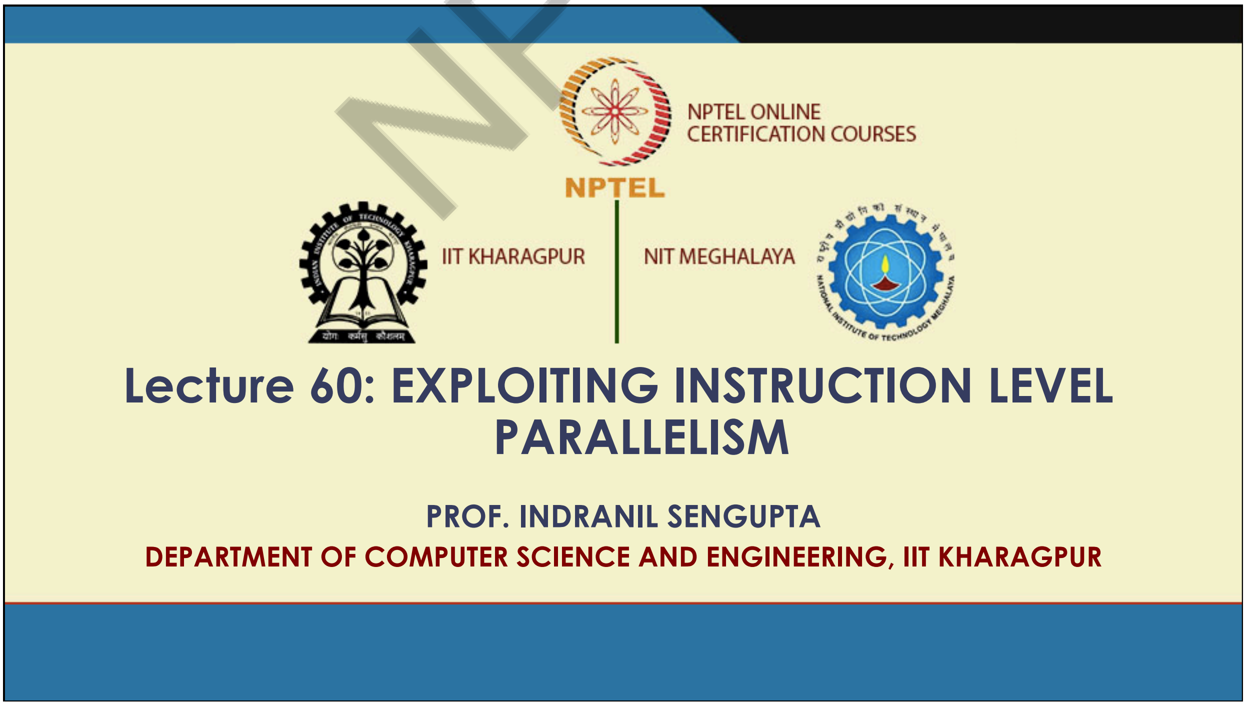
END OF LECTURE 59

IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

13



NPTEL ONLINE CERTIFICATION COURSES

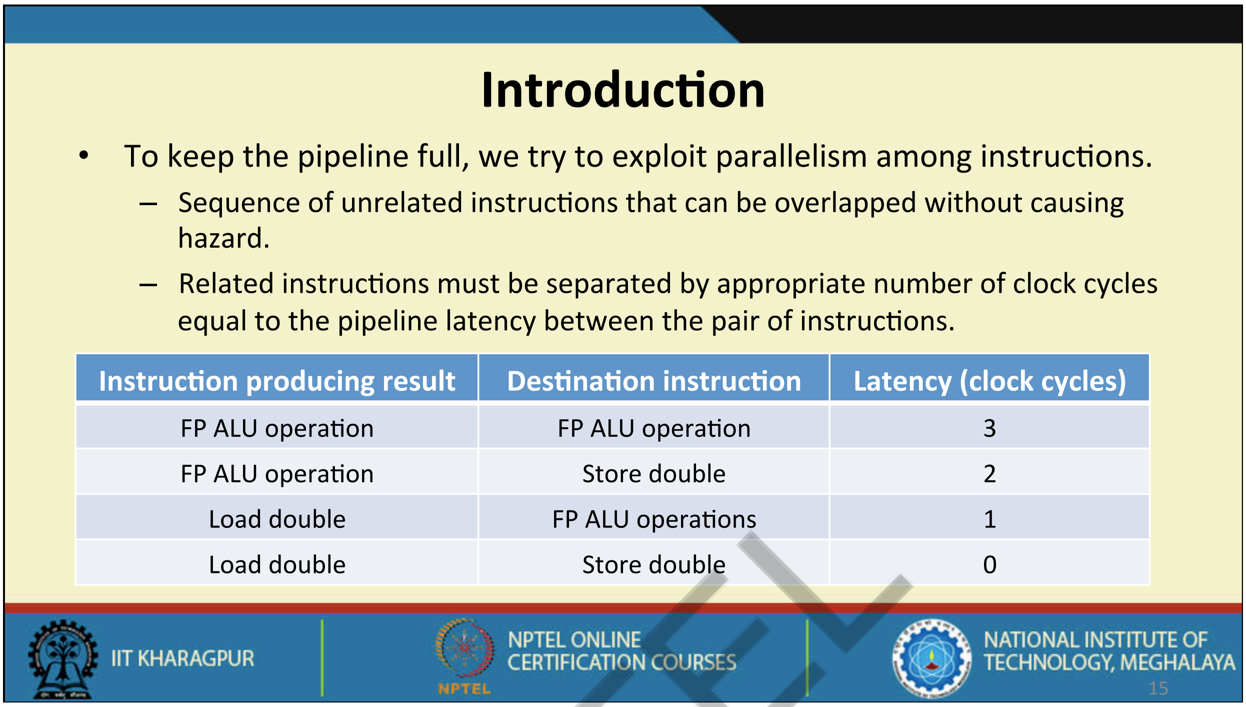
IIT KHARAGPUR

NIT MEGHALAYA

Lecture 60: EXPLOITING INSTRUCTION LEVEL PARALLELISM

PROF. INDRANIL SENGUPTA

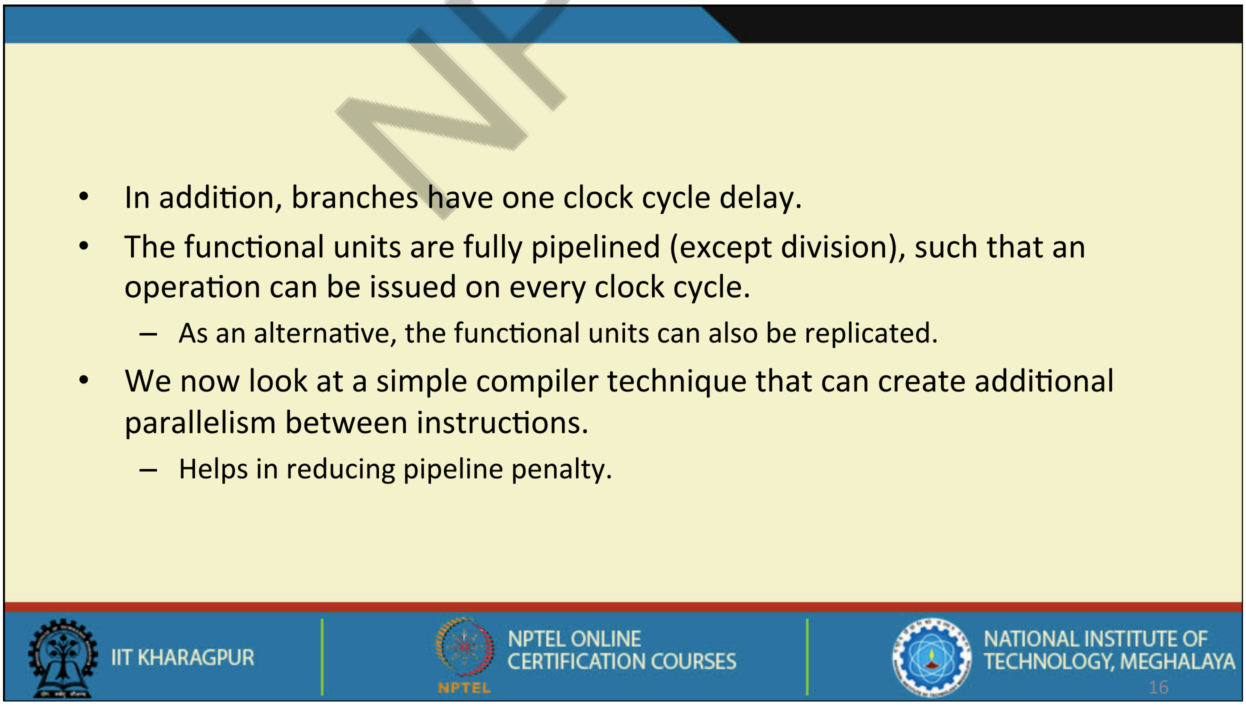
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, IIT KHARAGPUR



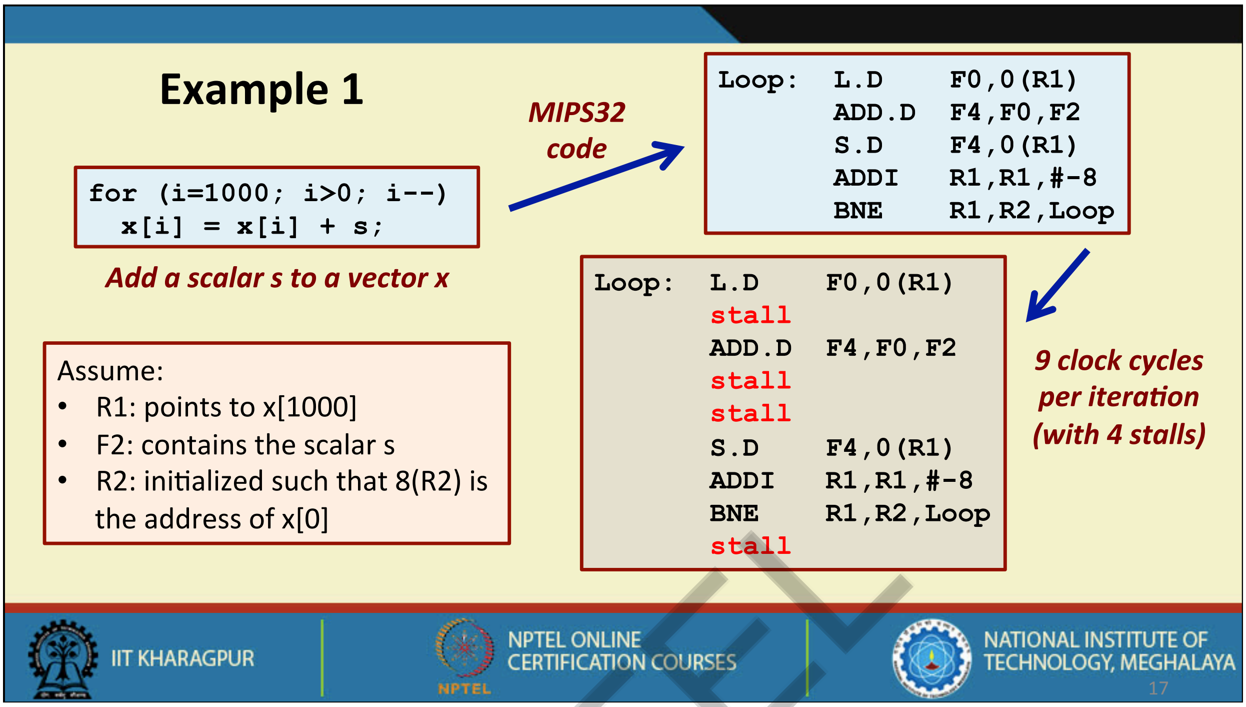
Introduction

- To keep the pipeline full, we try to exploit parallelism among instructions.
  - Sequence of unrelated instructions that can be overlapped without causing hazard.
  - Related instructions must be separated by appropriate number of clock cycles equal to the pipeline latency between the pair of instructions.

Instruction producing result	Destination instruction	Latency (clock cycles)
FP ALU operation	FP ALU operation	3
FP ALU operation	Store double	2
Load double	FP ALU operations	1
Load double	Store double	0



- In addition, branches have one clock cycle delay.
- The functional units are fully pipelined (except division), such that an operation can be issued on every clock cycle.
  - As an alternative, the functional units can also be replicated.
- We now look at a simple compiler technique that can create additional parallelism between instructions.
  - Helps in reducing pipeline penalty.



Example 1

```
for (i=1000; i>0; i--)
  x[i] = x[i] + s;
```

Add a scalar s to a vector x

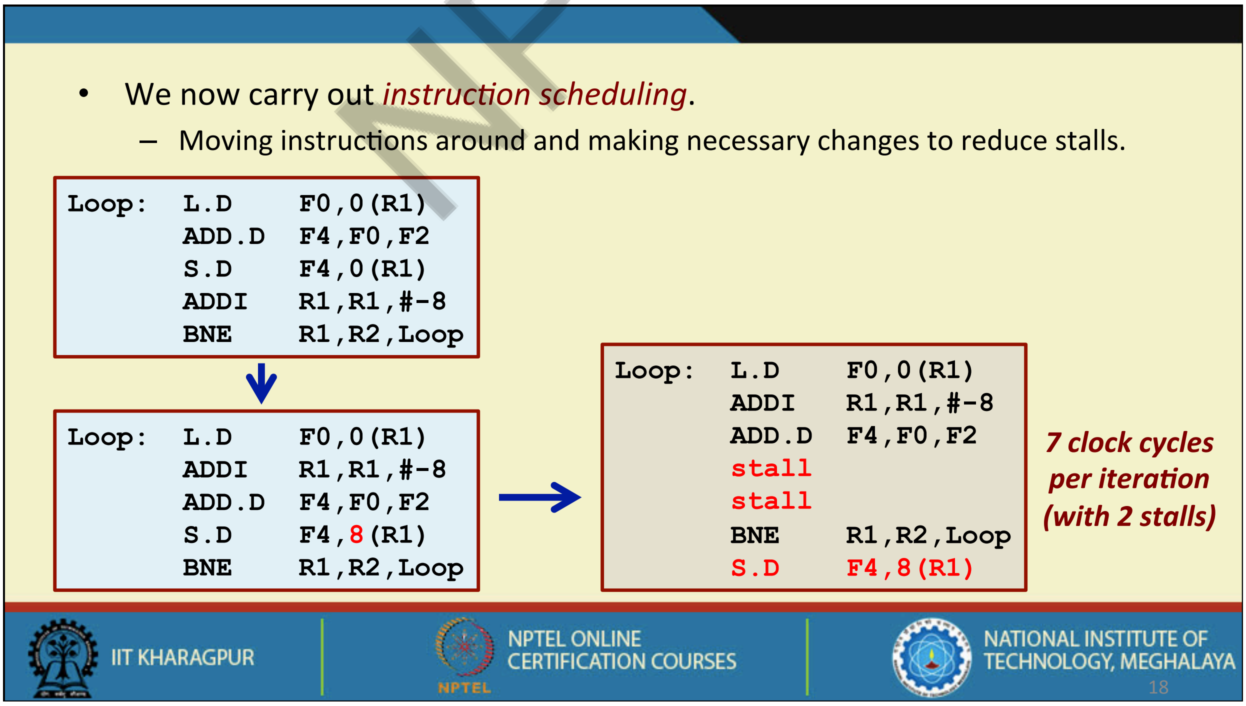
Assume:
- R1: points to x[1000]
- F2: contains the scalar s
- R2: initialized such that 8(R2) is the address of x[0]

MIPS32 code

```
Loop:  L.D     F0,0(R1)
       ADD.D   F4,F0,F2
       S.D     F4,0(R1)
       ADDI    R1,R1,#-8
       BNE     R1,R2,Loop
```

```
Loop:  L.D     F0,0(R1)
       stall
       ADD.D   F4,F0,F2
       stall
       stall
       S.D     F4,0(R1)
       ADDI    R1,R1,#-8
       BNE     R1,R2,Loop
       stall
```

9 clock cycles per iteration (with 4 stalls)



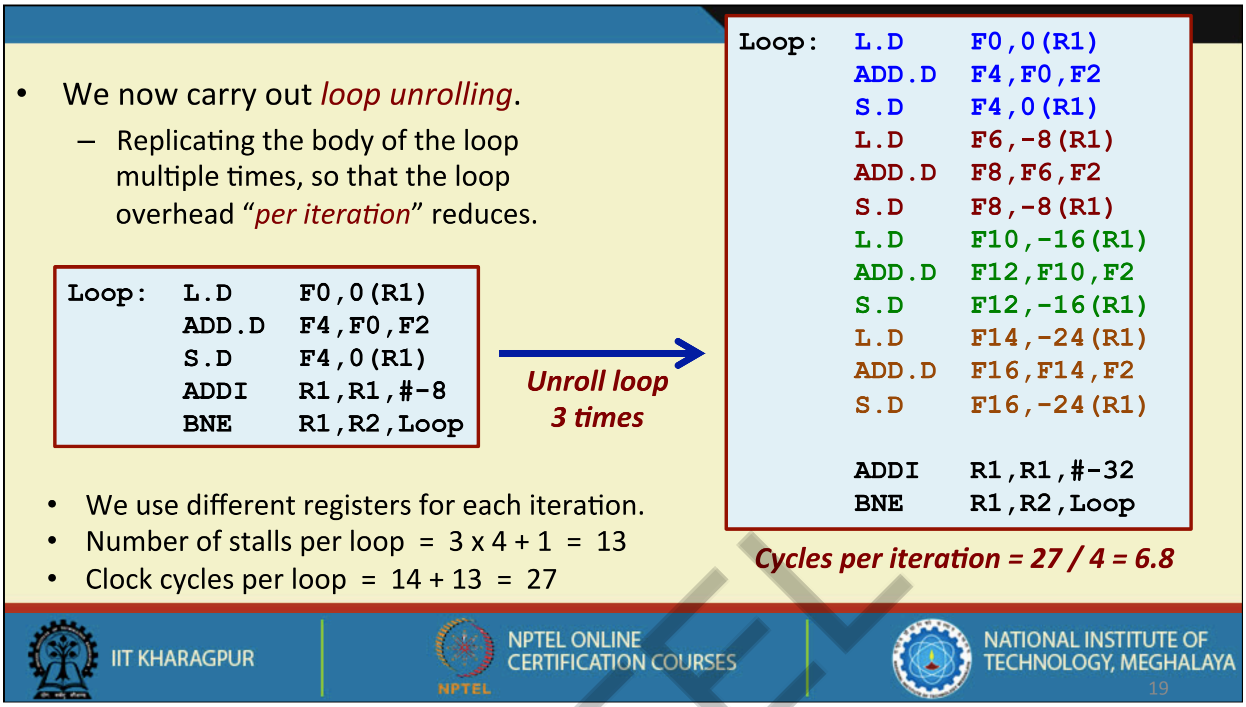
- We now carry out *instruction scheduling*.
  - Moving instructions around and making necessary changes to reduce stalls.

```
Loop:  L.D     F0,0(R1)
       ADD.D   F4,F0,F2
       S.D     F4,0(R1)
       ADDI    R1,R1,#-8
       BNE     R1,R2,Loop
```

```
Loop:  L.D     F0,0(R1)
       ADDI    R1,R1,#-8
       ADD.D   F4,F0,F2
       S.D     F4,8(R1)
       BNE     R1,R2,Loop
```

```
Loop:  L.D     F0,0(R1)
       ADDI    R1,R1,#-8
       ADD.D   F4,F0,F2
       stall
       stall
       BNE     R1,R2,Loop
       S.D     F4,8(R1)
```

7 clock cycles per iteration (with 2 stalls)



- We now carry out *loop unrolling*.
  - Replicating the body of the loop multiple times, so that the loop overhead "*per iteration*" reduces.

```
Loop:  L.D    F0,0(R1)
       ADD.D  F4,F0,F2
       S.D    F4,0(R1)
       ADDI   R1,R1,#-8
       BNE    R1,R2,Loop
```

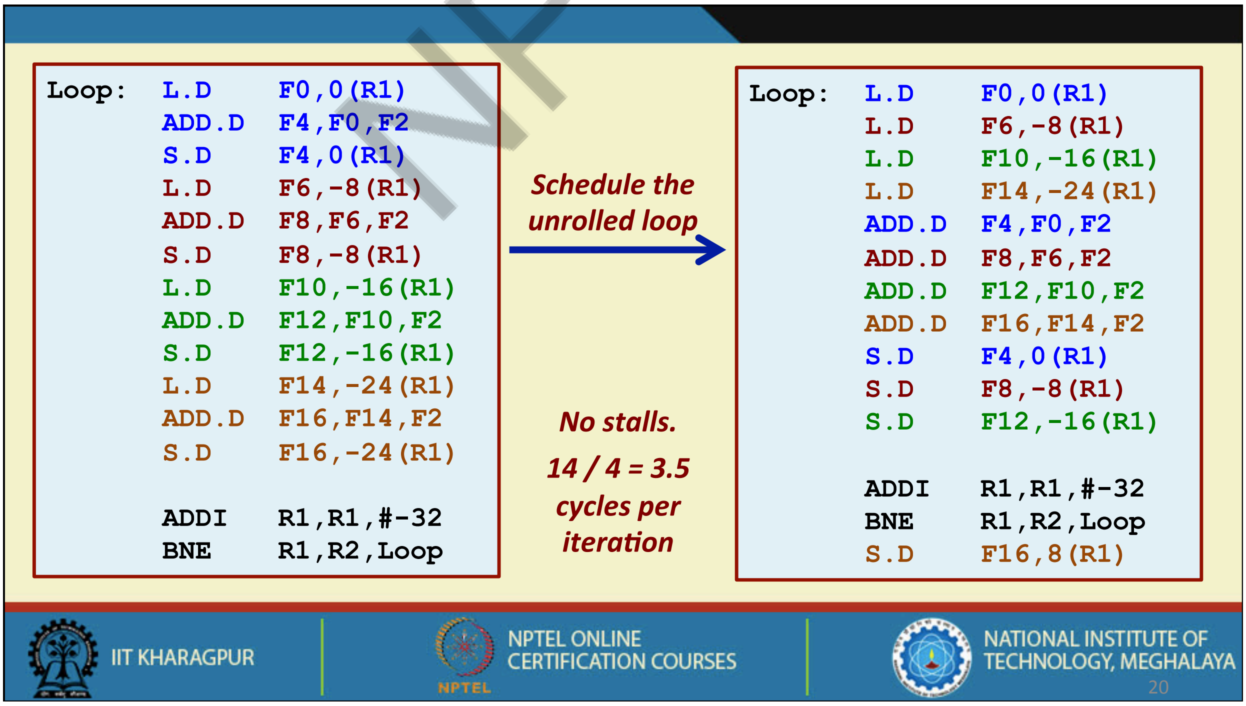
Unroll loop 3 times

```
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)

       ADDI   R1,R1,#-32
       BNE    R1,R2,Loop
```

- We use different registers for each iteration.
- Number of stalls per loop = 3 x 4 + 1 = 13
- Clock cycles per loop = 14 + 13 = 27

Cycles per iteration = 27 / 4 = 6.8



```
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)

       ADDI   R1,R1,#-32
       BNE    R1,R2,Loop
```

Schedule the 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)
       S.D    F12,-16(R1)

       ADDI   R1,R1,#-32
       BNE    R1,R2,Loop
       S.D    F16,8(R1)
```

No stalls.

14 / 4 = 3.5 cycles per iteration

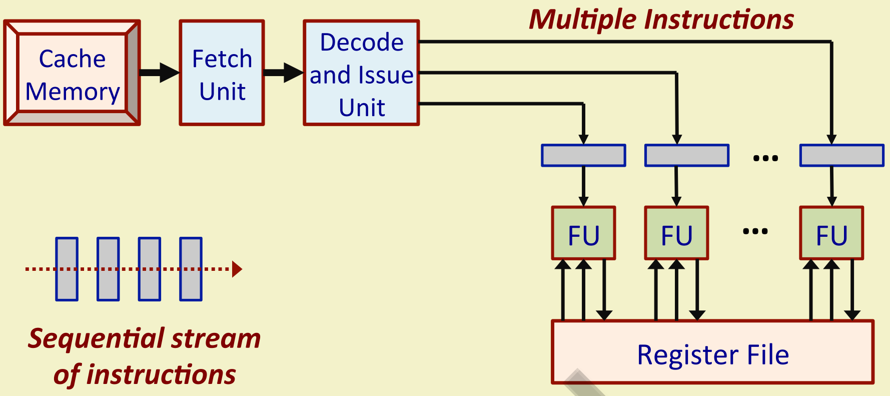
Loop Unrolling :: Summary

- Loop unrolling can expose more parallelism in instructions that can be scheduled.
  - Effective way of improving pipeline performance.
- Can be used to lower the CPI in architectures where more than one instructions can be issued per cycle.
  - a) Superscalar architecture
  - b) Very Long Instruction Word (VLIW) architecture

A Superscalar Version of MIPS32

- Superscalar Machines:
  - Machines that can issue multiple independent instructions per clock cycle when they are properly scheduled by the compiler.
  - Can result in a CPI of less than 1.
- How does it work?
  - The hardware can issue a small number (say, 2 to 4) of independent instructions in every clock cycle.
  - The hardware checks for conflicts between instructions.
  - If the instructions are dependent, then only the first instruction in the sequence will be issued.

Superscalar Architecture Schematic

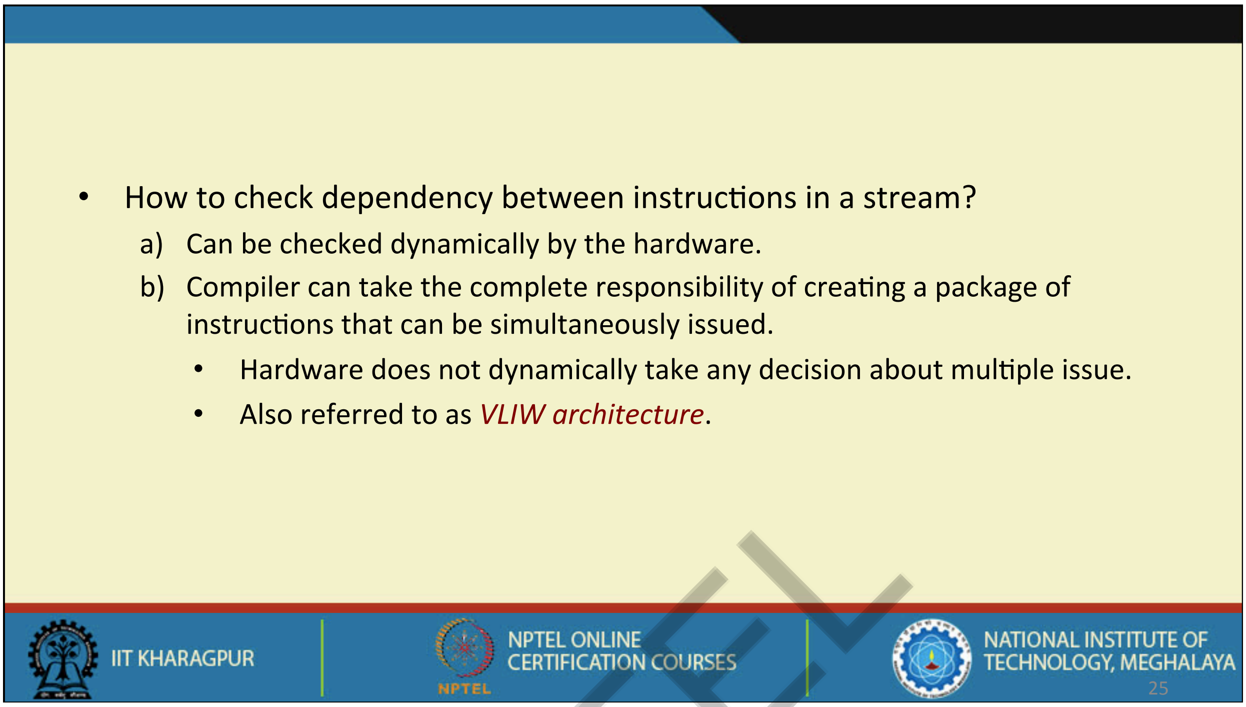


Example

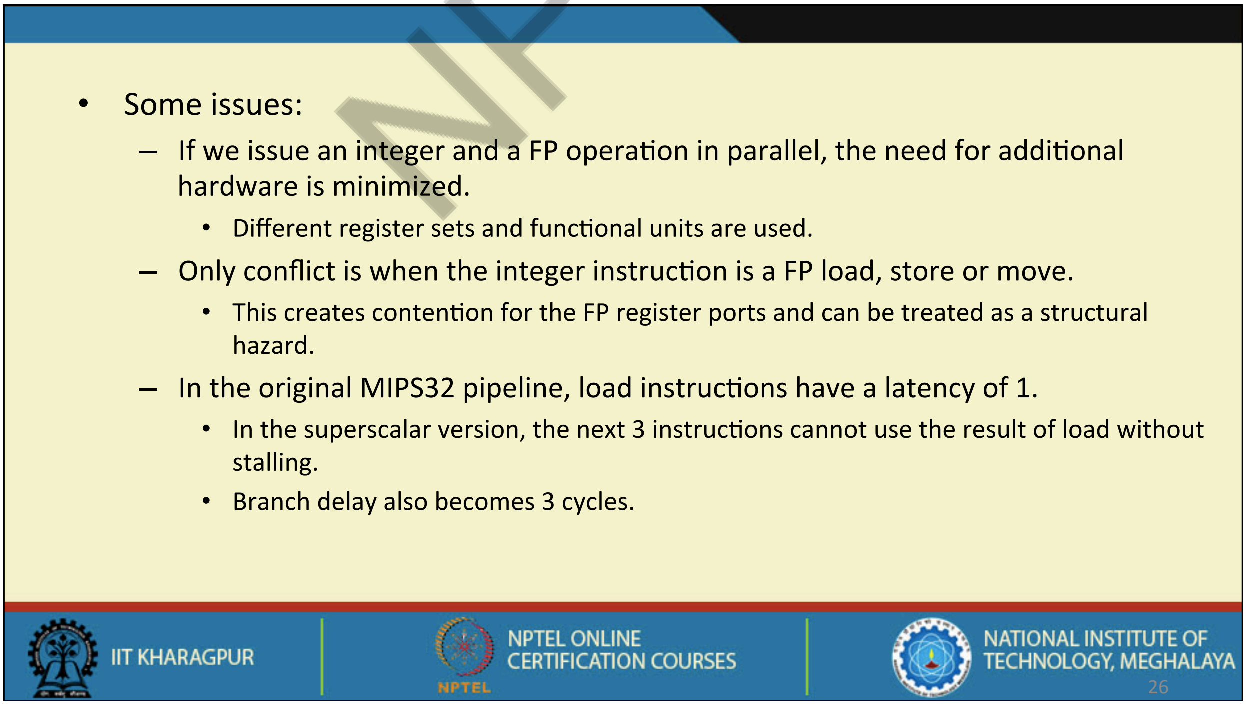
- Suppose two instructions can be issued every clock cycle.
  a) One can be a load, store, branch or integer ALU operation.
  b) The other can be any floating-point operation.

Integer instr.	IF	ID	EX	MEM	WB		
FP instr.	IF	ID	EX	MEM	WB		
Integer instr.		IF	ID	EX	MEM	WB	
FP instr.		IF	ID	EX	MEM	WB	
Integer instr.			IF	ID	EX	MEM	WB
FP instr.			IF	ID	EX	MEM	WB

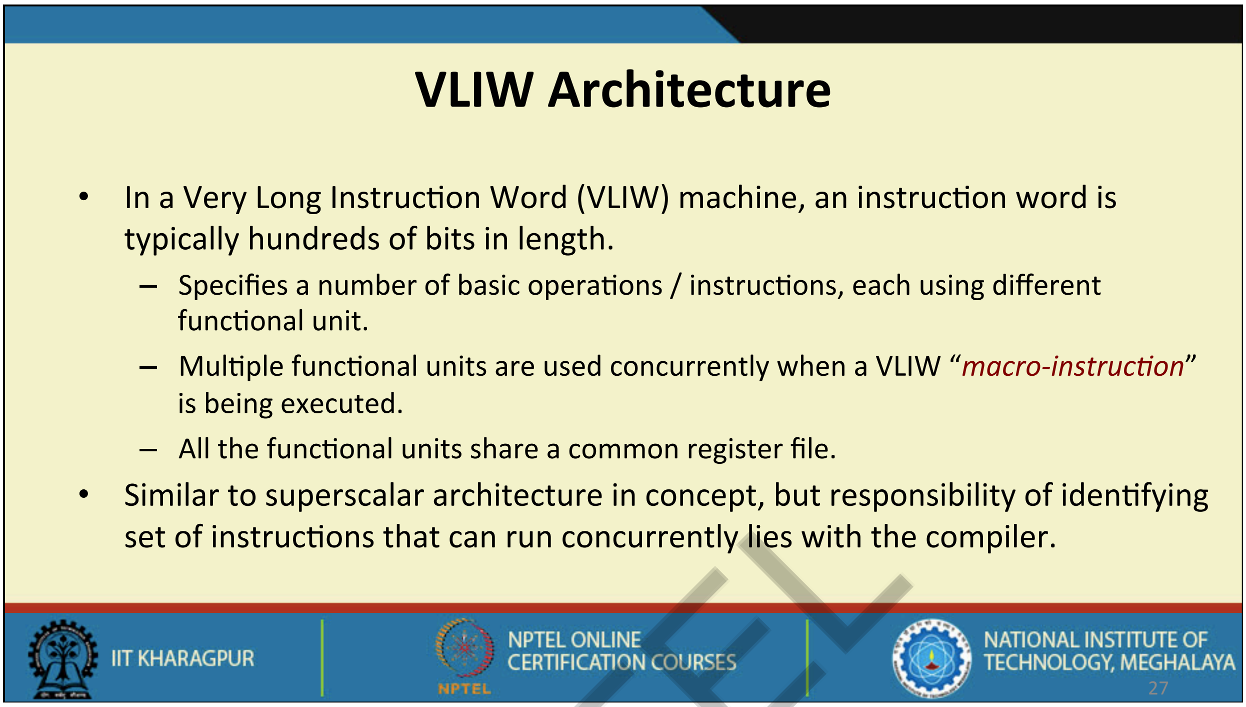
- **Used only for illustration.**
- **We have not shown how FP operations extend the EX cycle.**



- How to check dependency between instructions in a stream?
  a) Can be checked dynamically by the hardware.
  b) Compiler can take the complete responsibility of creating a package of instructions that can be simultaneously issued.
    - Hardware does not dynamically take any decision about multiple issue.
    - Also referred to as *VLIW architecture*.

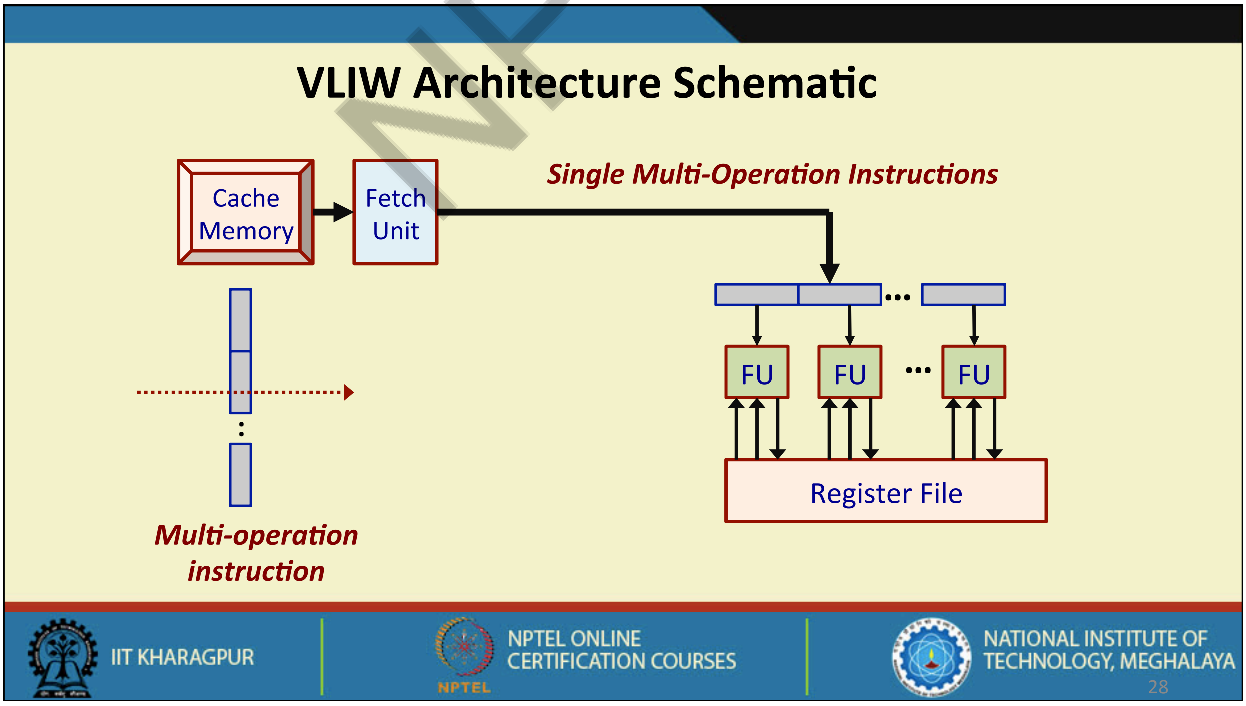


- Some issues:
  - If we issue an integer and a FP operation in parallel, the need for additional hardware is minimized.
    - Different register sets and functional units are used.
  - Only conflict is when the integer instruction is a FP load, store or move.
    - This creates contention for the FP register ports and can be treated as a structural hazard.
  - In the original MIPS32 pipeline, load instructions have a latency of 1.
    - In the superscalar version, the next 3 instructions cannot use the result of load without stalling.
    - Branch delay also becomes 3 cycles.



VLIW Architecture

- In a Very Long Instruction Word (VLIW) machine, an instruction word is typically hundreds of bits in length.
  - Specifies a number of basic operations / instructions, each using different functional unit.
  - Multiple functional units are used concurrently when a VLIW “*macro-instruction*” is being executed.
  - All the functional units share a common register file.
- Similar to superscalar architecture in concept, but responsibility of identifying set of instructions that can run concurrently lies with the compiler.



VLIW Architecture Schematic

Cache Memory

Fetch Unit

Single Multi-Operation Instructions

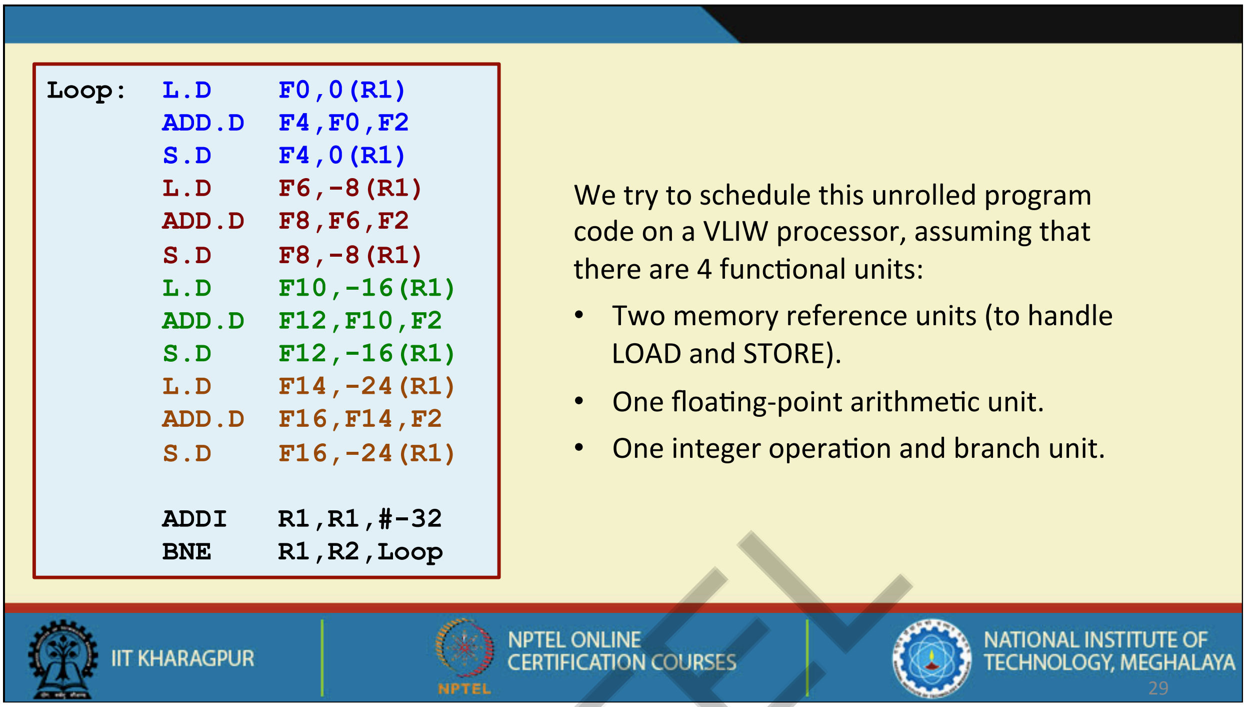
FU

FU

FU

Register File

Multi-operation instruction

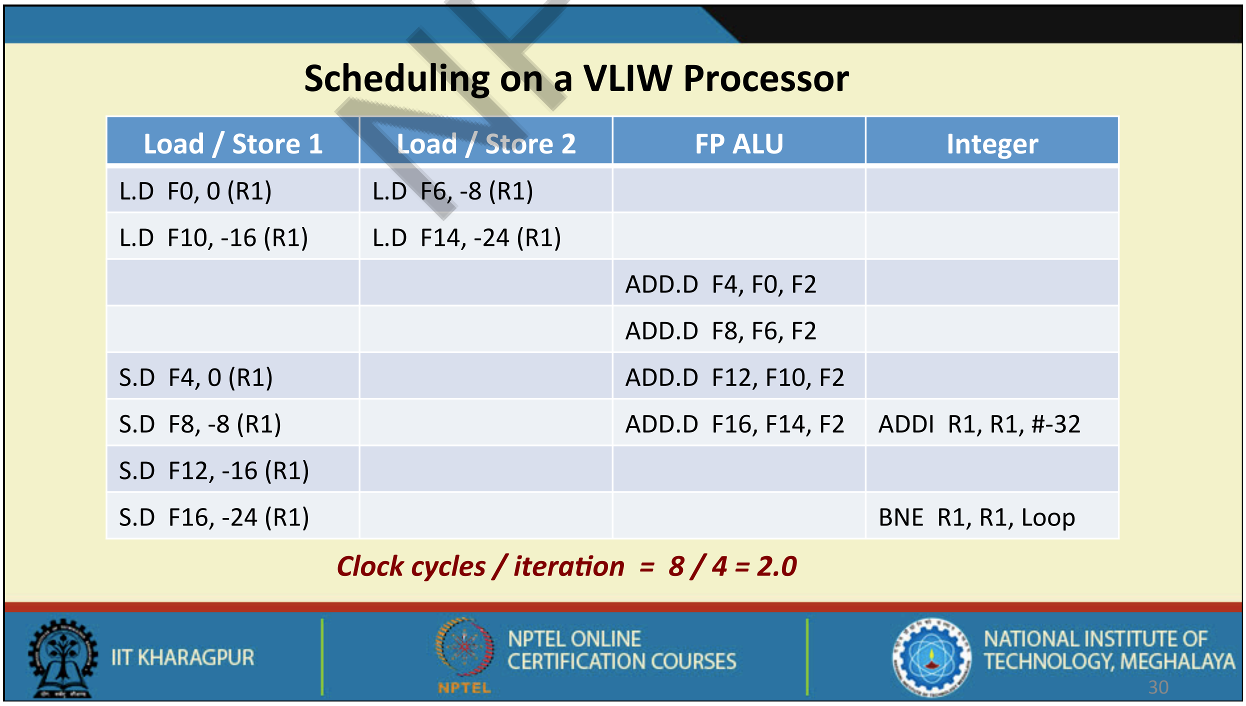


```
Loop:  L.D     F0,0(R1)
       ADD.D   F4,F0,F2
       S.D     F4,0(R1)
       L.D     F6,-8(R1)
       ADD.D   F8,F6,F2
       S.D     F8,-8(R1)
       L.D     F10,-16(R1)
       ADD.D   F12,F10,F2
       S.D     F12,-16(R1)
       L.D     F14,-24(R1)
       ADD.D   F16,F14,F2
       S.D     F16,-24(R1)

       ADDI    R1,R1,#-32
       BNE     R1,R2,Loop
```

We try to schedule this unrolled program code on a VLIW processor, assuming that there are 4 functional units:

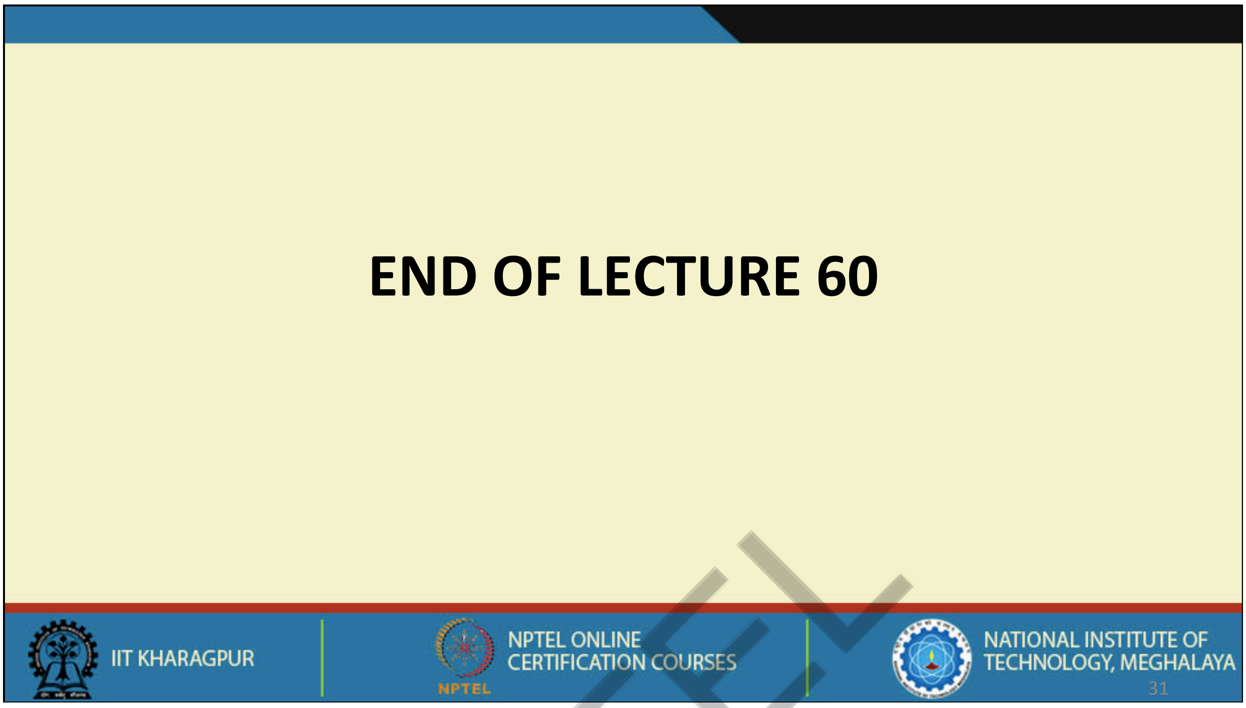
- Two memory reference units (to handle LOAD and STORE).
- One floating-point arithmetic unit.
- One integer operation and branch unit.



Scheduling on a VLIW Processor

Load / Store 1	Load / Store 2	FP ALU	Integer
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	
S.D F4, 0 (R1)		ADD.D F12, F10, F2	
S.D F8, -8 (R1)		ADD.D F16, F14, F2	ADDI R1, R1, #-32
S.D F12, -16 (R1)			
S.D F16, -24 (R1)			BNE R1, R1, Loop

Clock cycles / iteration = 8 / 4 = 2.0



END OF LECTURE 60

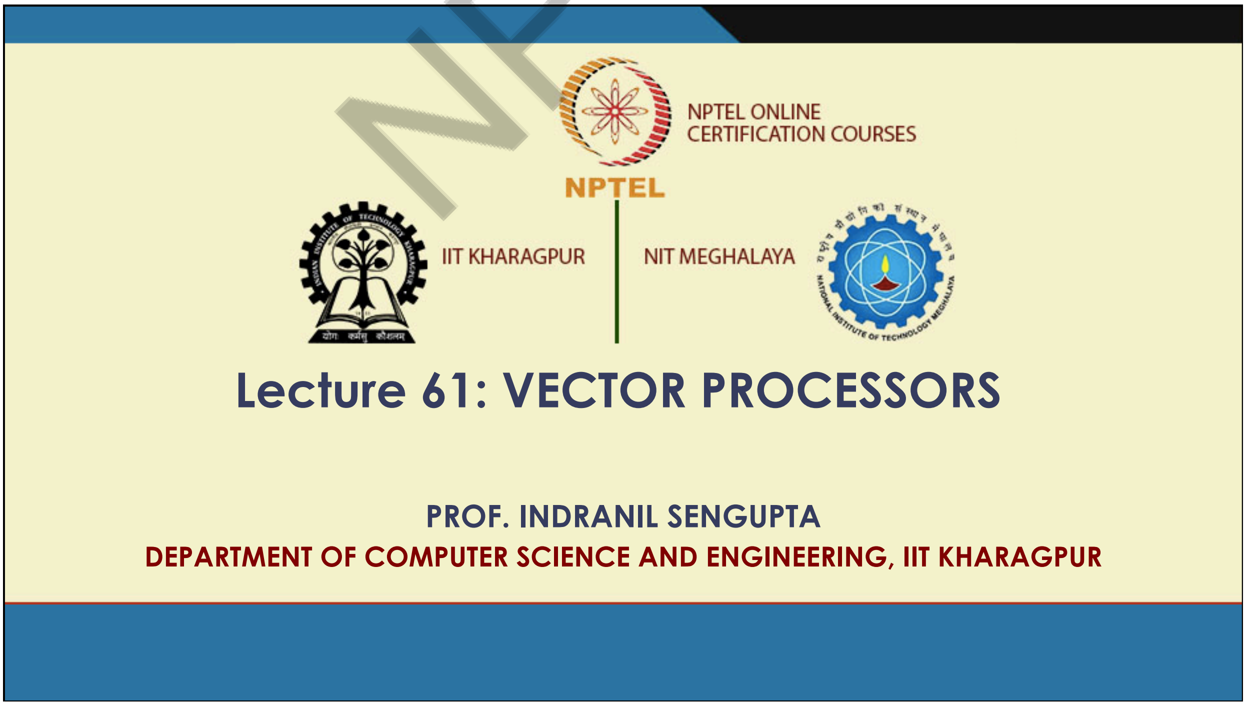
IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

NATIONAL INSTITUTE OF
TECHNOLOGY, MEGHALAYA

31



NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

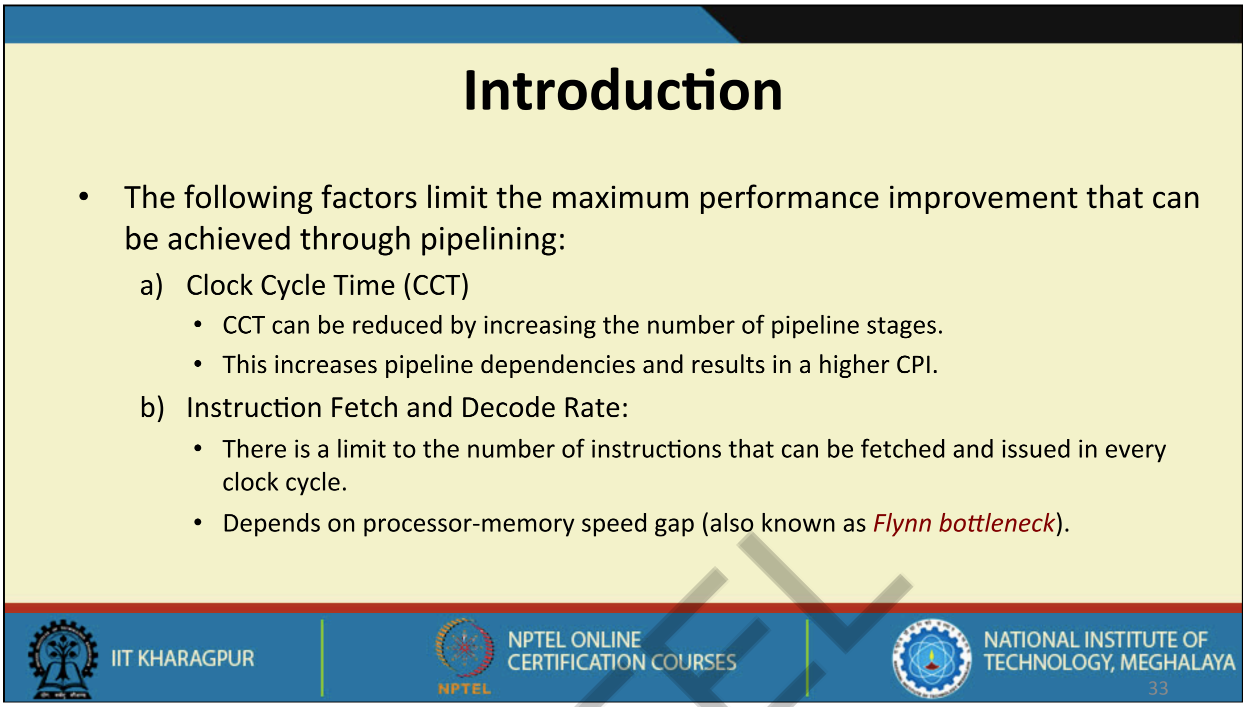
IIT KHARAGPUR

NIT MEGHALAYA

Lecture 61: VECTOR PROCESSORS

PROF. INDRANIL SENGUPTA

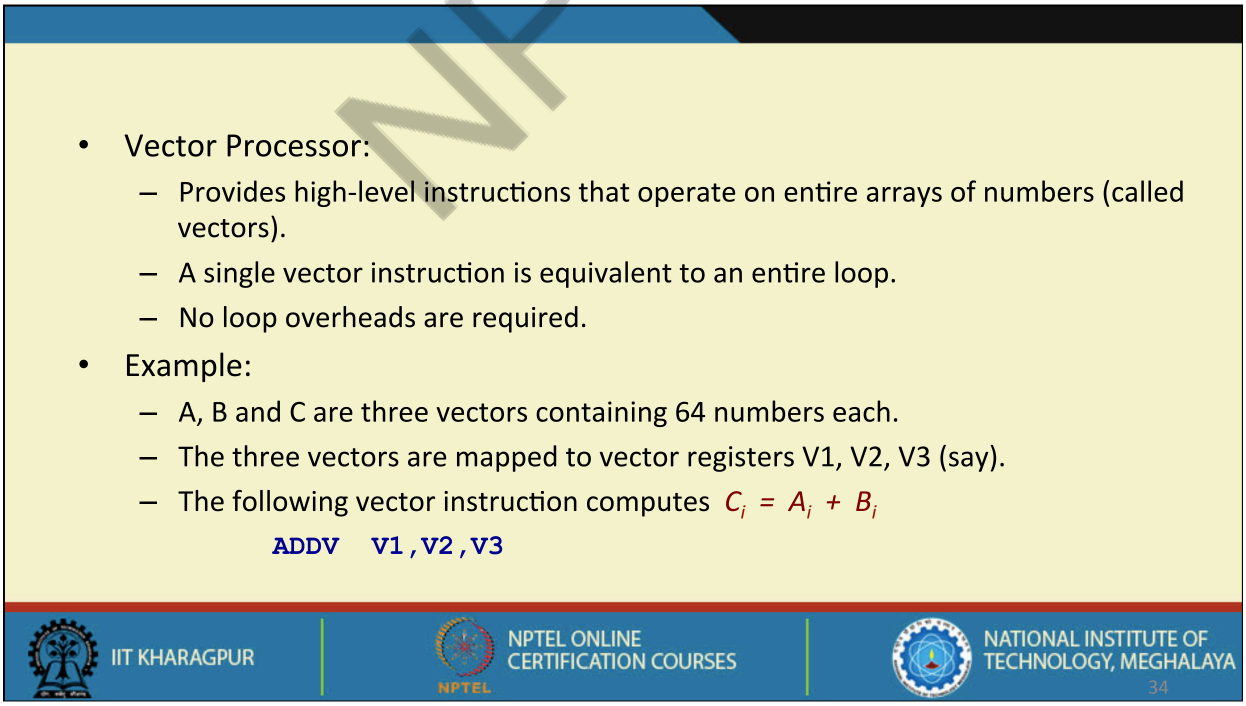
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, IIT KHARAGPUR



Introduction

- The following factors limit the maximum performance improvement that can be achieved through pipelining:
  a) Clock Cycle Time (CCT)
    - CCT can be reduced by increasing the number of pipeline stages.
    - This increases pipeline dependencies and results in a higher CPI.

  b) Instruction Fetch and Decode Rate:
    - There is a limit to the number of instructions that can be fetched and issued in every clock cycle.
    - Depends on processor-memory speed gap (also known as *Flynn bottleneck*).

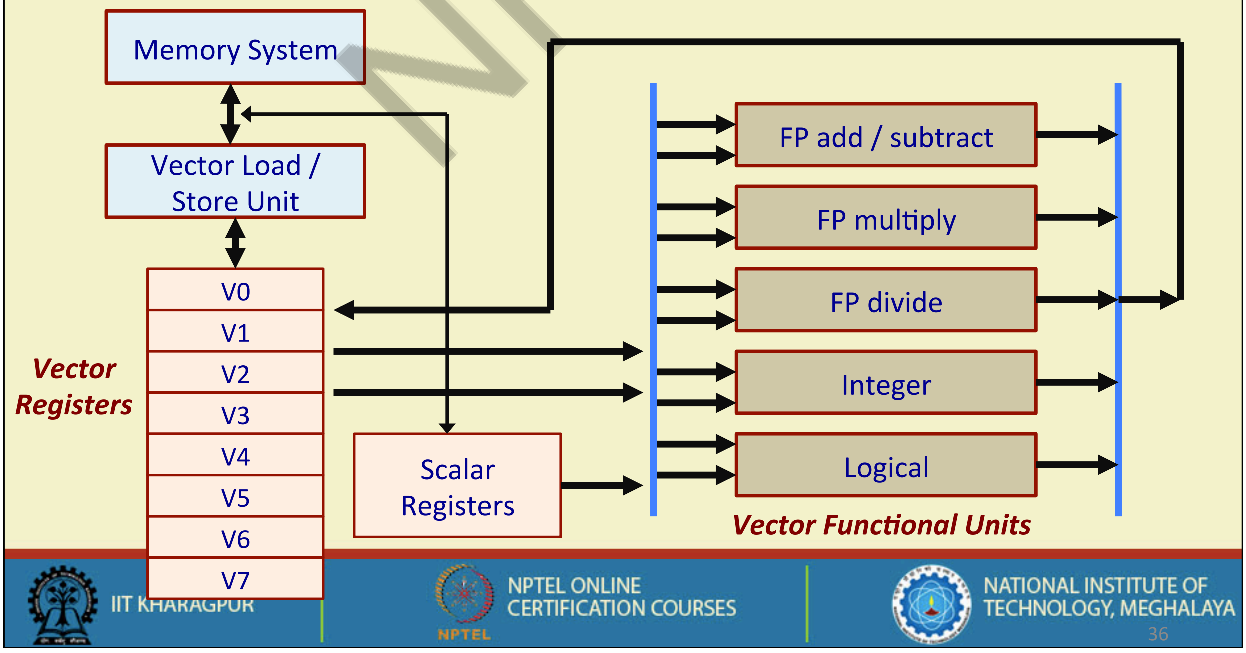


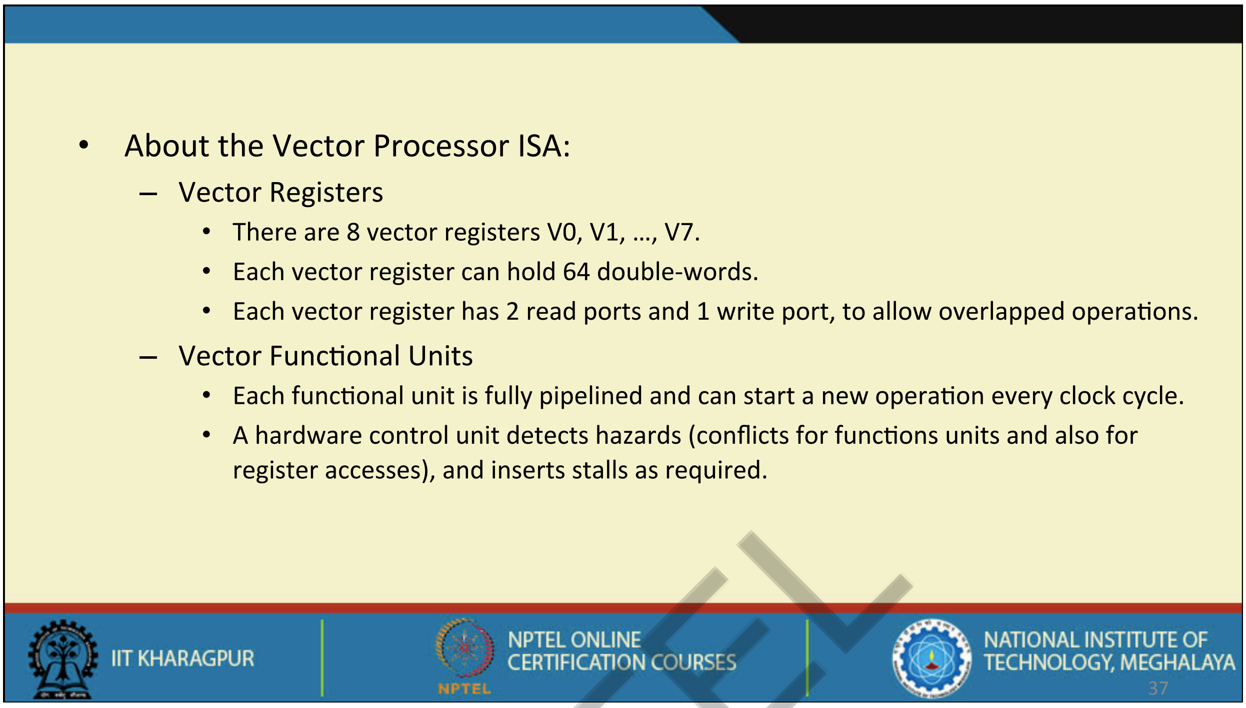
- Vector Processor:
  - Provides high-level instructions that operate on entire arrays of numbers (called vectors).
  - A single vector instruction is equivalent to an entire loop.
  - No loop overheads are required.
- Example:
  - A, B and C are three vectors containing 64 numbers each.
  - The three vectors are mapped to vector registers V1, V2, V3 (say).
  - The following vector instruction computes $C_i = A_i + B_i$

```
ADDV  V1,V2,V3
```

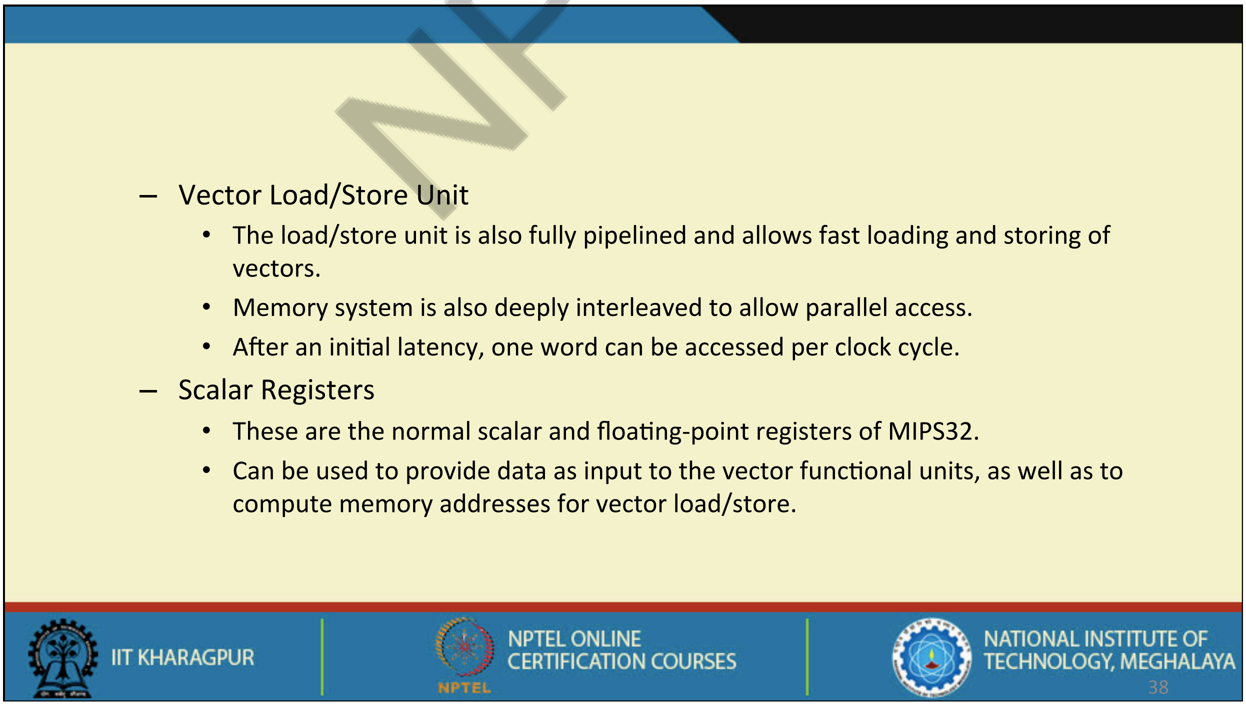
Basic Vector Processor Architecture

- A vector processor typically consists of an ordinary pipelined scalar unit plus a vector unit.
- All functional units within the vector unit are deeply pipelined, resulting in a shorter clock cycle time C.
  - Deep pipelining on vectors do not result in hazards, since every computation is independent of the others.
- We shall illustrate some concepts based on a hypothetical vector processor that is based on the MIPS32 architecture.

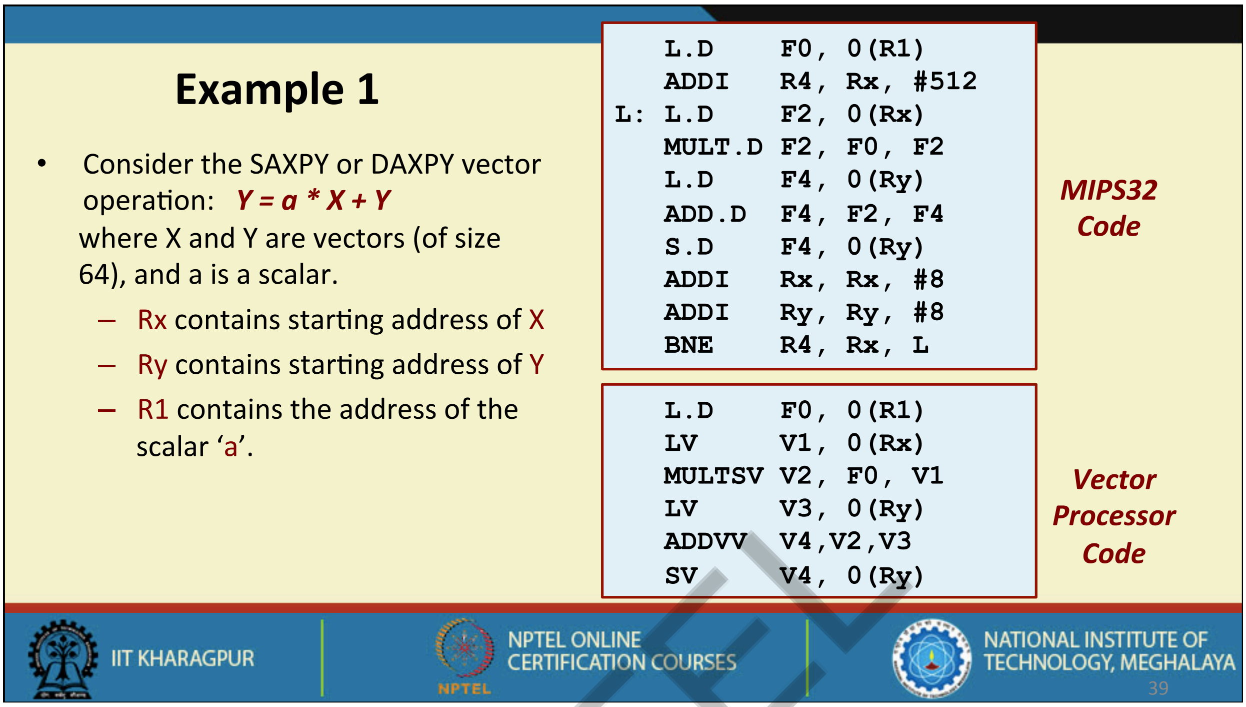




- About the Vector Processor ISA:
  - – Vector Registers
    - There are 8 vector registers V0, V1, …, V7.
    - Each vector register can hold 64 double-words.
    - Each vector register has 2 read ports and 1 write port, to allow overlapped operations.
  - – Vector Functional Units
    - Each functional unit is fully pipelined and can start a new operation every clock cycle.
    - A hardware control unit detects hazards (conflicts for functions units and also for register accesses), and inserts stalls as required.



- – Vector Load/Store Unit
  - The load/store unit is also fully pipelined and allows fast loading and storing of vectors.
  - Memory system is also deeply interleaved to allow parallel access.
  - After an initial latency, one word can be accessed per clock cycle.
- – Scalar Registers
  - These are the normal scalar and floating-point registers of MIPS32.
  - Can be used to provide data as input to the vector functional units, as well as to compute memory addresses for vector load/store.



Example 1

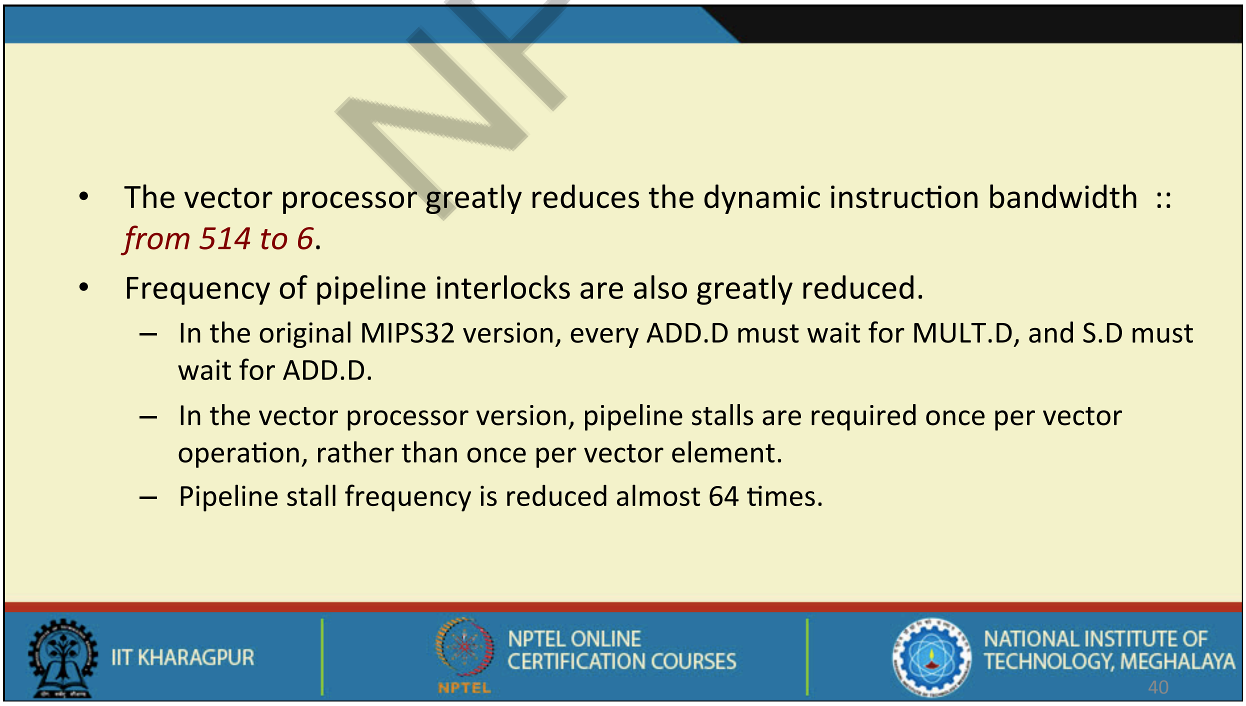
- Consider the SAXPY or DAXPY vector operation: ***Y = a * X + Y*** where X and Y are vectors (of size 64), and a is a scalar.
  - Rx contains starting address of X
  - Ry contains starting address of Y
  - R1 contains the address of the scalar ‘a’.

```
    L.D     F0, 0(R1)
    ADDI    R4, Rx, #512
L:  L.D     F2, 0(Rx)
    MULT.D  F2, F0, F2
    L.D     F4, 0(Ry)
    ADD.D   F4, F2, F4
    S.D     F4, 0(Ry)
    ADDI    Rx, Rx, #8
    ADDI    Ry, Ry, #8
    BNE     R4, Rx, L
```

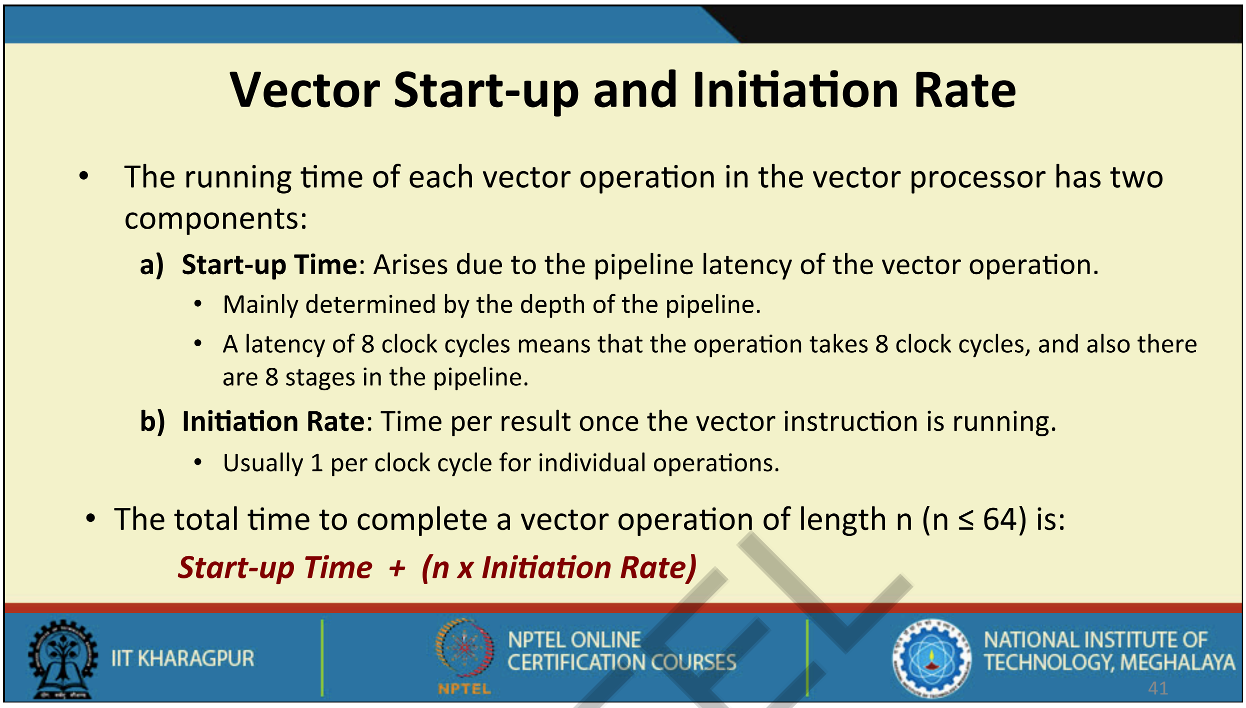
MIPS32 Code

```
    L.D     F0, 0(R1)
    LV      V1, 0(Rx)
    MULTSV  V2, F0, V1
    LV      V3, 0(Ry)
    ADDVV   V4,V2,V3
    SV      V4, 0(Ry)
```

Vector Processor Code



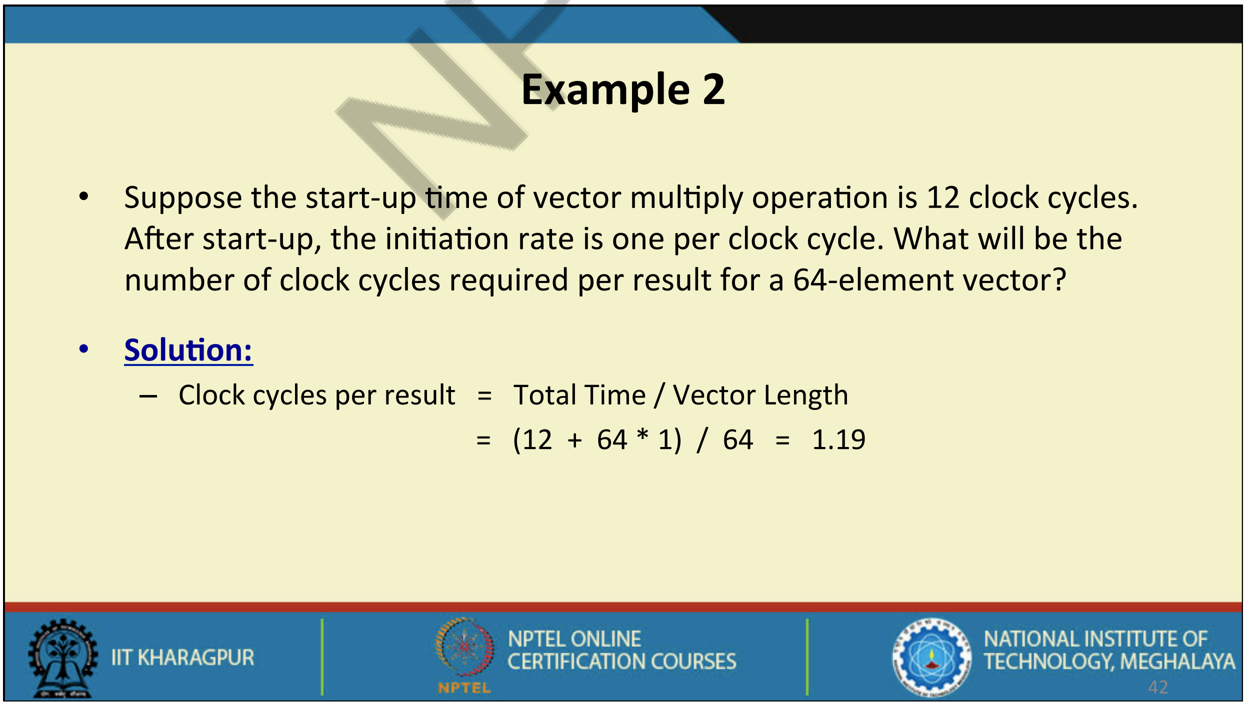
- The vector processor greatly reduces the dynamic instruction bandwidth :: *from 514 to 6*.
- Frequency of pipeline interlocks are also greatly reduced.
  - In the original MIPS32 version, every ADD.D must wait for MULT.D, and S.D must wait for ADD.D.
  - In the vector processor version, pipeline stalls are required once per vector operation, rather than once per vector element.
  - Pipeline stall frequency is reduced almost 64 times.



Vector Start-up and Initiation Rate

- The running time of each vector operation in the vector processor has two components:
  - **a) Start-up Time**: Arises due to the pipeline latency of the vector operation.
    - Mainly determined by the depth of the pipeline.
    - A latency of 8 clock cycles means that the operation takes 8 clock cycles, and also there are 8 stages in the pipeline.
  - **b) Initiation Rate**: Time per result once the vector instruction is running.
    - Usually 1 per clock cycle for individual operations.
- The total time to complete a vector operation of length n (n ≤ 64) is:

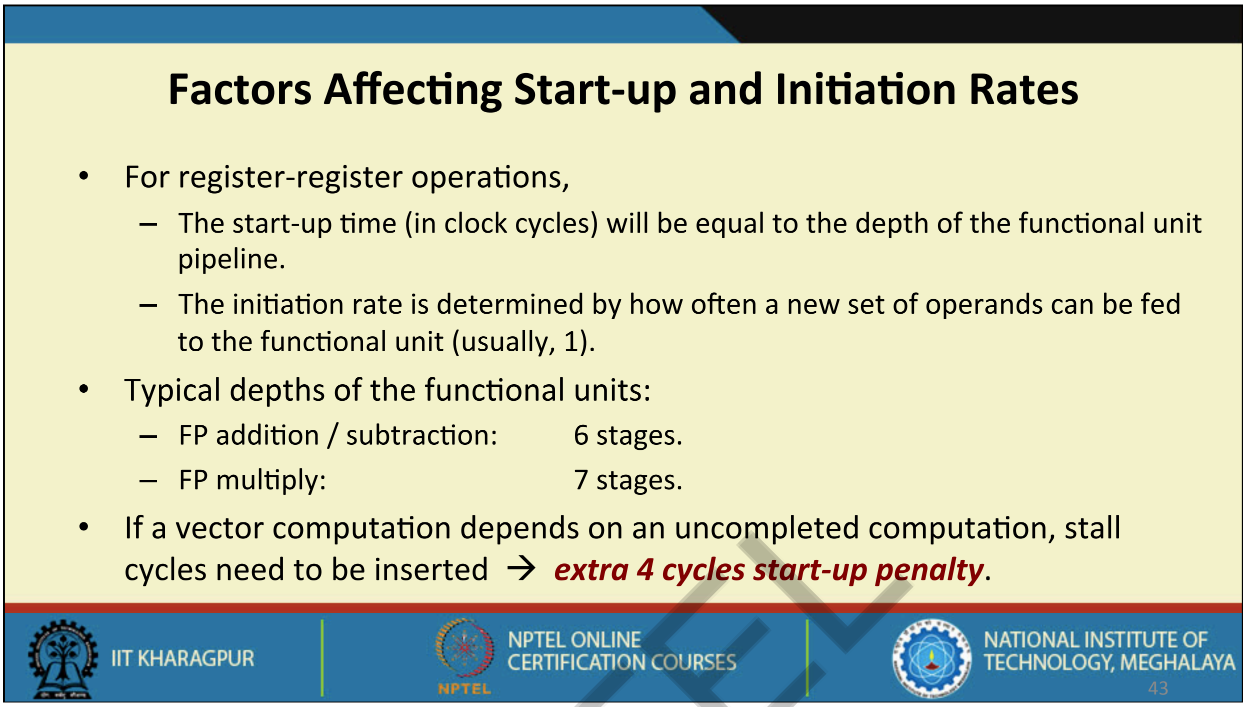
  Start-up Time + (n x Initiation Rate)



Example 2

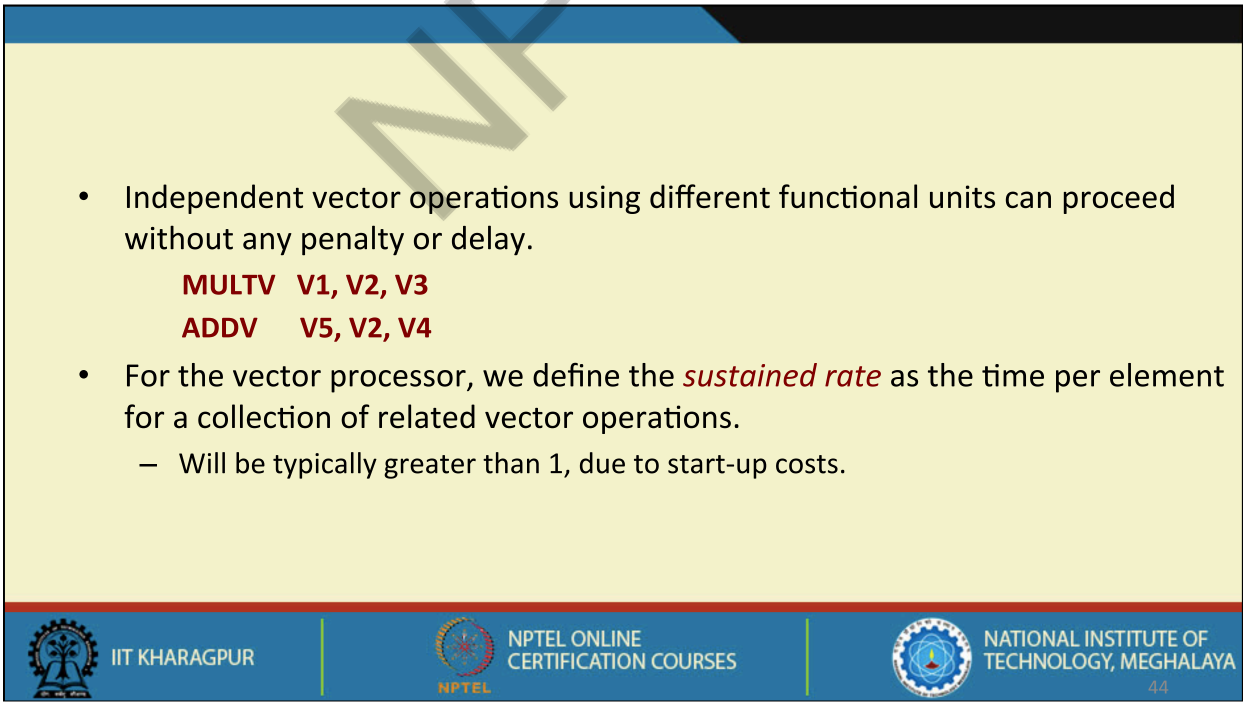
- Suppose the start-up time of vector multiply operation is 12 clock cycles. After start-up, the initiation rate is one per clock cycle. What will be the number of clock cycles required per result for a 64-element vector?
- **Solution:**
  - Clock cycles per result = Total Time / Vector Length

    = (12 + 64 * 1) / 64 = 1.19



Factors Affecting Start-up and Initiation Rates

- For register-register operations,
  - The start-up time (in clock cycles) will be equal to the depth of the functional unit pipeline.
  - The initiation rate is determined by how often a new set of operands can be fed to the functional unit (usually, 1).
- Typical depths of the functional units:
  - FP addition / subtraction: 6 stages.
  - FP multiply: 7 stages.
- If a vector computation depends on an uncompleted computation, stall cycles need to be inserted → ***extra 4 cycles start-up penalty***.

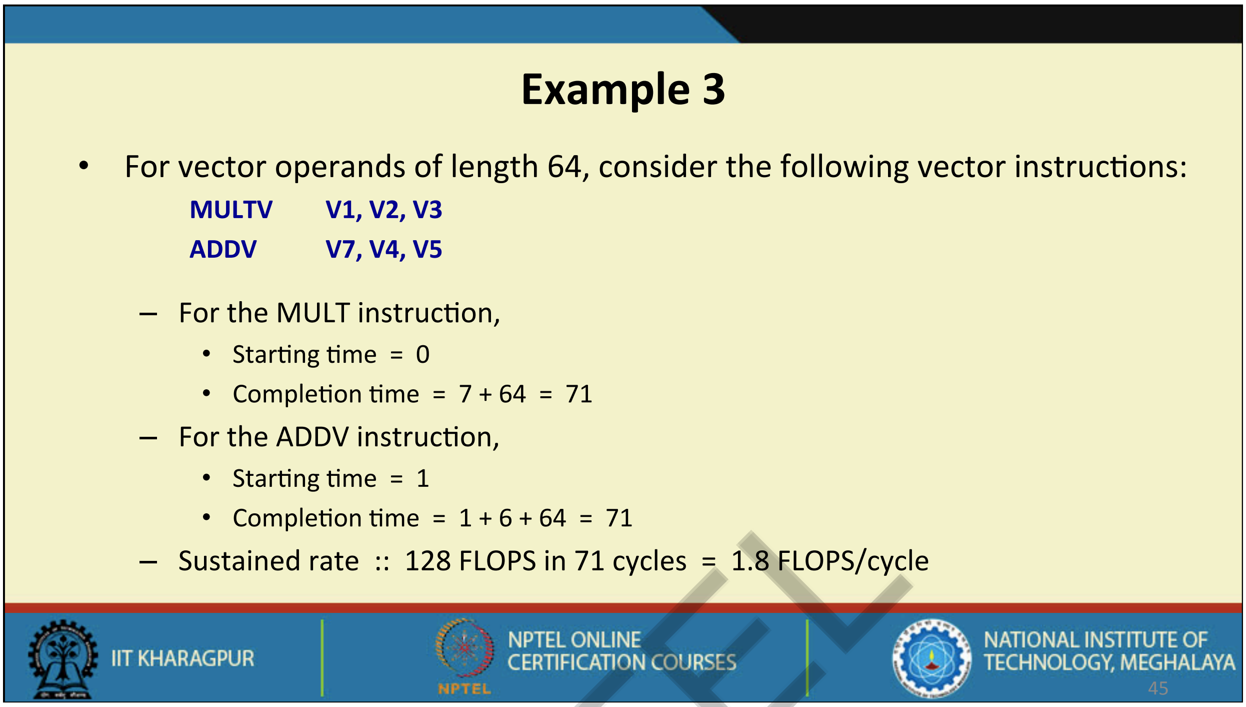


- Independent vector operations using different functional units can proceed without any penalty or delay.

  MULTV V1, V2, V3

  ADDV V5, V2, V4

- For the vector processor, we define the *sustained rate* as the time per element for a collection of related vector operations.
  - Will be typically greater than 1, due to start-up costs.

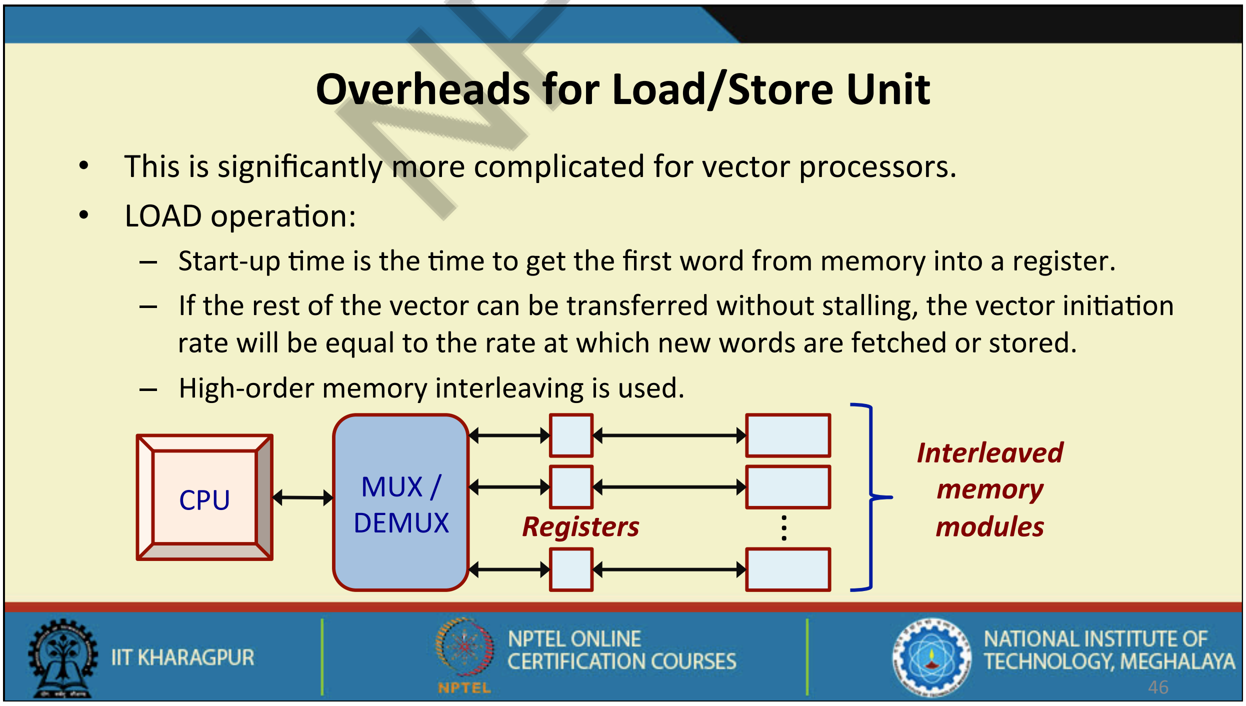


Example 3

- For vector operands of length 64, consider the following vector instructions:

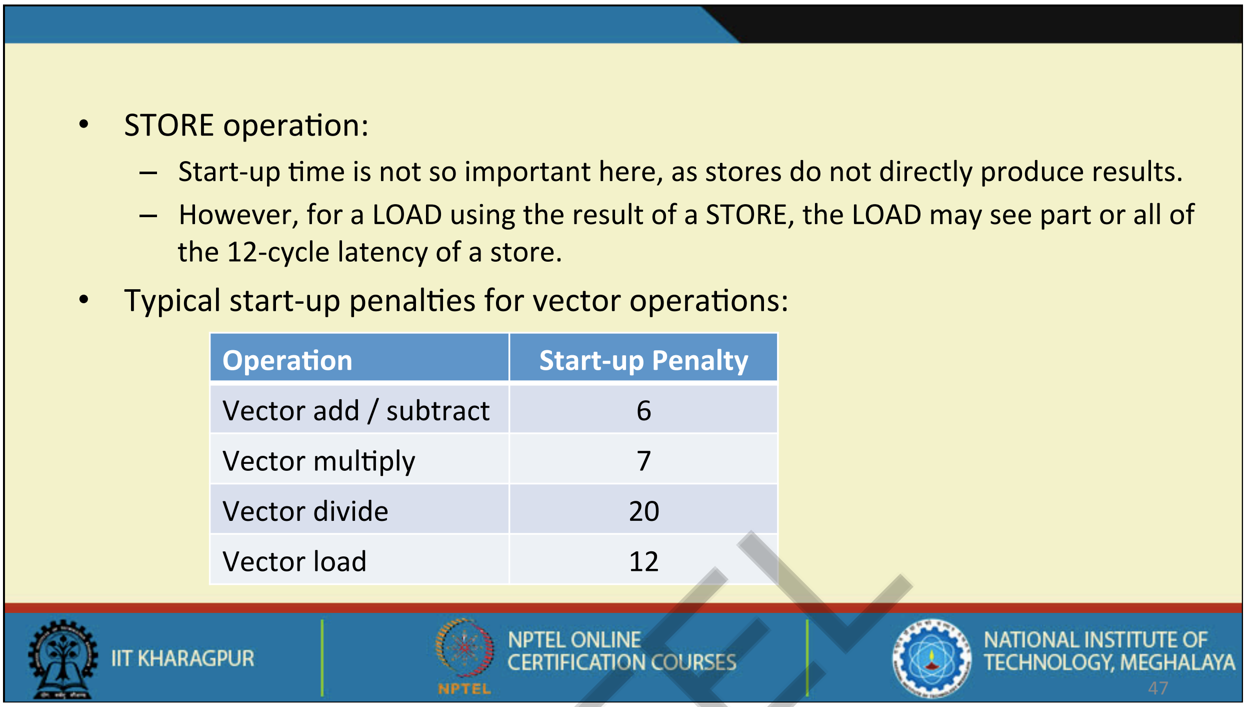
```
MULTV    V1, V2, V3
ADDV     V7, V4, V5
```

  – For the MULT instruction,
    - Starting time = 0
    - Completion time = 7 + 64 = 71
  – For the ADDV instruction,
    - Starting time = 1
    - Completion time = 1 + 6 + 64 = 71
  – Sustained rate :: 128 FLOPS in 71 cycles = 1.8 FLOPS/cycle



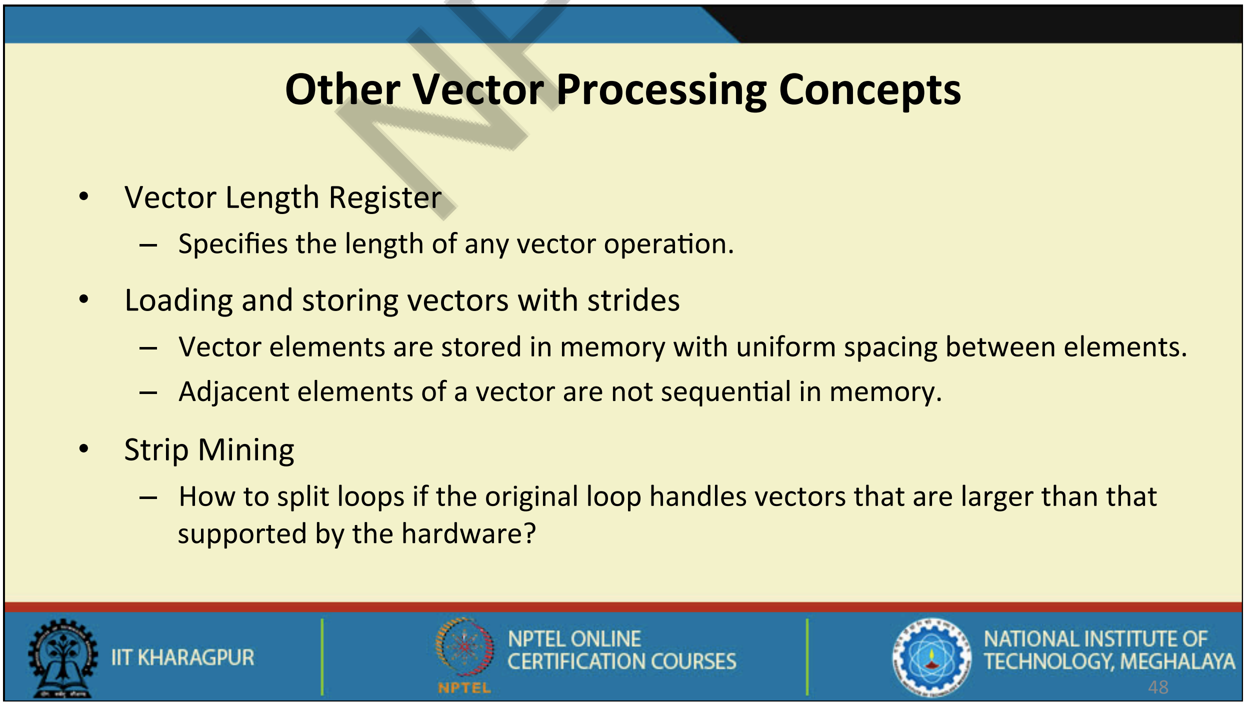
Overheads for Load/Store Unit

- This is significantly more complicated for vector processors.
- LOAD operation:
  – Start-up time is the time to get the first word from memory into a register.
  – If the rest of the vector can be transferred without stalling, the vector initiation rate will be equal to the rate at which new words are fetched or stored.
  – High-order memory interleaving is used.



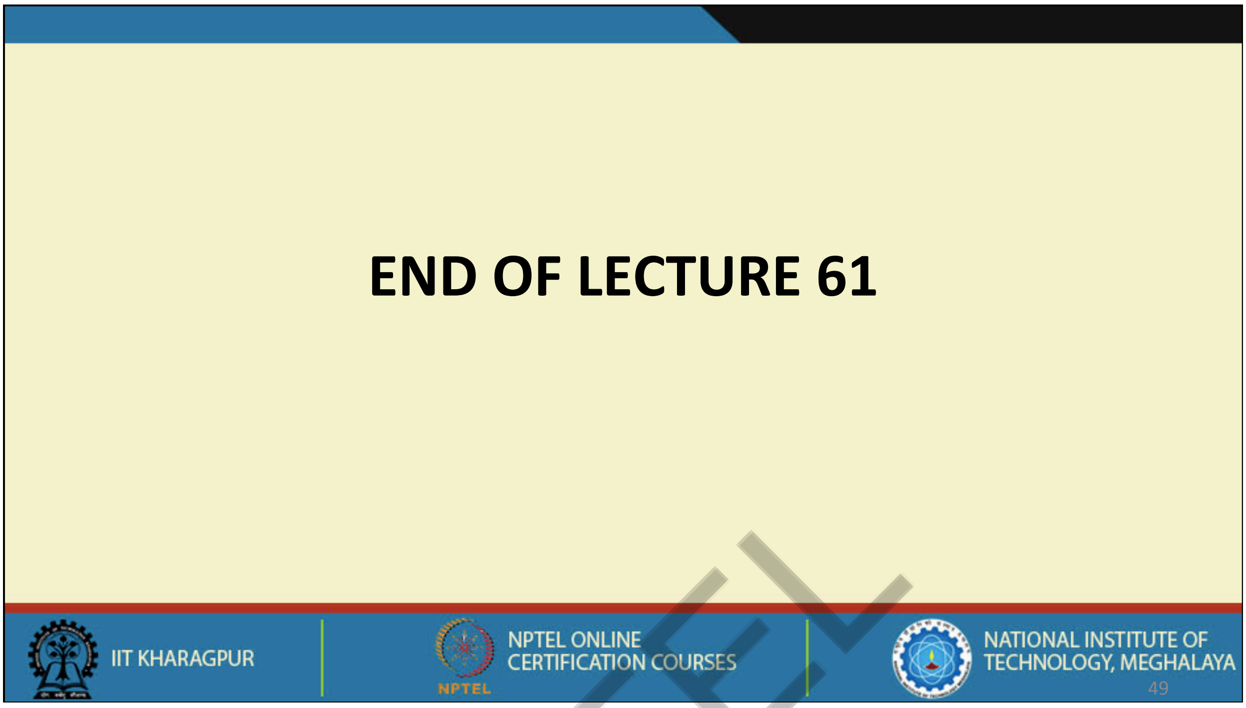
- STORE operation:
  - Start-up time is not so important here, as stores do not directly produce results.
  - However, for a LOAD using the result of a STORE, the LOAD may see part or all of the 12-cycle latency of a store.
- Typical start-up penalties for vector operations:

Operation	Start-up Penalty
Vector add / subtract	6
Vector multiply	7
Vector divide	20
Vector load	12



Other Vector Processing Concepts

- Vector Length Register
  - Specifies the length of any vector operation.
- Loading and storing vectors with strides
  - Vector elements are stored in memory with uniform spacing between elements.
  - Adjacent elements of a vector are not sequential in memory.
- Strip Mining
  - How to split loops if the original loop handles vectors that are larger than that supported by the hardware?



END OF LECTURE 61

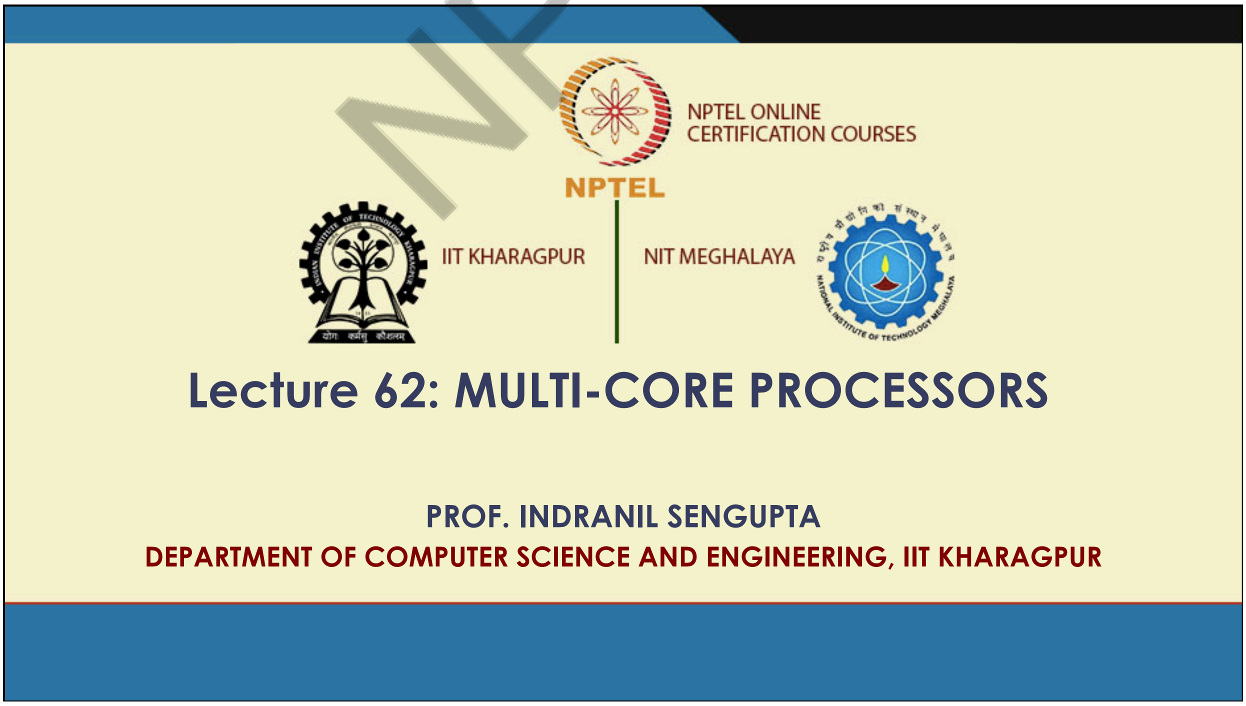
IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

NATIONAL INSTITUTE OF
TECHNOLOGY, MEGHALAYA

49



NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

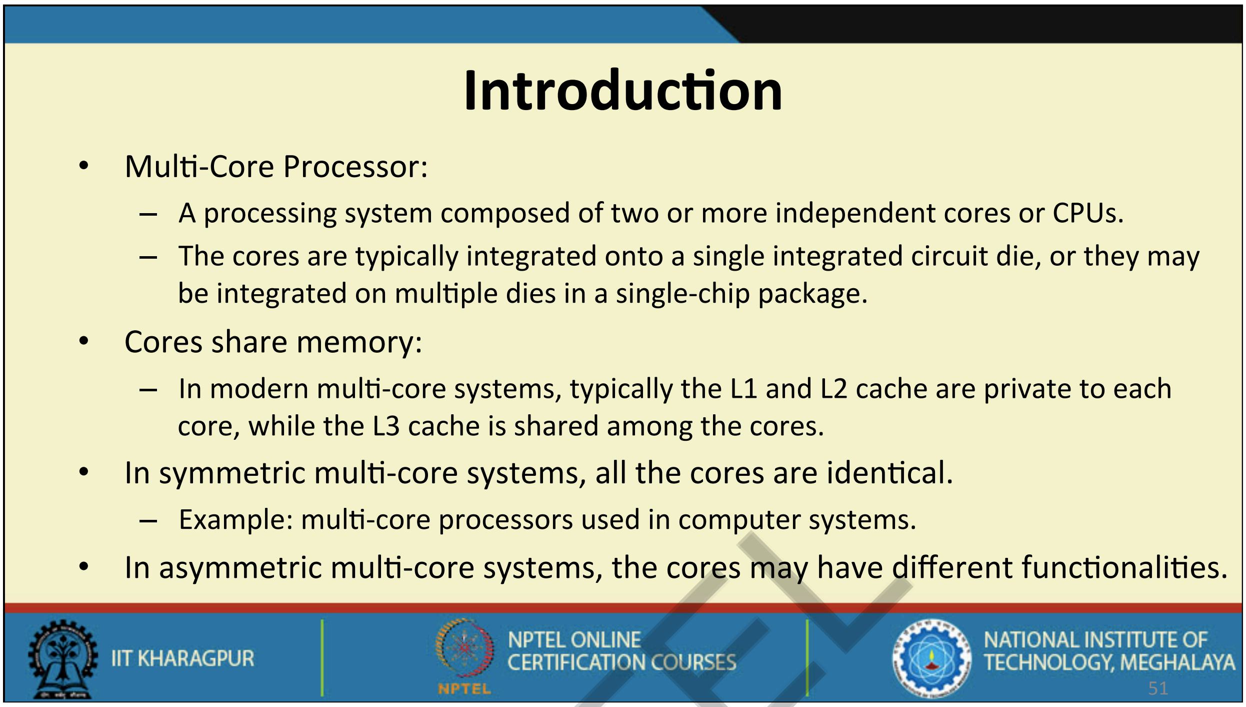
IIT KHARAGPUR

NIT MEGHALAYA

Lecture 62: MULTI-CORE PROCESSORS

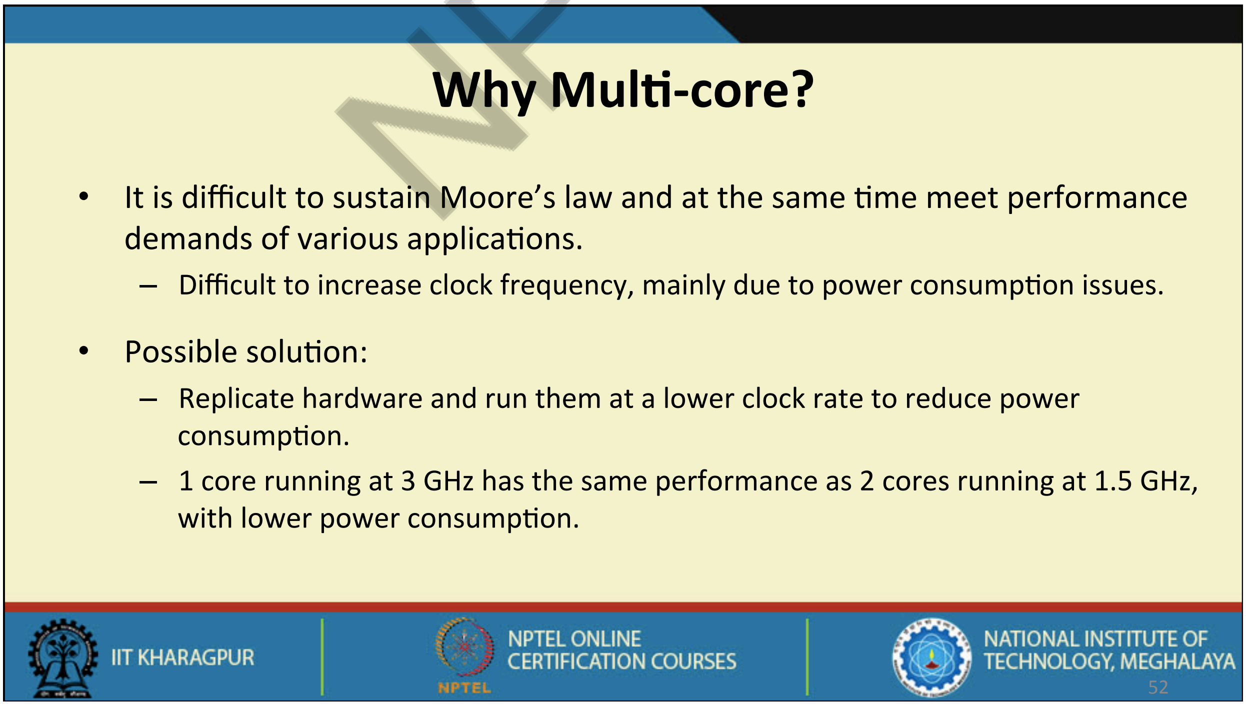
PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, IIT KHARAGPUR



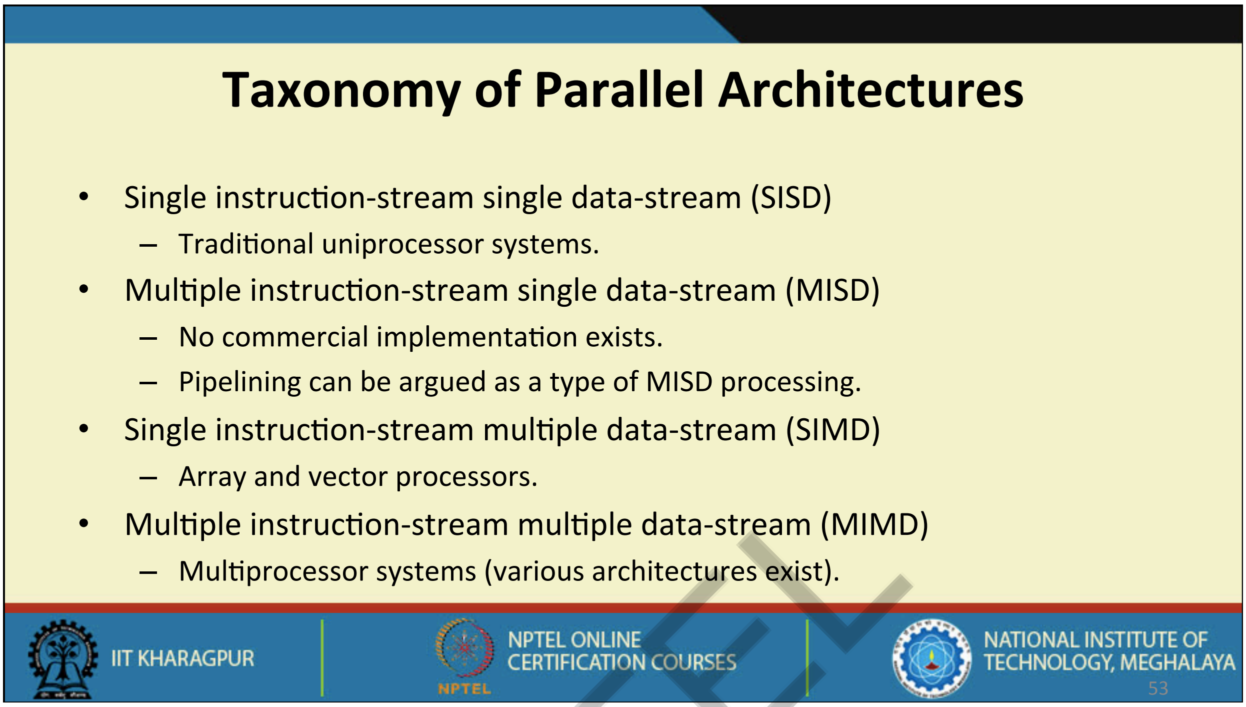
Introduction

- Multi-Core Processor:
  - A processing system composed of two or more independent cores or CPUs.
  - The cores are typically integrated onto a single integrated circuit die, or they may be integrated on multiple dies in a single-chip package.
- Cores share memory:
  - In modern multi-core systems, typically the L1 and L2 cache are private to each core, while the L3 cache is shared among the cores.
- In symmetric multi-core systems, all the cores are identical.
  - Example: multi-core processors used in computer systems.
- In asymmetric multi-core systems, the cores may have different functionalities.



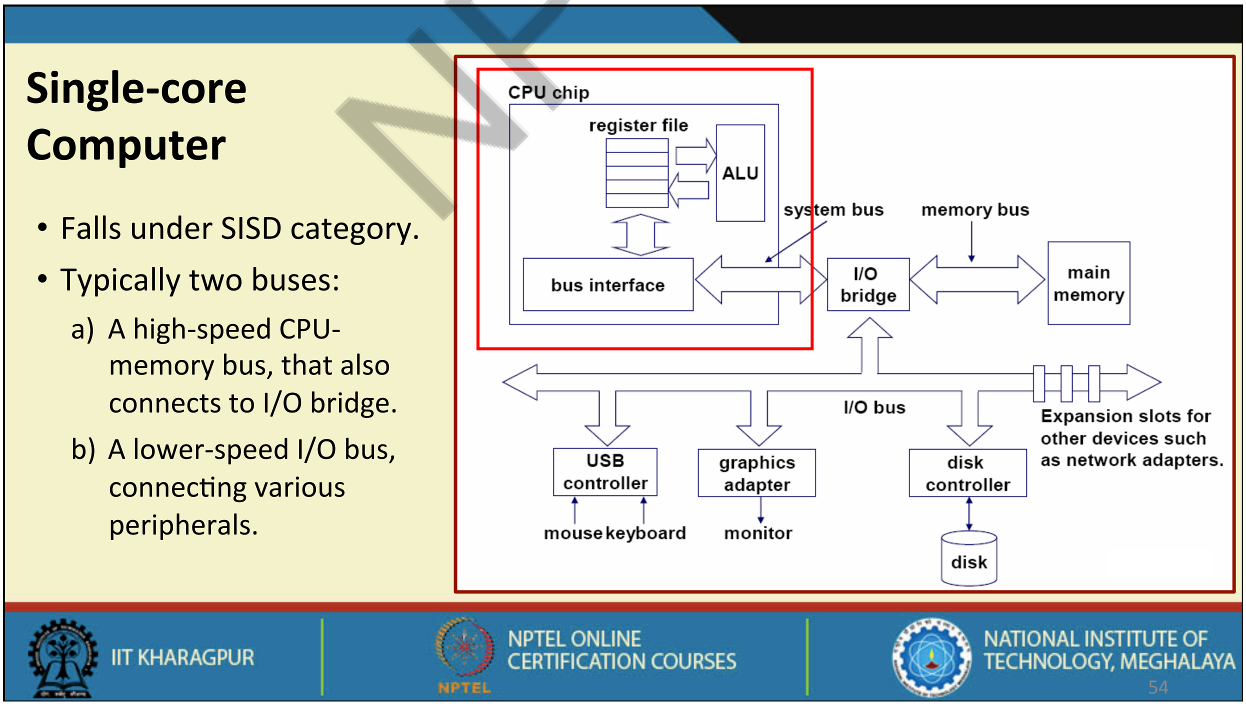
Why Multi-core?

- It is difficult to sustain Moore's law and at the same time meet performance demands of various applications.
  - Difficult to increase clock frequency, mainly due to power consumption issues.
- Possible solution:
  - Replicate hardware and run them at a lower clock rate to reduce power consumption.
  - 1 core running at 3 GHz has the same performance as 2 cores running at 1.5 GHz, with lower power consumption.



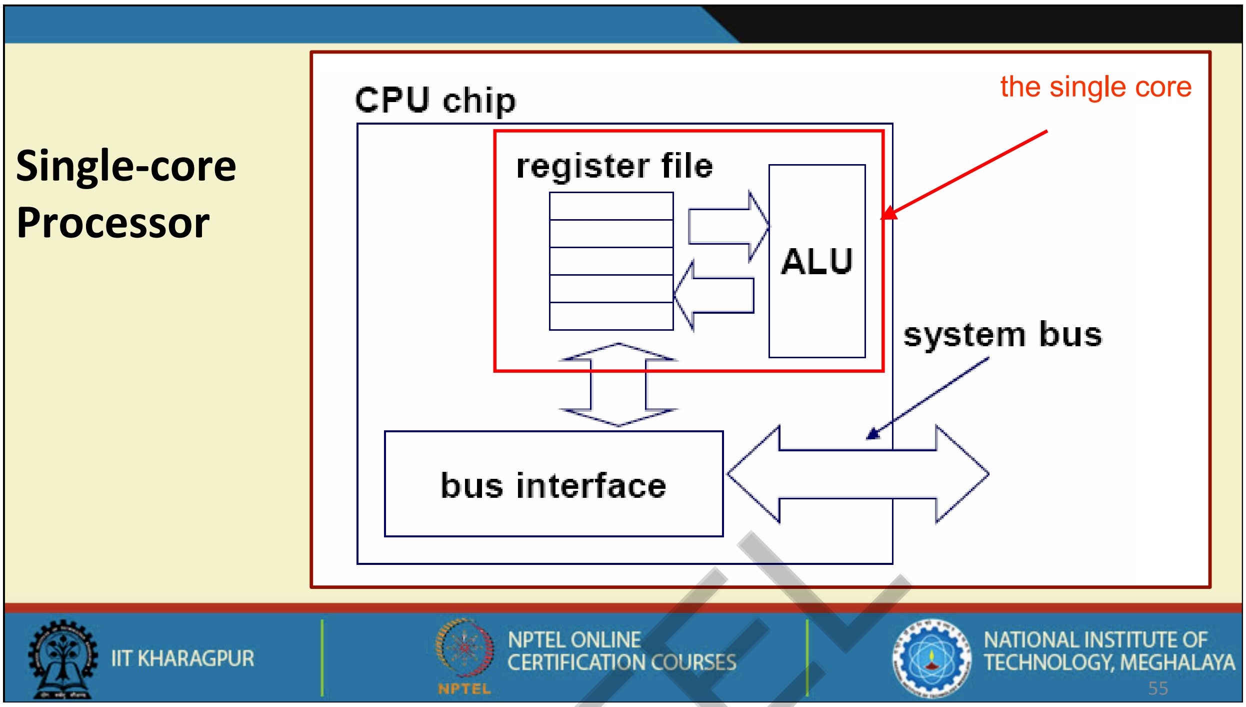
Taxonomy of Parallel Architectures

- Single instruction-stream single data-stream (SISD)
  - Traditional uniprocessor systems.
- Multiple instruction-stream single data-stream (MISD)
  - No commercial implementation exists.
  - Pipelining can be argued as a type of MISD processing.
- Single instruction-stream multiple data-stream (SIMD)
  - Array and vector processors.
- Multiple instruction-stream multiple data-stream (MIMD)
  - Multiprocessor systems (various architectures exist).



Single-core Computer

- Falls under SISD category.
- Typically two buses:
  a) A high-speed CPU-memory bus, that also connects to I/O bridge.
  b) A lower-speed I/O bus, connecting various peripherals.



Single-core Processor

CPU chip

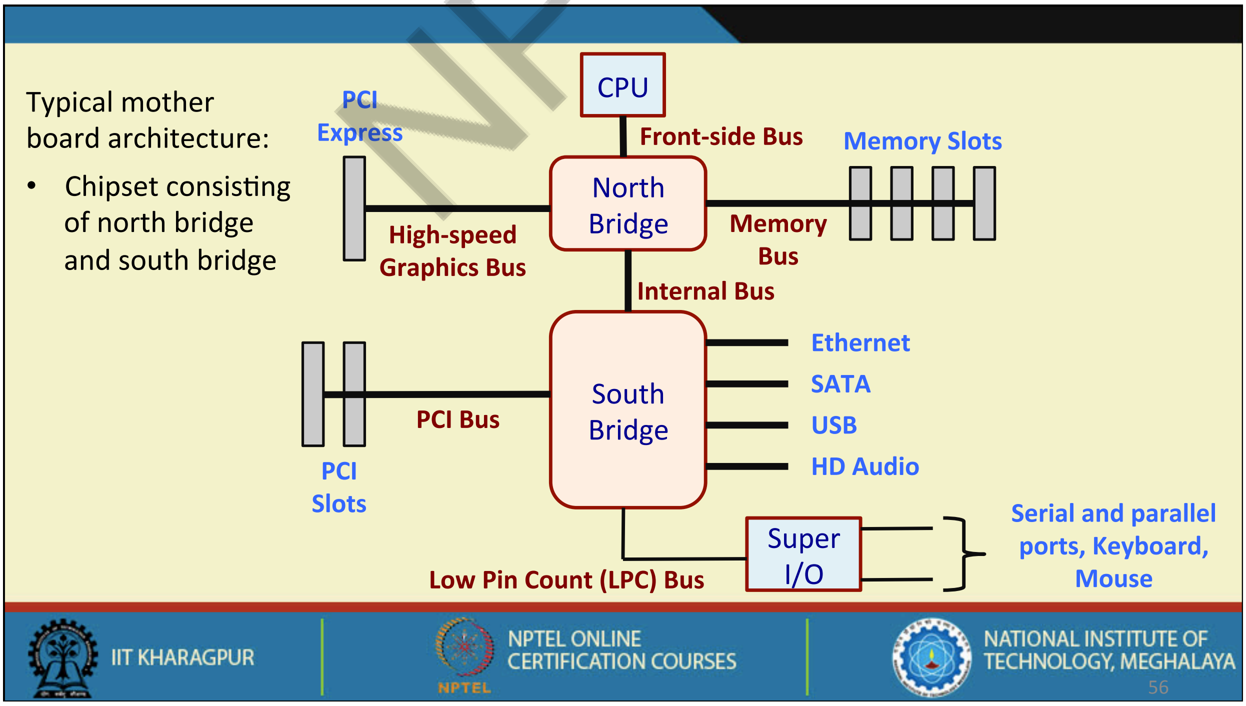
the single core

register file

ALU

system bus

bus interface



Typical mother board architecture:

- Chipset consisting of north bridge and south bridge

CPU

Front-side Bus

PCI Express

North Bridge

Memory Slots

High-speed Graphics Bus

Memory Bus

Internal Bus

South Bridge

Ethernet

SATA

USB

HD Audio

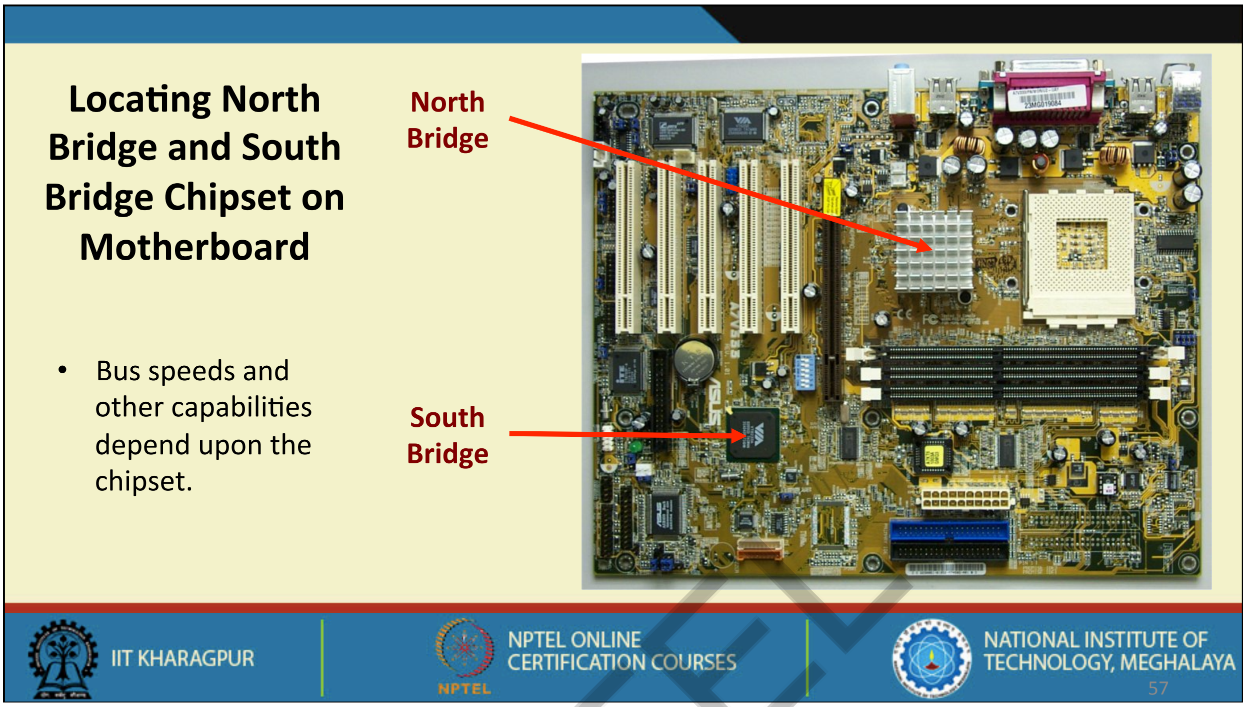
PCI Bus

PCI Slots

Super I/O

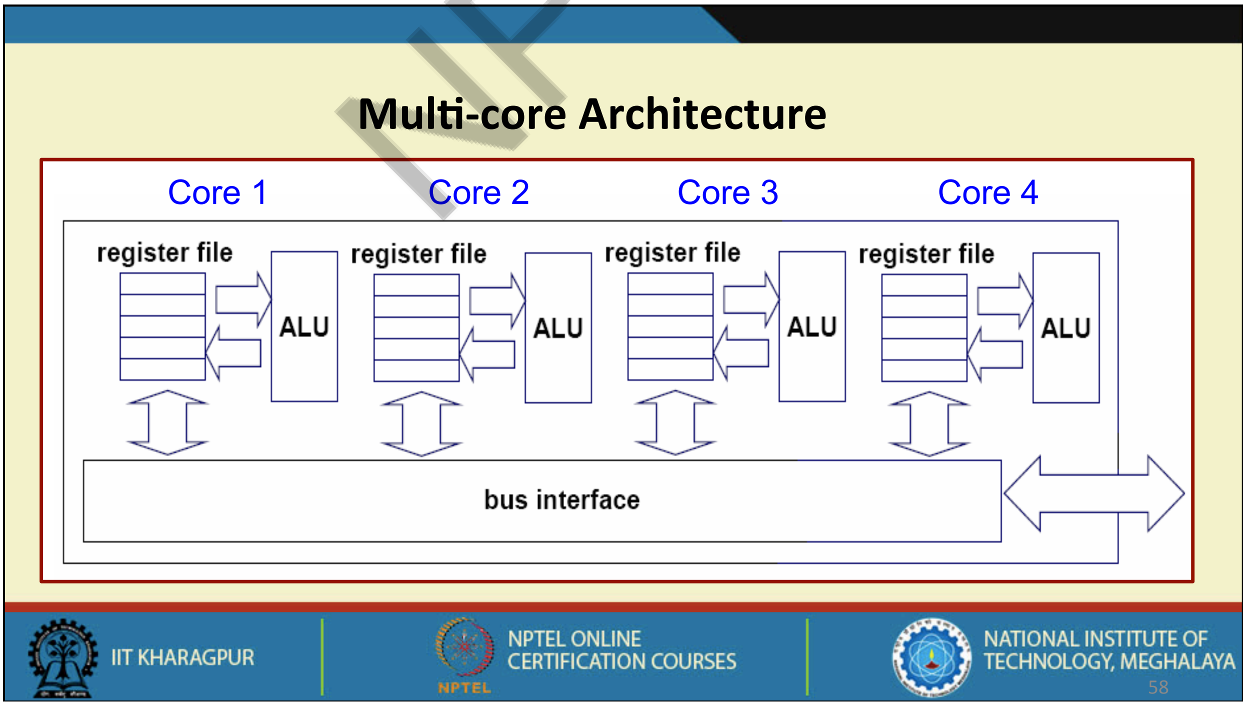
Serial and parallel ports, Keyboard, Mouse

Low Pin Count (LPC) Bus

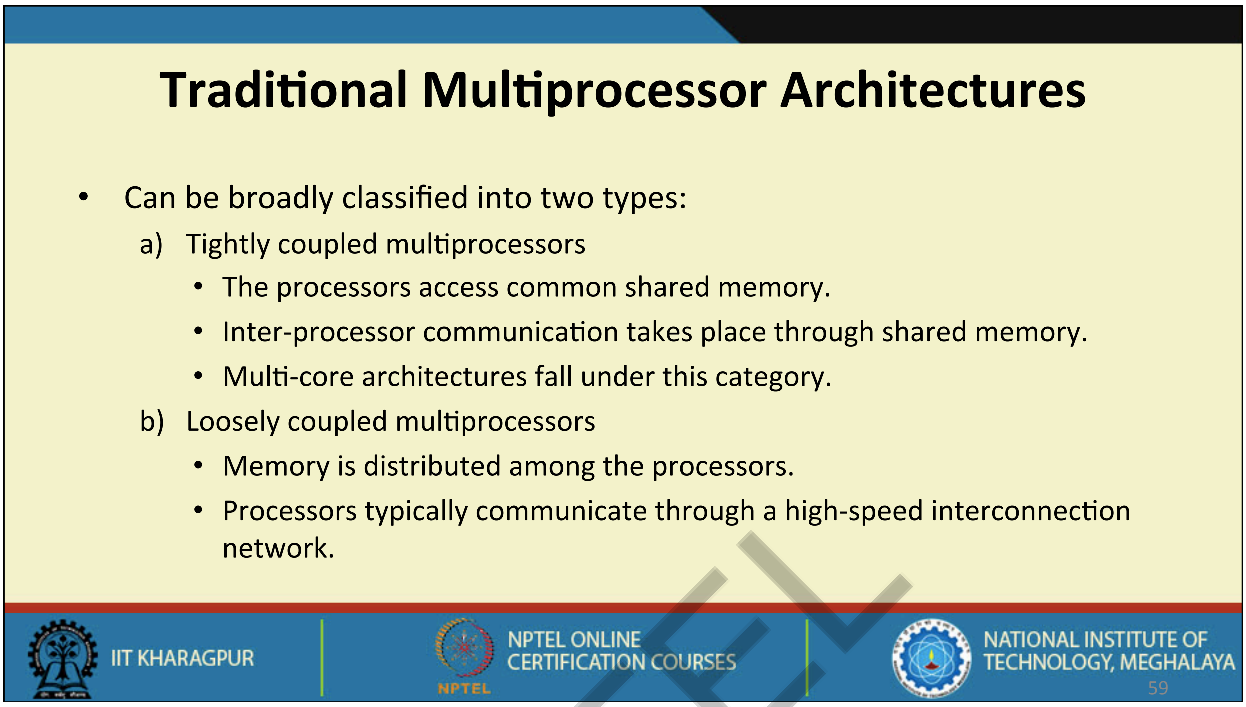


Locating North Bridge and South Bridge Chipset on Motherboard

- Bus speeds and other capabilities depend upon the chipset.



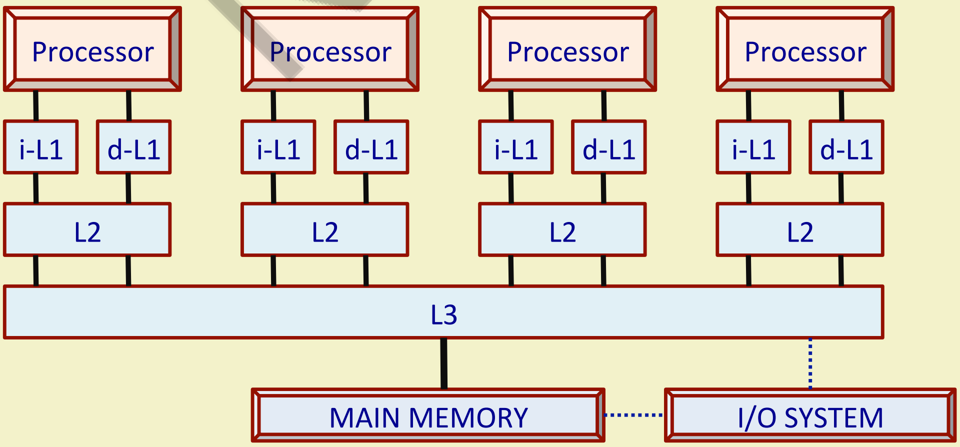
Multi-core Architecture



Traditional Multiprocessor Architectures

- Can be broadly classified into two types:
  - a) Tightly coupled multiprocessors
    - The processors access common shared memory.
    - Inter-processor communication takes place through shared memory.
    - Multi-core architectures fall under this category.
  - b) Loosely coupled multiprocessors
    - Memory is distributed among the processors.
    - Processors typically communicate through a high-speed interconnection network.

(a) Tightly Coupled Multiprocessors



Processor Processor Processor Processor

i-L1 d-L1 i-L1 d-L1 i-L1 d-L1 i-L1 d-L1

L2 L2 L2 L2

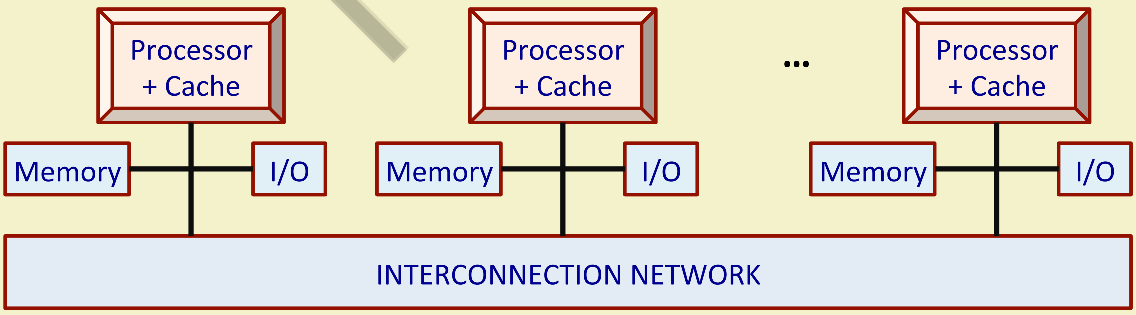
L3

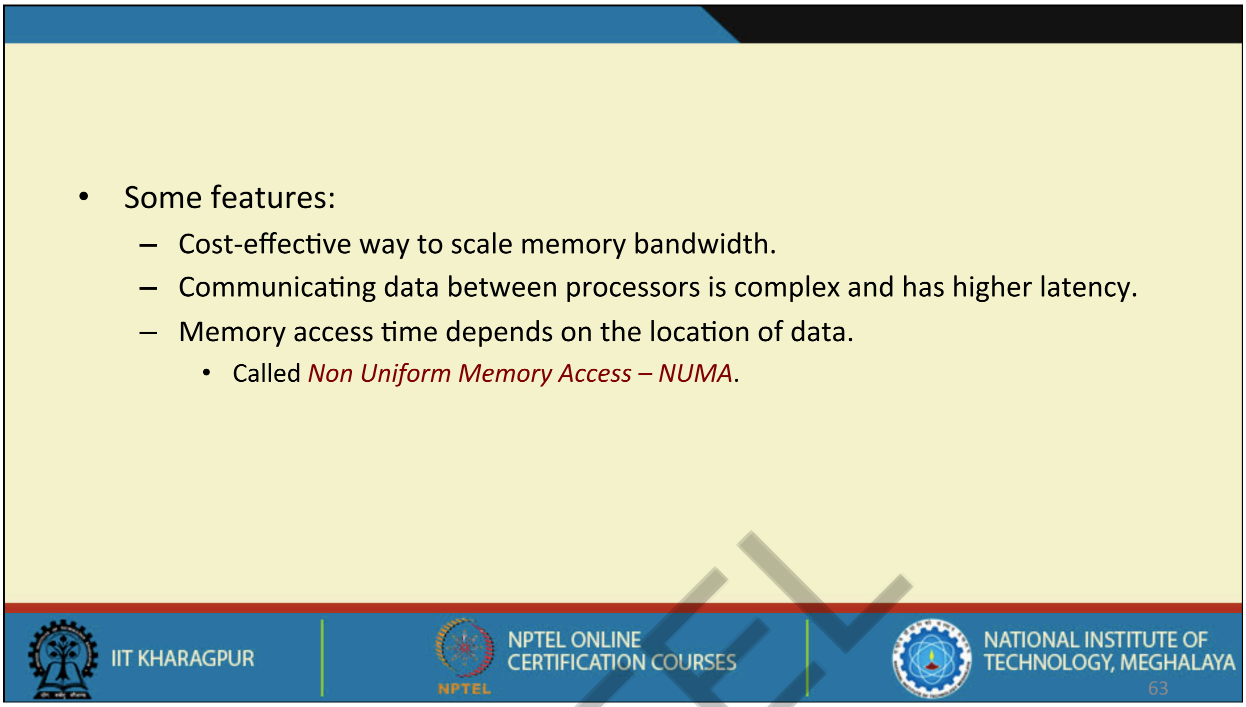
MAIN MEMORY

I/O SYSTEM

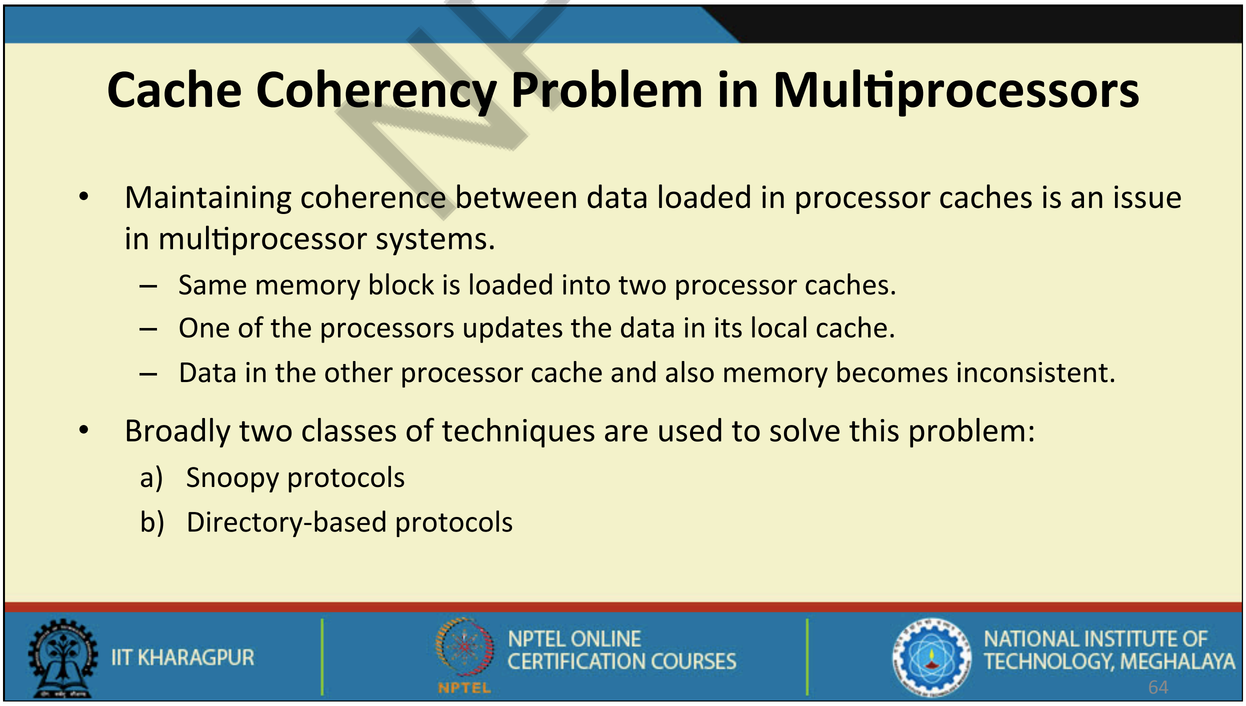
- Some features:
  - – Difficult to extend it to large number of processors.
  - – Memory bandwidth requirements increase with the number of processors.
  - – Memory access time for all processors is uniform.
    - Called *Uniform Memory Access – UMA*.

(b) Loosely Coupled Multiprocessors



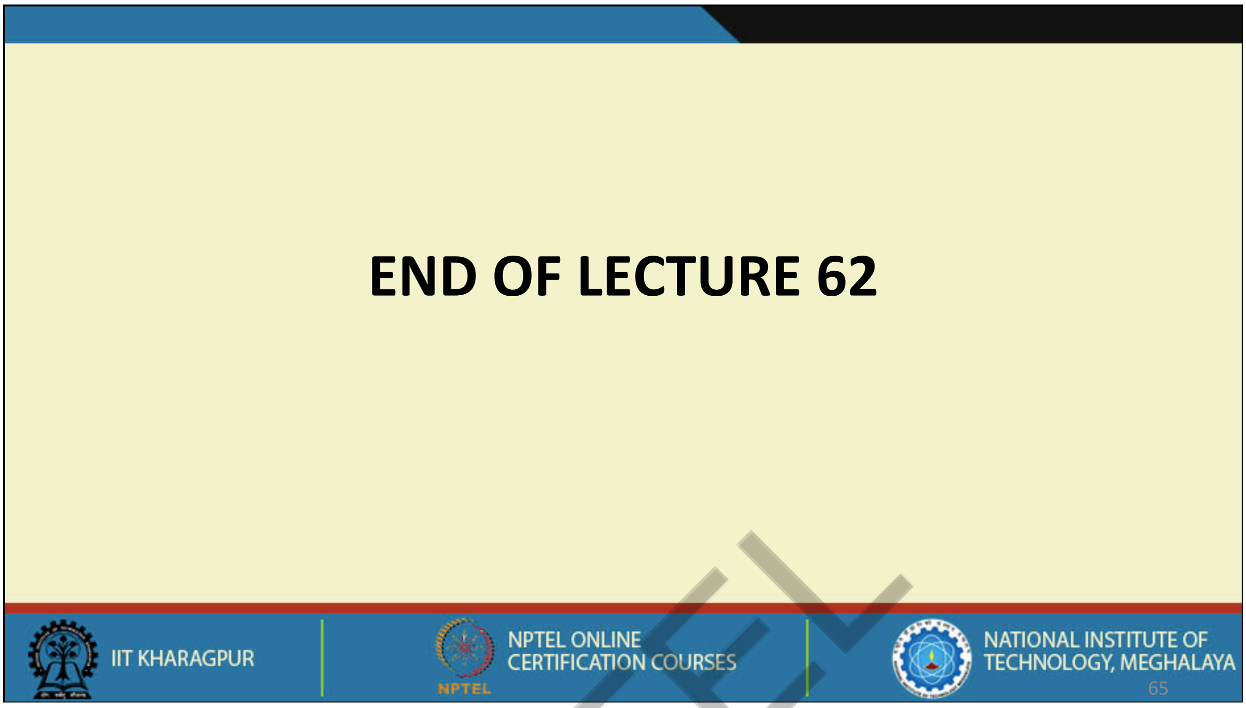


- Some features:
  - Cost-effective way to scale memory bandwidth.
  - Communicating data between processors is complex and has higher latency.
  - Memory access time depends on the location of data.
    - Called *Non Uniform Memory Access – NUMA*.

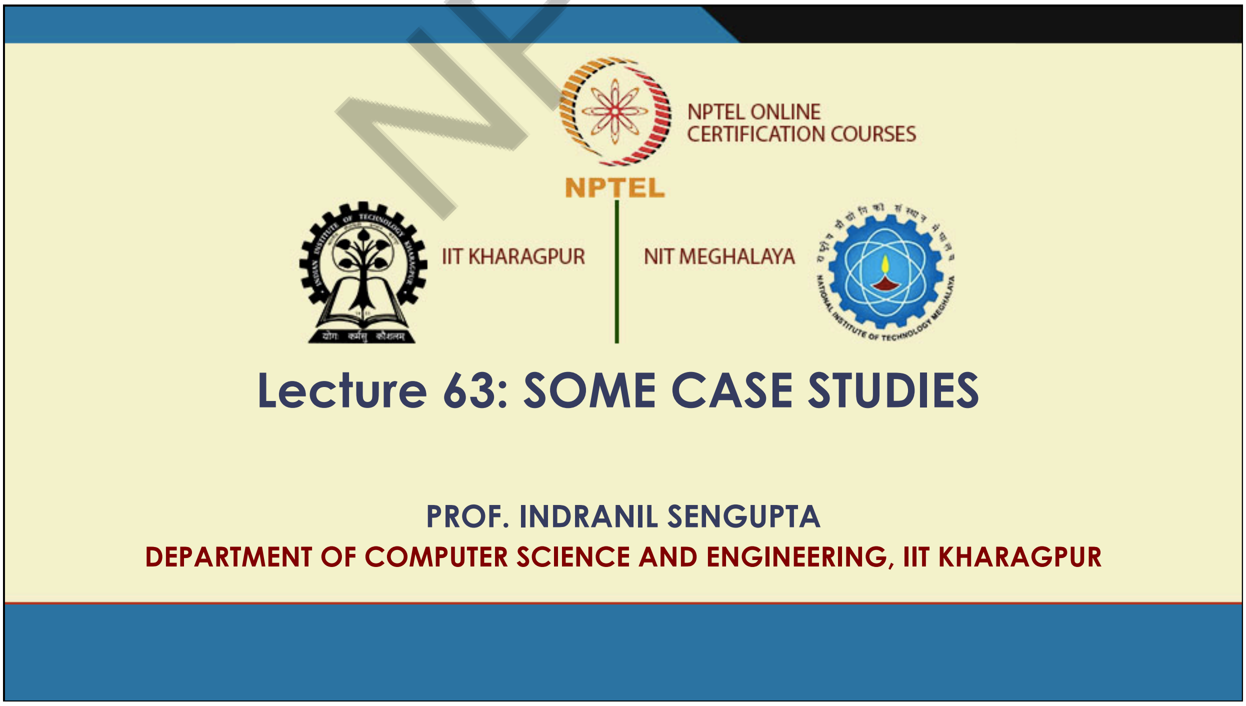


Cache Coherency Problem in Multiprocessors

- Maintaining coherence between data loaded in processor caches is an issue in multiprocessor systems.
  - Same memory block is loaded into two processor caches.
  - One of the processors updates the data in its local cache.
  - Data in the other processor cache and also memory becomes inconsistent.
- Broadly two classes of techniques are used to solve this problem:
  a) Snoopy protocols
  b) Directory-based protocols



END OF LECTURE 62



NPTEL ONLINE CERTIFICATION COURSES

NPTEL

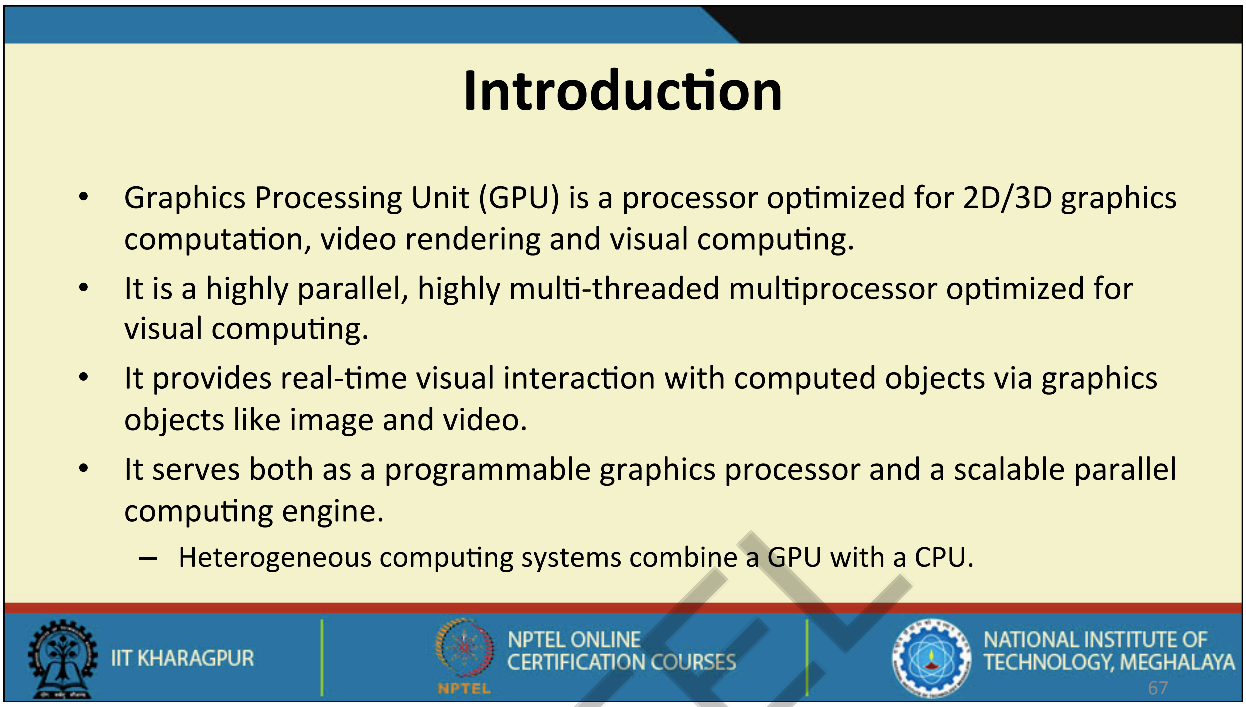
IIT KHARAGPUR

NIT MEGHALAYA

Lecture 63: SOME CASE STUDIES

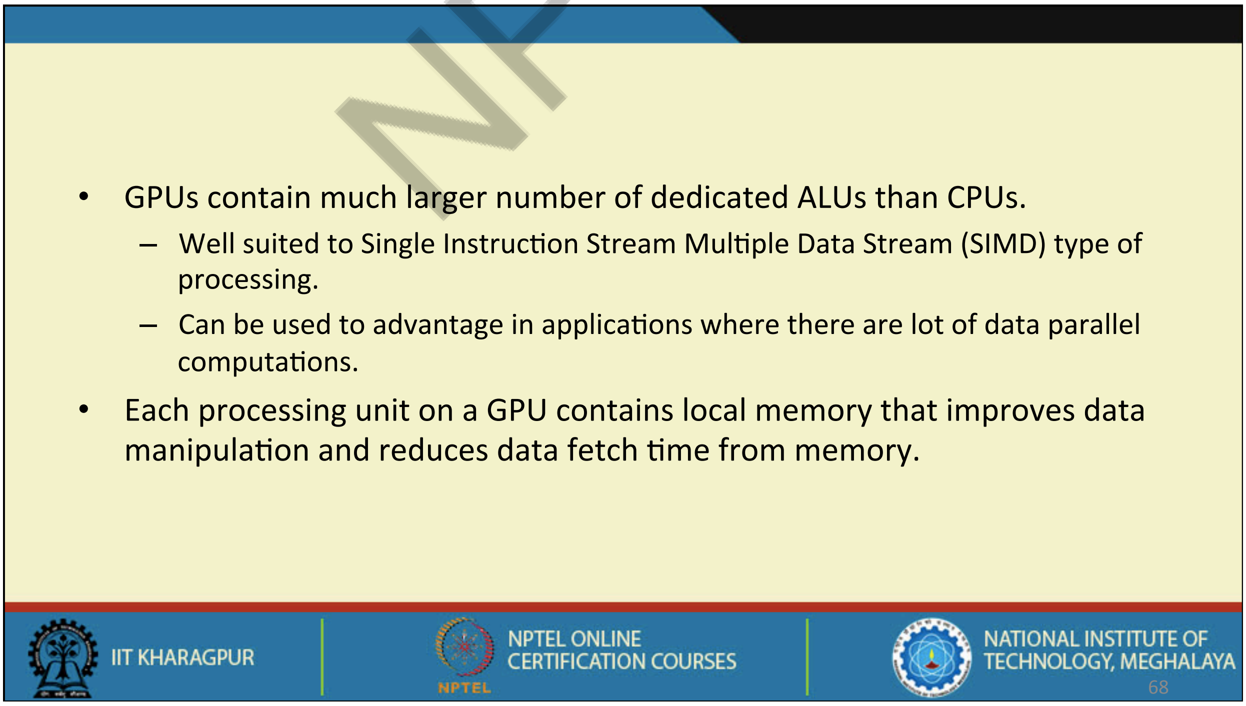
PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, IIT KHARAGPUR

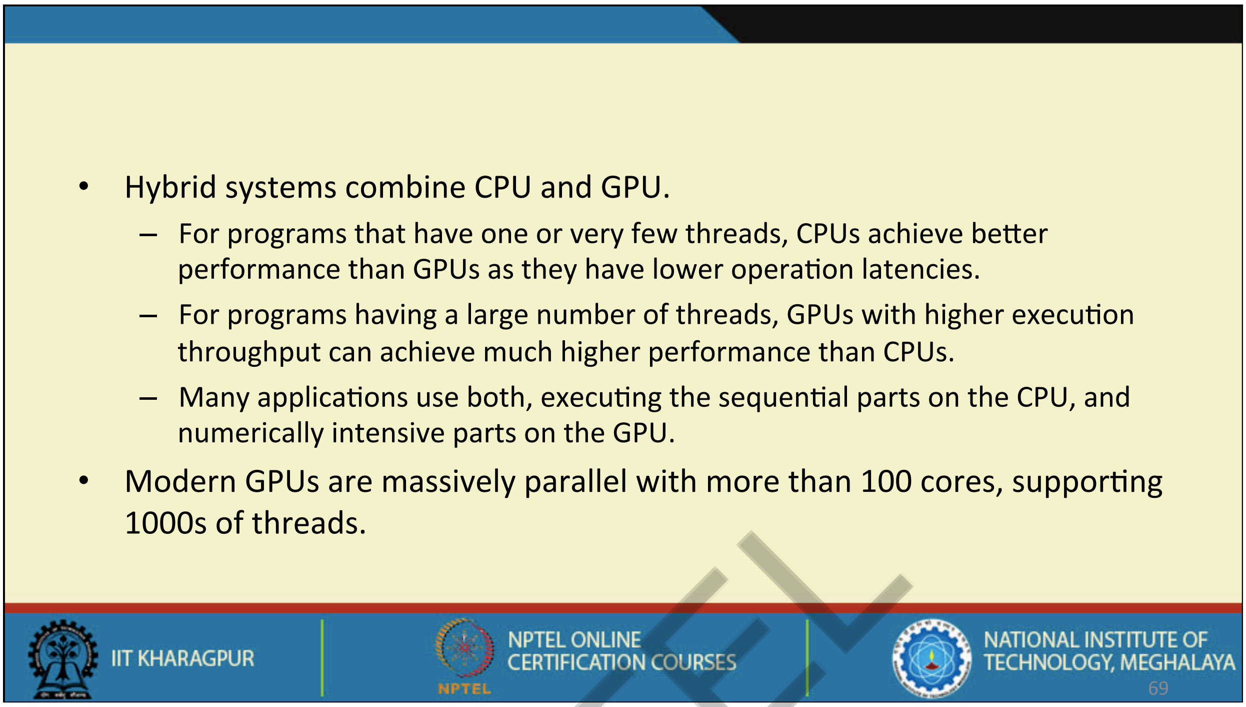


Introduction

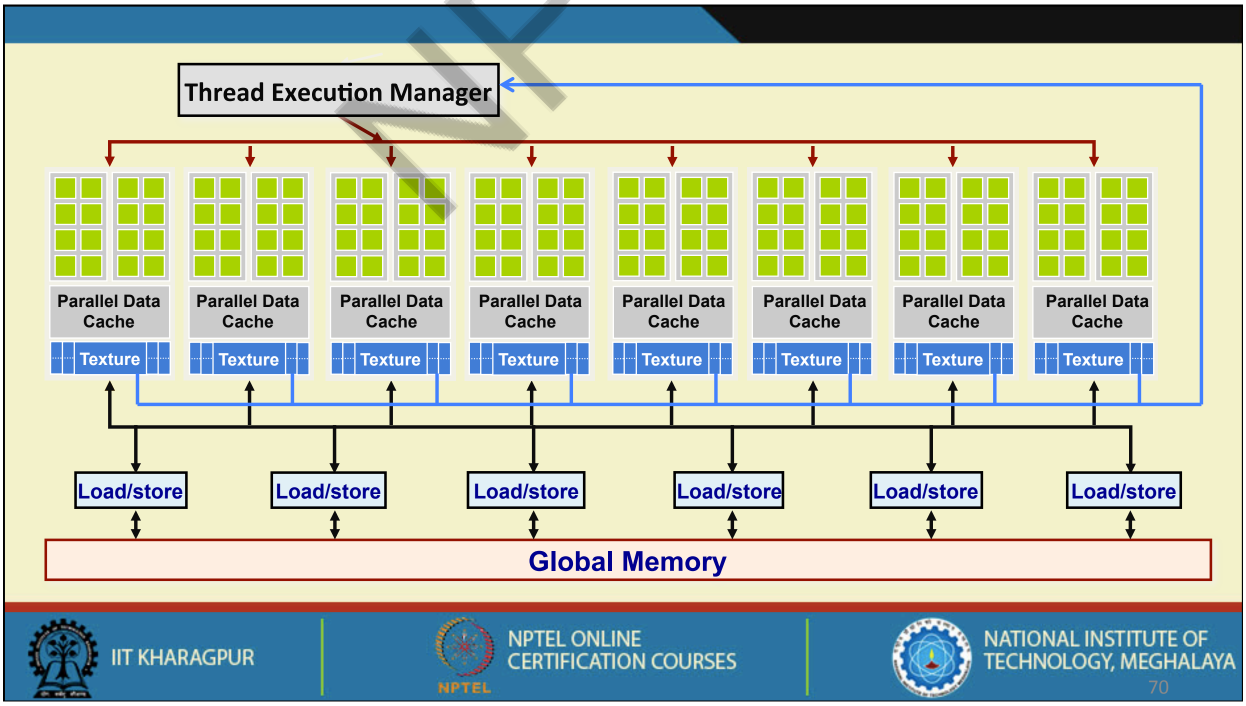
- Graphics Processing Unit (GPU) is a processor optimized for 2D/3D graphics computation, video rendering and visual computing.
- It is a highly parallel, highly multi-threaded multiprocessor optimized for visual computing.
- It provides real-time visual interaction with computed objects via graphics objects like image and video.
- It serves both as a programmable graphics processor and a scalable parallel computing engine.
  - Heterogeneous computing systems combine a GPU with a CPU.



- GPUs contain much larger number of dedicated ALUs than CPUs.
  - Well suited to Single Instruction Stream Multiple Data Stream (SIMD) type of processing.
  - Can be used to advantage in applications where there are lot of data parallel computations.
- Each processing unit on a GPU contains local memory that improves data manipulation and reduces data fetch time from memory.



- Hybrid systems combine CPU and GPU.
  - For programs that have one or very few threads, CPUs achieve better performance than GPUs as they have lower operation latencies.
  - For programs having a large number of threads, GPUs with higher execution throughput can achieve much higher performance than CPUs.
  - Many applications use both, executing the sequential parts on the CPU, and numerically intensive parts on the GPU.
- Modern GPUs are massively parallel with more than 100 cores, supporting 1000s of threads.



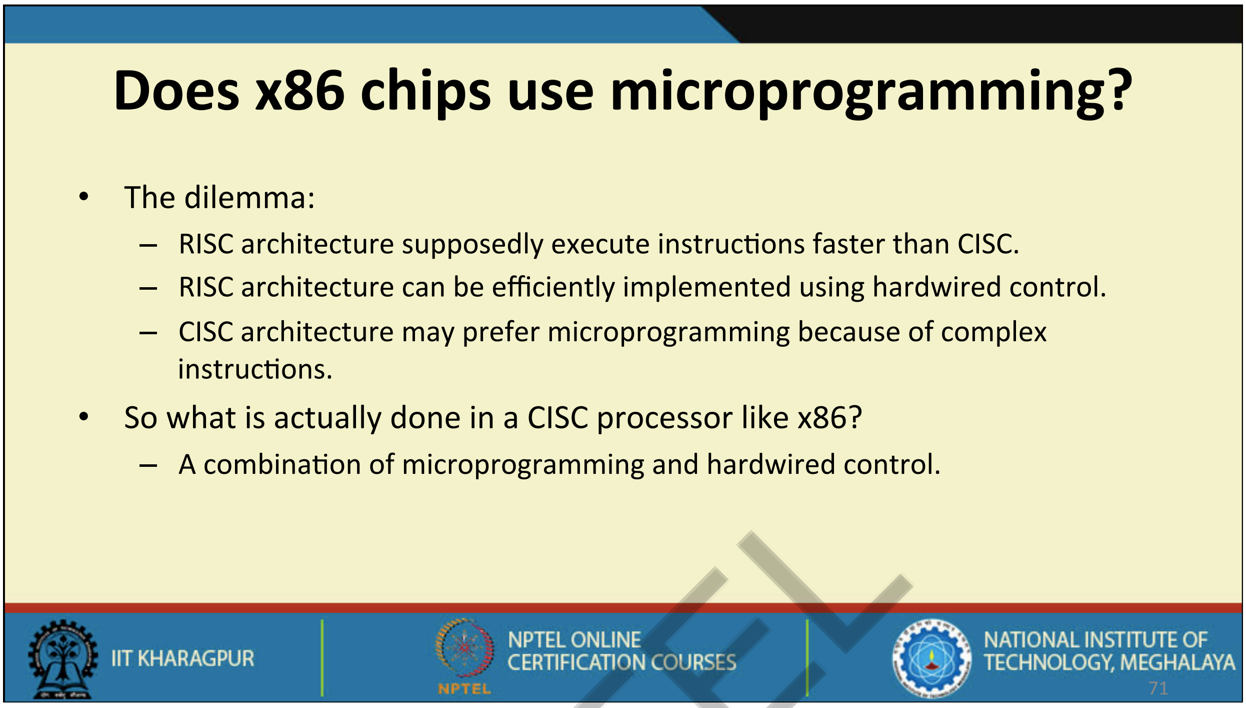
Thread Execution Manager

Parallel Data Cache | Parallel Data Cache | Parallel Data Cache | Parallel Data Cache | Parallel Data Cache | Parallel Data Cache | Parallel Data Cache | Parallel Data Cache

Texture | Texture | Texture | Texture | Texture | Texture | Texture | Texture

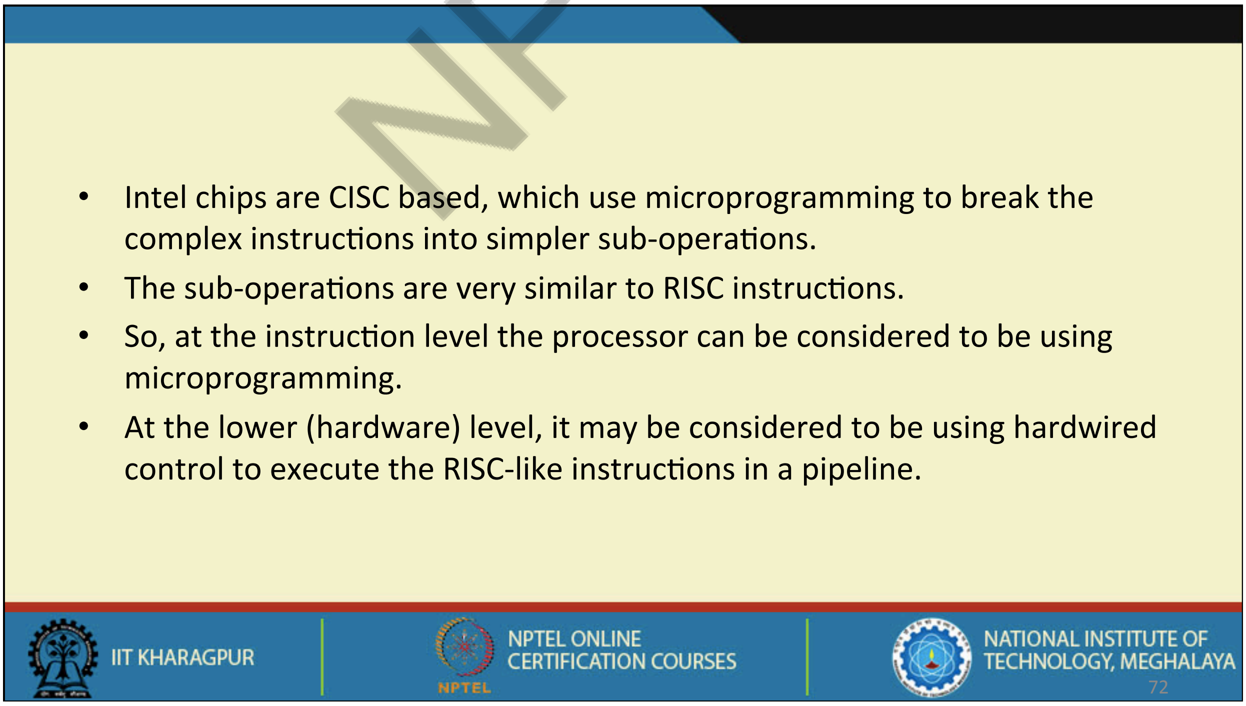
Load/store | Load/store | Load/store | Load/store | Load/store | Load/store

Global Memory

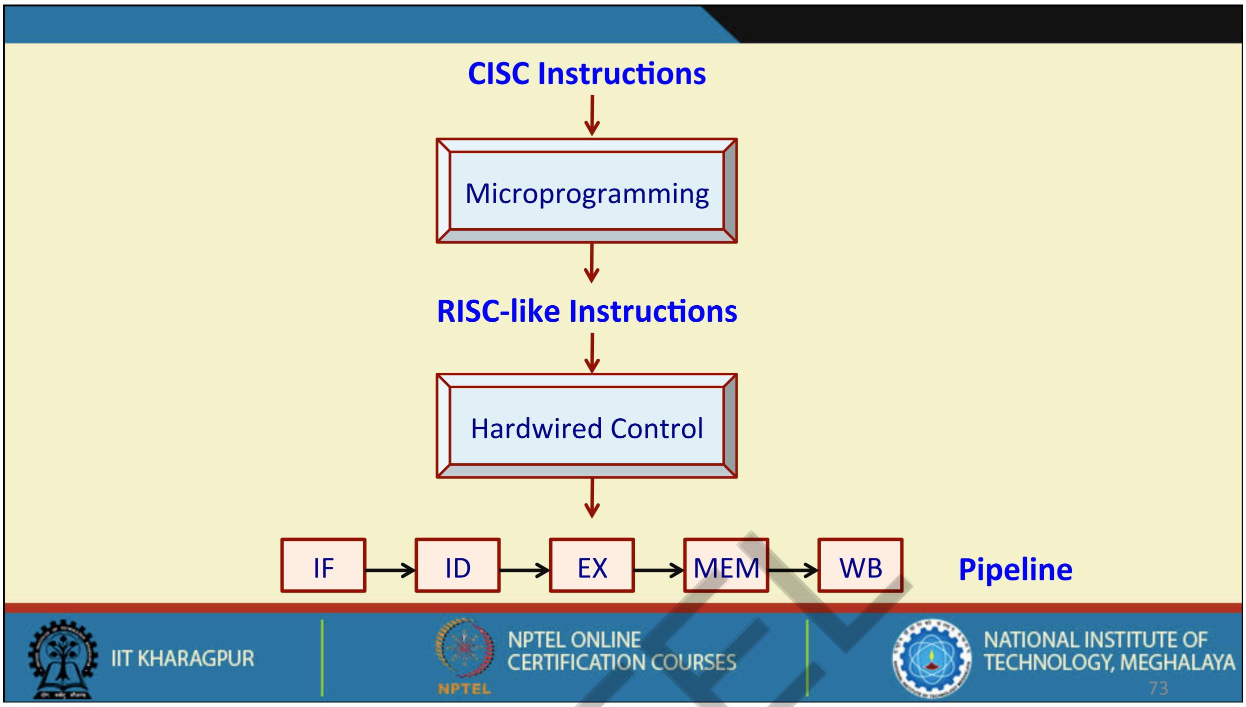


Does x86 chips use microprogramming?

- The dilemma:
  - RISC architecture supposedly execute instructions faster than CISC.
  - RISC architecture can be efficiently implemented using hardwired control.
  - CISC architecture may prefer microprogramming because of complex instructions.
- So what is actually done in a CISC processor like x86?
  - A combination of microprogramming and hardwired control.



- Intel chips are CISC based, which use microprogramming to break the complex instructions into simpler sub-operations.
- The sub-operations are very similar to RISC instructions.
- So, at the instruction level the processor can be considered to be using microprogramming.
- At the lower (hardware) level, it may be considered to be using hardwired control to execute the RISC-like instructions in a pipeline.



CISC Instructions

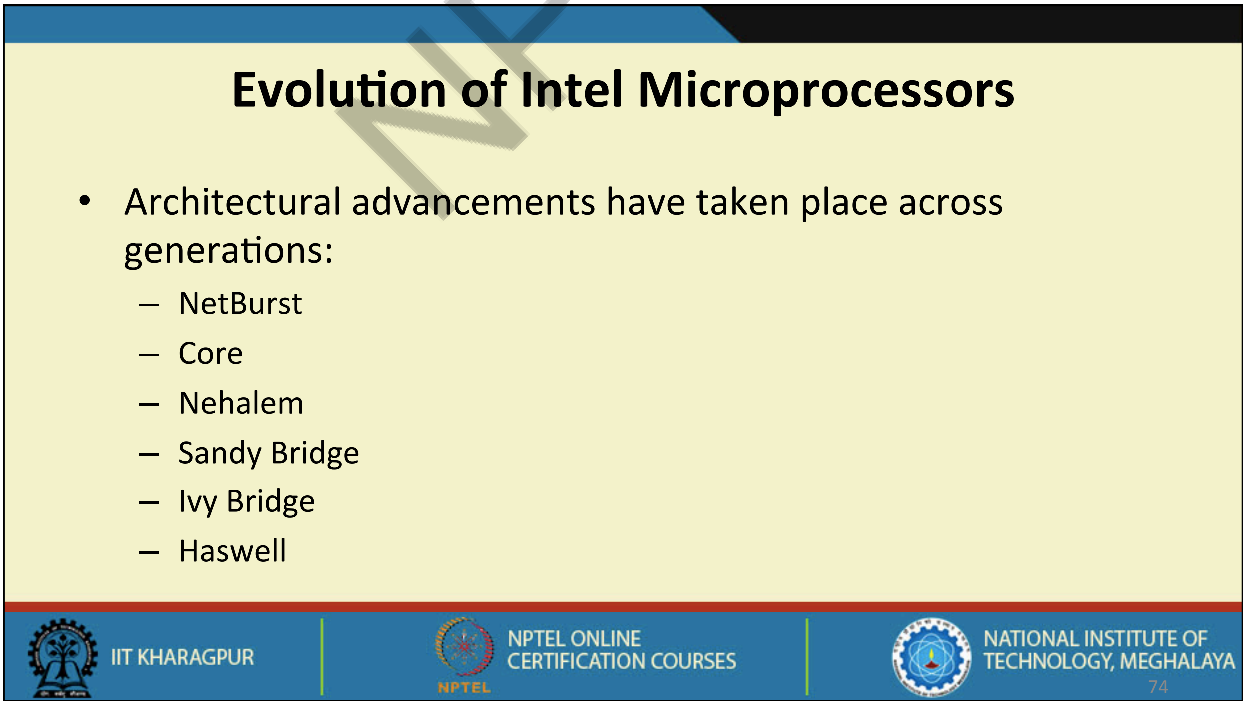
Microprogramming

RISC-like Instructions

Hardwired Control

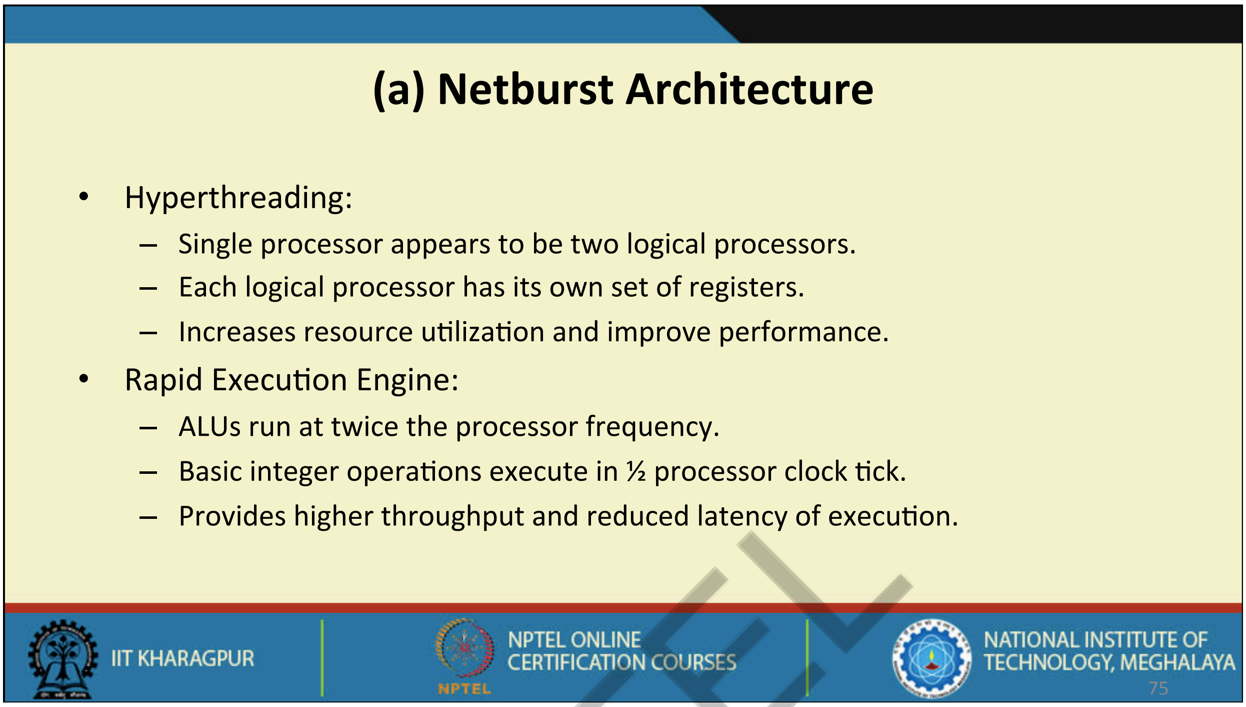
IF → ID → EX → MEM → WB

Pipeline



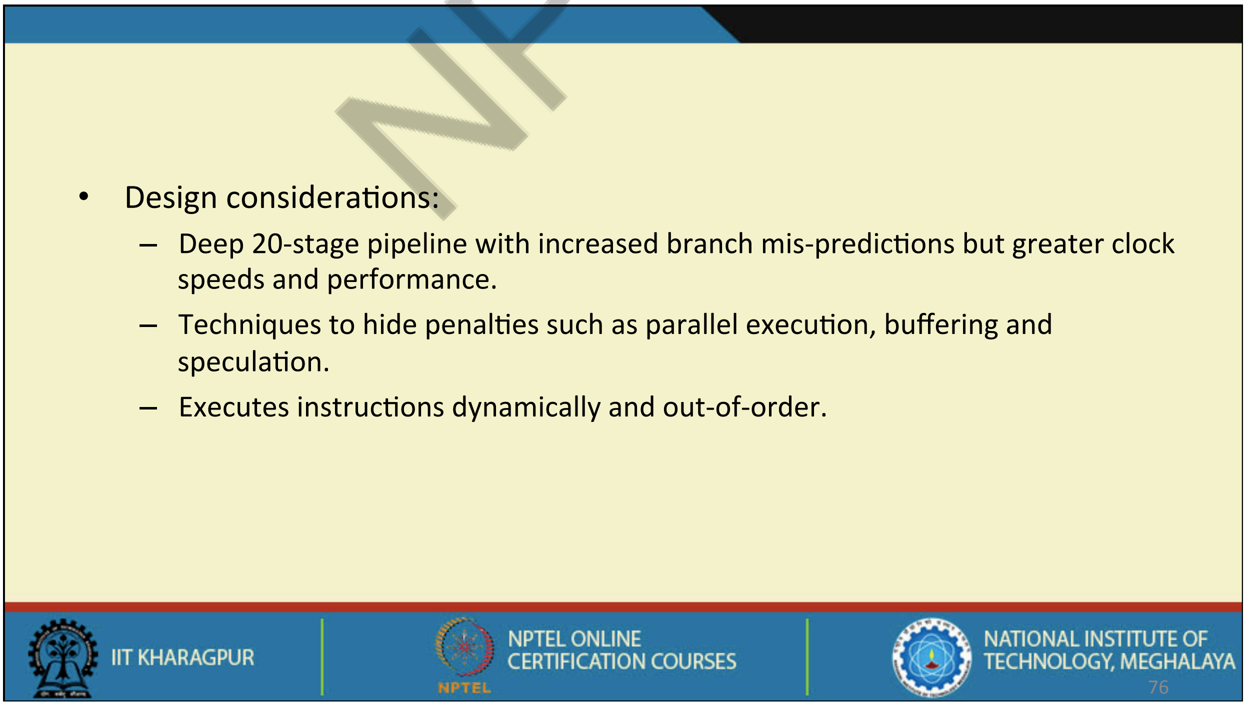
Evolution of Intel Microprocessors

- Architectural advancements have taken place across generations:
  - NetBurst
  - Core
  - Nehalem
  - Sandy Bridge
  - Ivy Bridge
  - Haswell

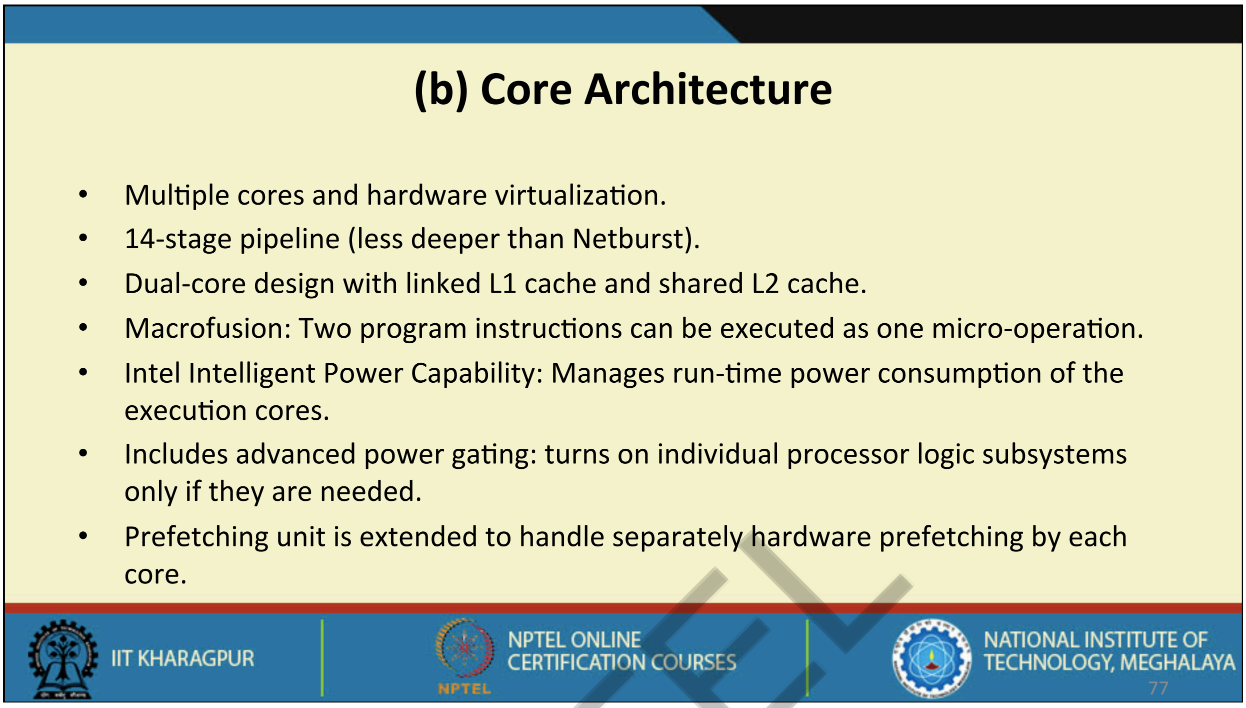


(a) Netburst Architecture

- Hyperthreading:
  - Single processor appears to be two logical processors.
  - Each logical processor has its own set of registers.
  - Increases resource utilization and improve performance.
- Rapid Execution Engine:
  - ALUs run at twice the processor frequency.
  - Basic integer operations execute in ½ processor clock tick.
  - Provides higher throughput and reduced latency of execution.

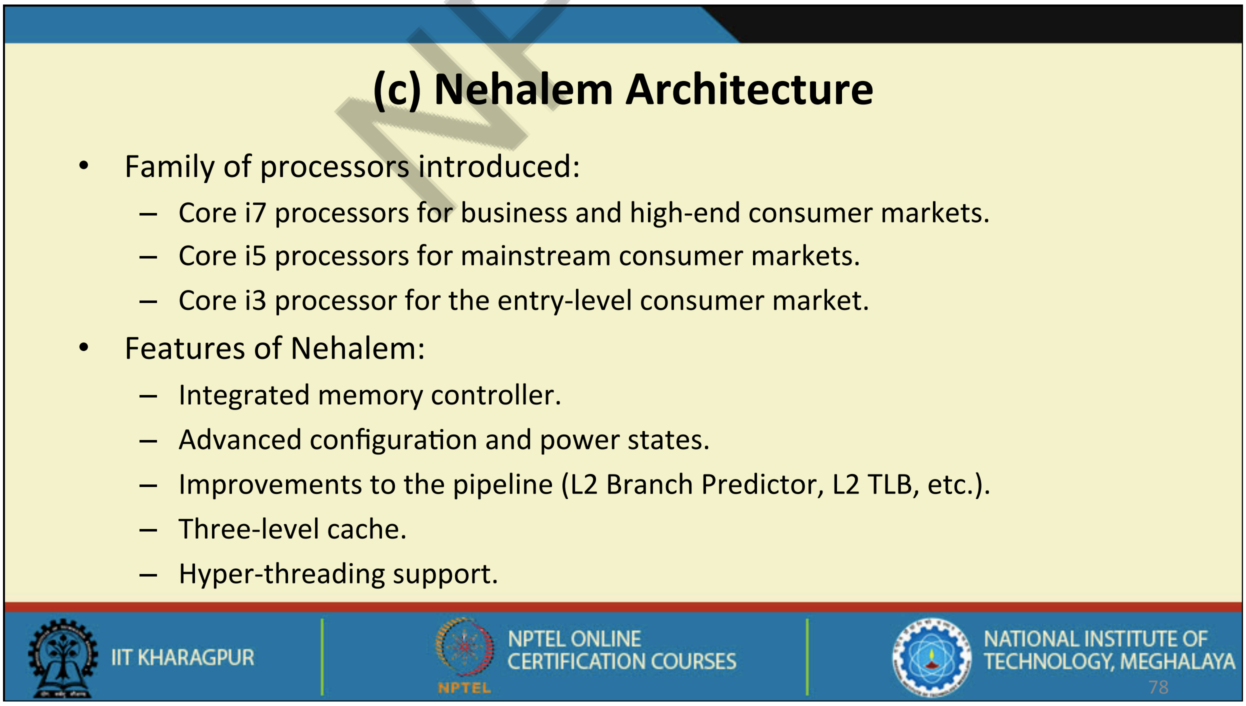


- Design considerations:
  - Deep 20-stage pipeline with increased branch mis-predictions but greater clock speeds and performance.
  - Techniques to hide penalties such as parallel execution, buffering and speculation.
  - Executes instructions dynamically and out-of-order.



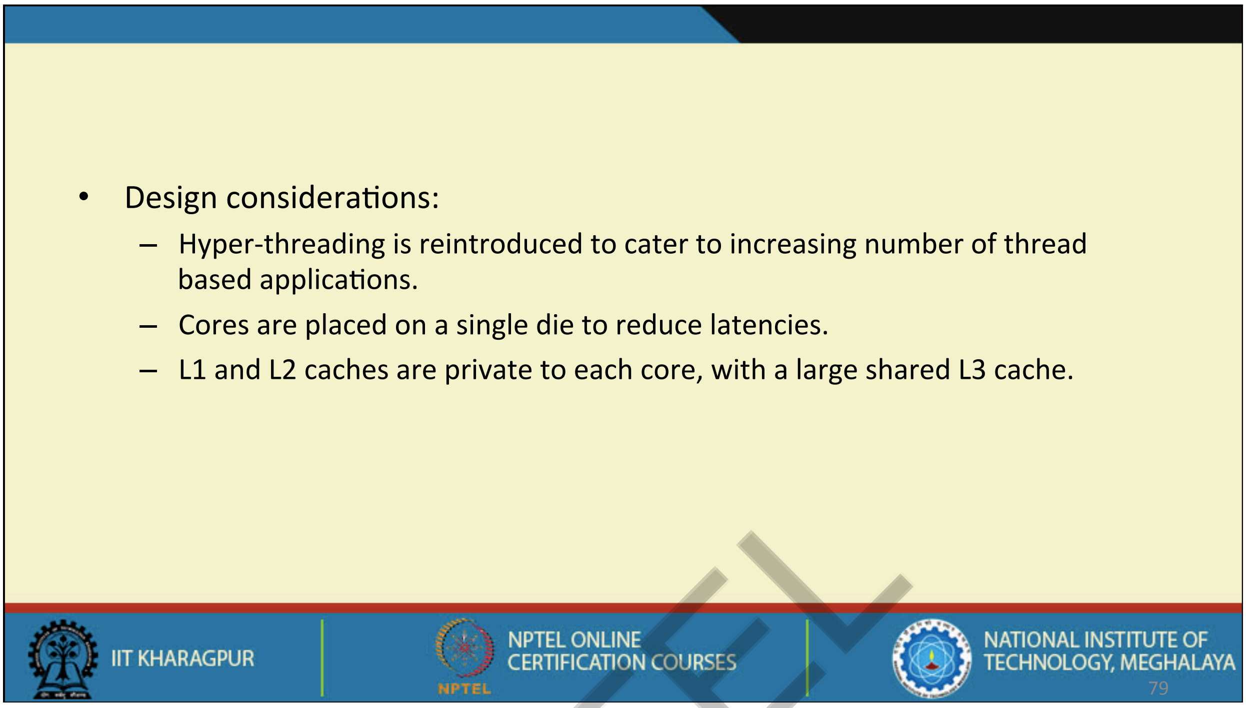
(b) Core Architecture

- Multiple cores and hardware virtualization.
- 14-stage pipeline (less deeper than Netburst).
- Dual-core design with linked L1 cache and shared L2 cache.
- Macrofusion: Two program instructions can be executed as one micro-operation.
- Intel Intelligent Power Capability: Manages run-time power consumption of the execution cores.
- Includes advanced power gating: turns on individual processor logic subsystems only if they are needed.
- Prefetching unit is extended to handle separately hardware prefetching by each core.

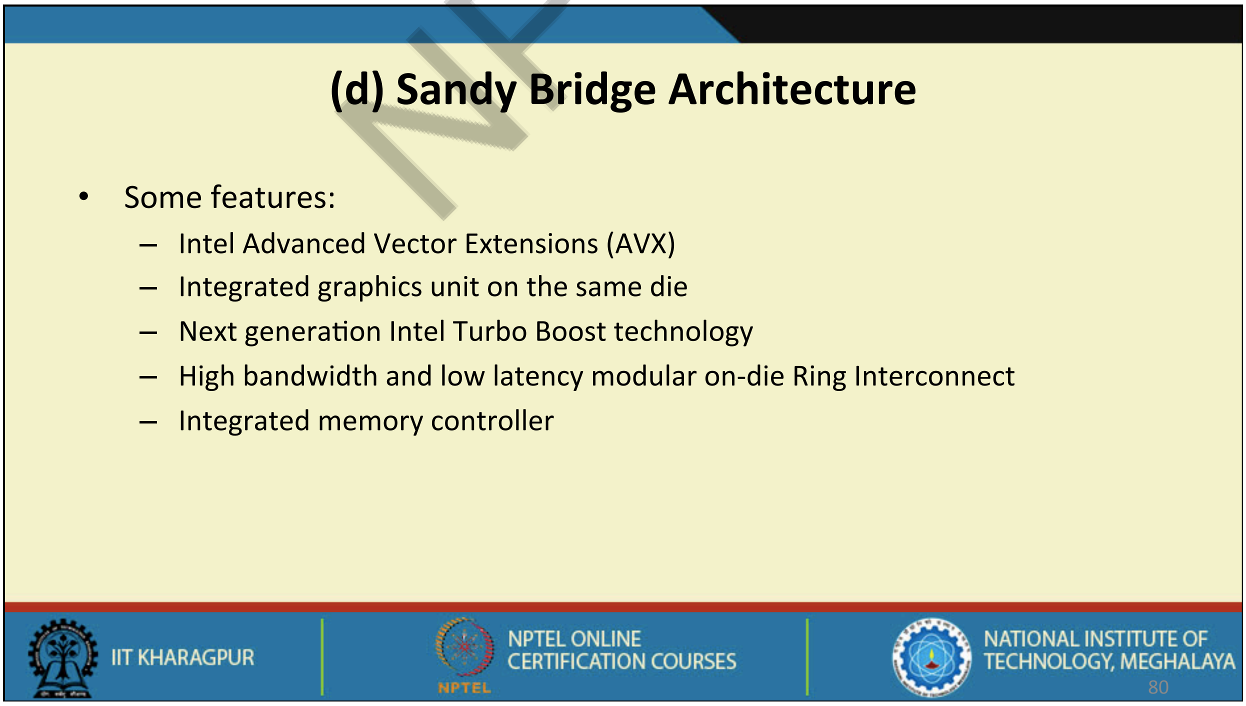


(c) Nehalem Architecture

- Family of processors introduced:
  - Core i7 processors for business and high-end consumer markets.
  - Core i5 processors for mainstream consumer markets.
  - Core i3 processor for the entry-level consumer market.
- Features of Nehalem:
  - Integrated memory controller.
  - Advanced configuration and power states.
  - Improvements to the pipeline (L2 Branch Predictor, L2 TLB, etc.).
  - Three-level cache.
  - Hyper-threading support.



- Design considerations:
  - Hyper-threading is reintroduced to cater to increasing number of thread based applications.
  - Cores are placed on a single die to reduce latencies.
  - L1 and L2 caches are private to each core, with a large shared L3 cache.



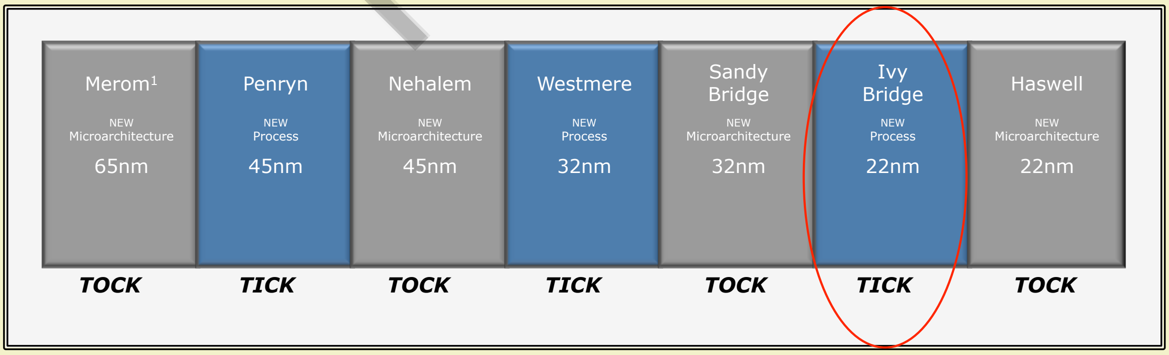
(d) Sandy Bridge Architecture

- Some features:
  - Intel Advanced Vector Extensions (AVX)
  - Integrated graphics unit on the same die
  - Next generation Intel Turbo Boost technology
  - High bandwidth and low latency modular on-die Ring Interconnect
  - Integrated memory controller

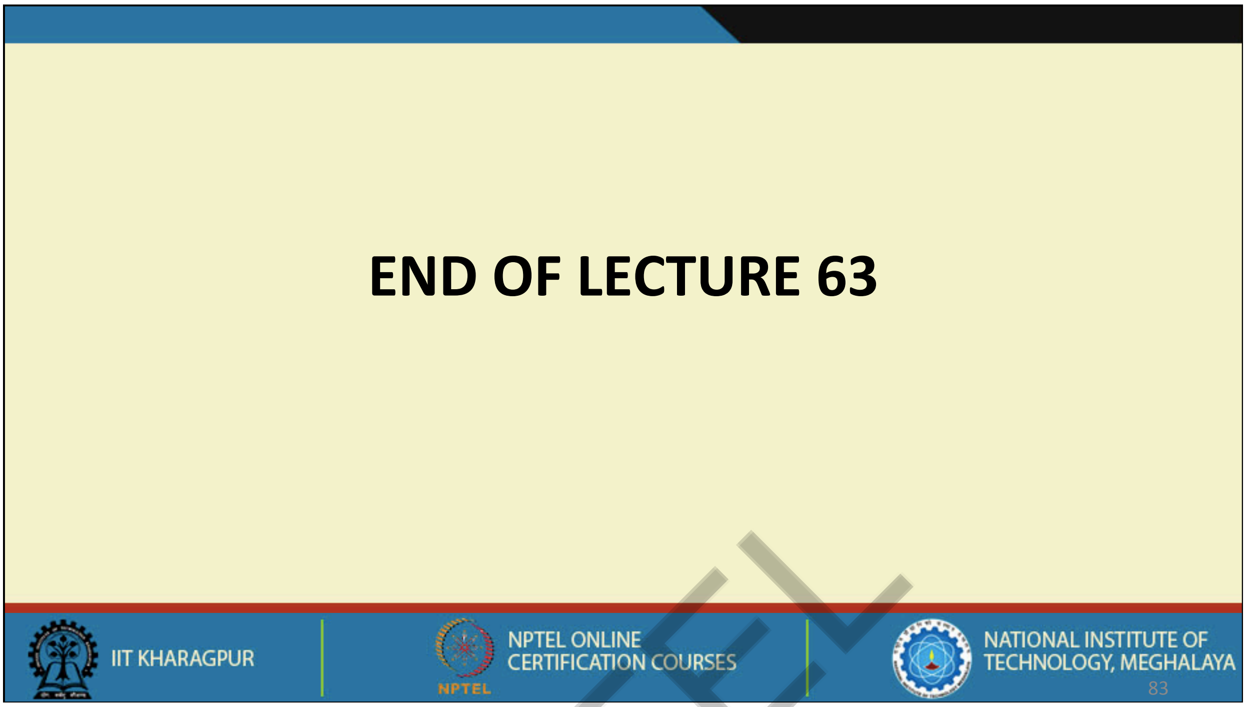
(e) Haswell Architecture

- Next generation branch prediction
- Improved front-end
  - Initialize TLB and cache misses speculatively
  - Handle cache misses in parallel to hide latency
  - Improved branch prediction
- Deeper buffers for more instruction parallelism
- More execution units, shorter latencies
- More load/store bandwidth for better prefretching

Intel’s Tick-Tock Development Model



Merom[1]	Penryn	Nehalem	Westmere	Sandy Bridge	Ivy Bridge	Haswell
NEW Microarchitecture	NEW Process	NEW Microarchitecture	NEW Process	NEW Microarchitecture	NEW Process	NEW Microarchitecture
65nm	45nm	45nm	32nm	32nm	22nm	22nm
TOCK	***TICK***	***TOCK***	***TICK***	***TOCK***	***TICK***	***TOCK***



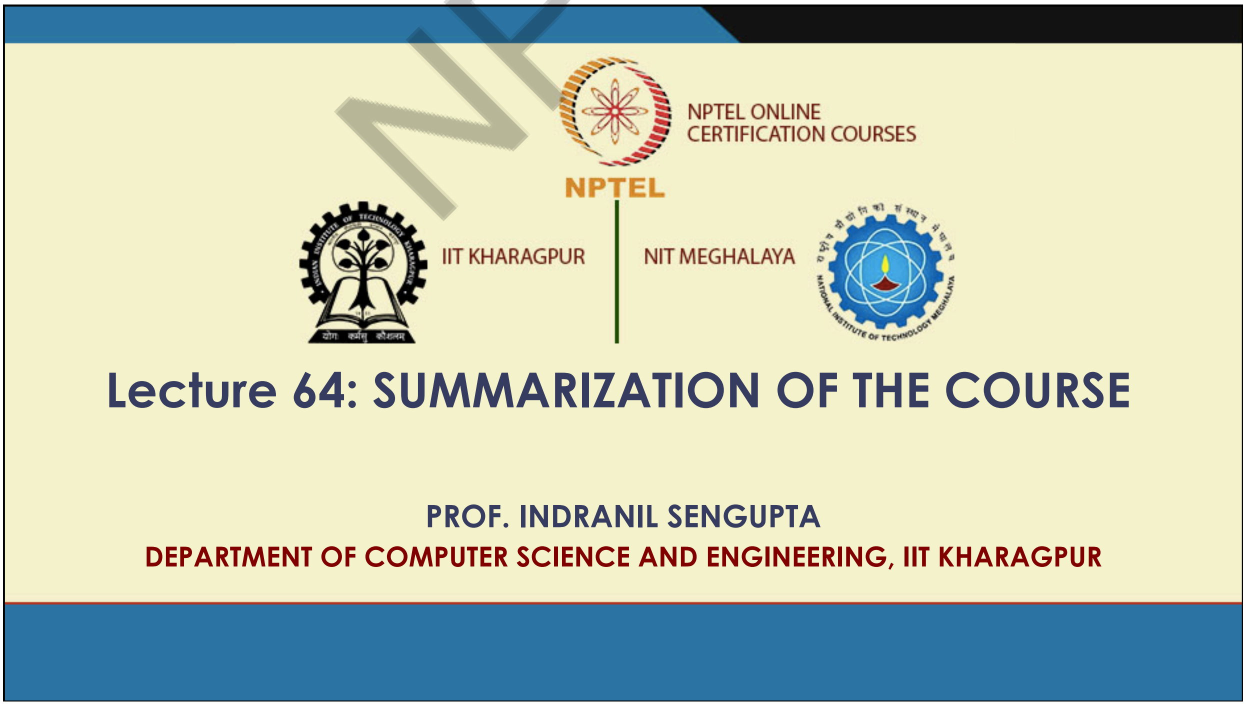
END OF LECTURE 63

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NATIONAL INSTITUTE OF
TECHNOLOGY, MEGHALAYA

83



NPTEL ONLINE
CERTIFICATION COURSES

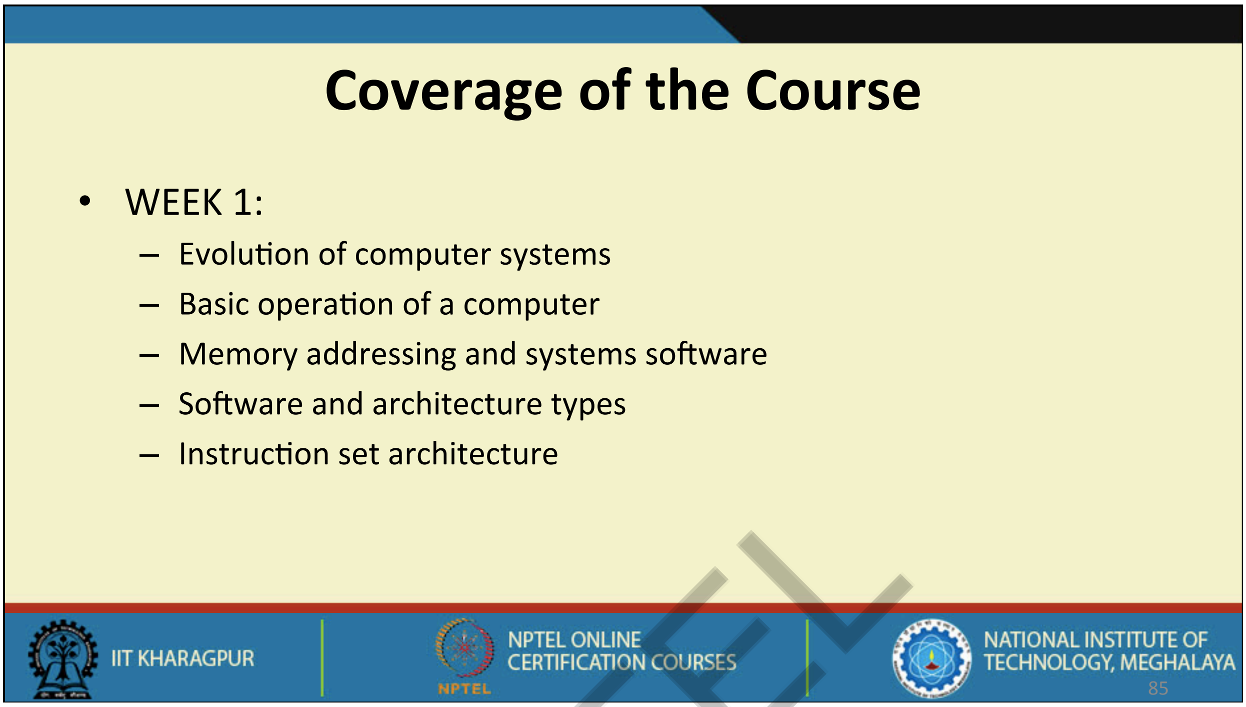
IIT KHARAGPUR

NIT MEGHALAYA

Lecture 64: SUMMARIZATION OF THE COURSE

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, IIT KHARAGPUR

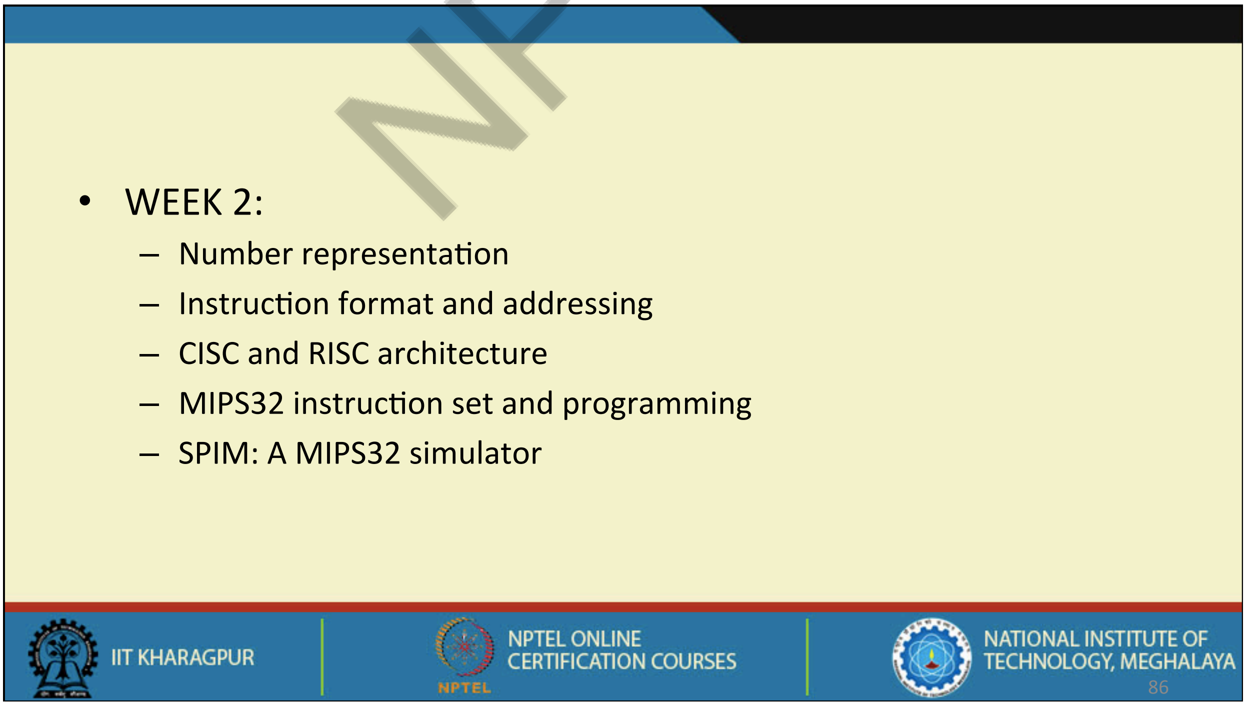


Coverage of the Course

- WEEK 1:
  - Evolution of computer systems
  - Basic operation of a computer
  - Memory addressing and systems software
  - Software and architecture types
  - Instruction set architecture

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

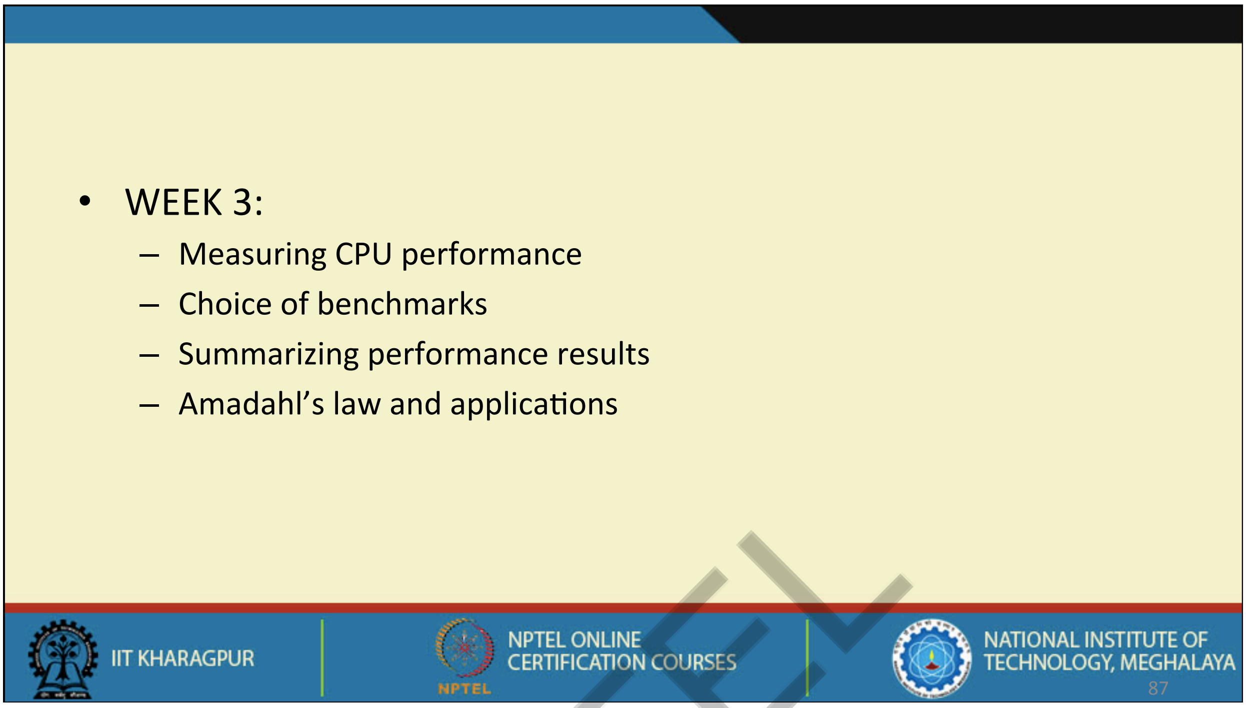
85



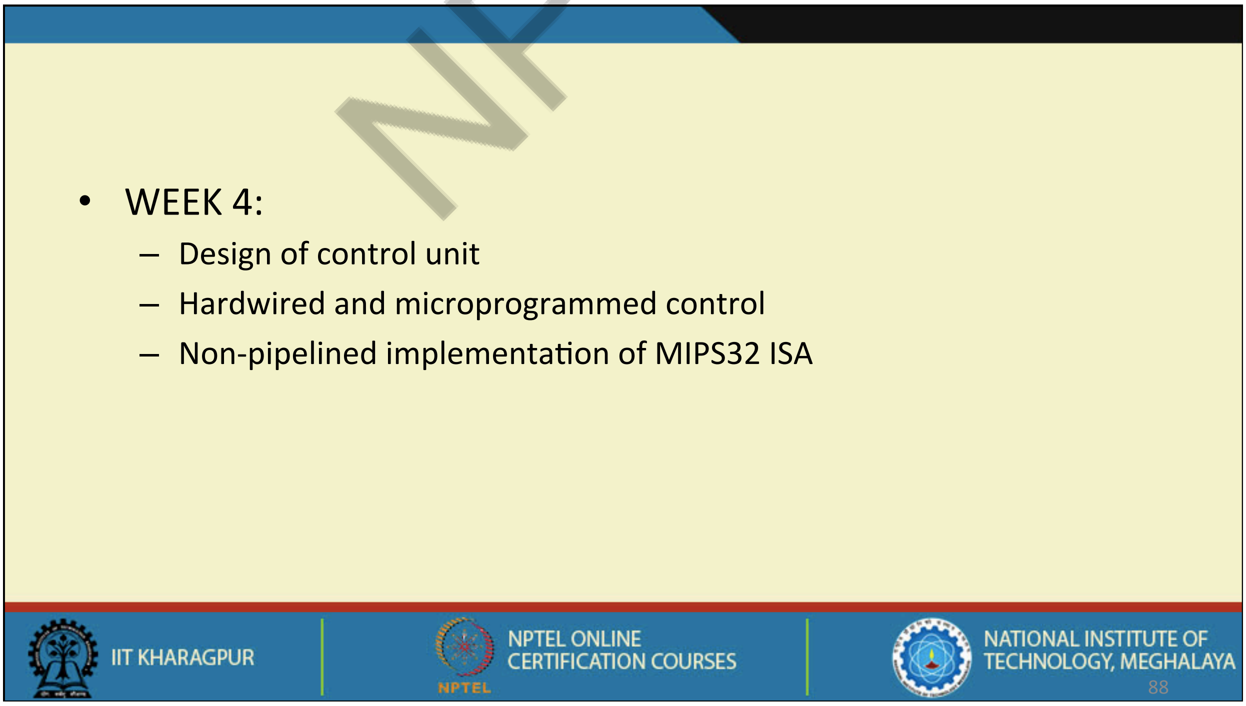
- WEEK 2:
  - Number representation
  - Instruction format and addressing
  - CISC and RISC architecture
  - MIPS32 instruction set and programming
  - SPIM: A MIPS32 simulator

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

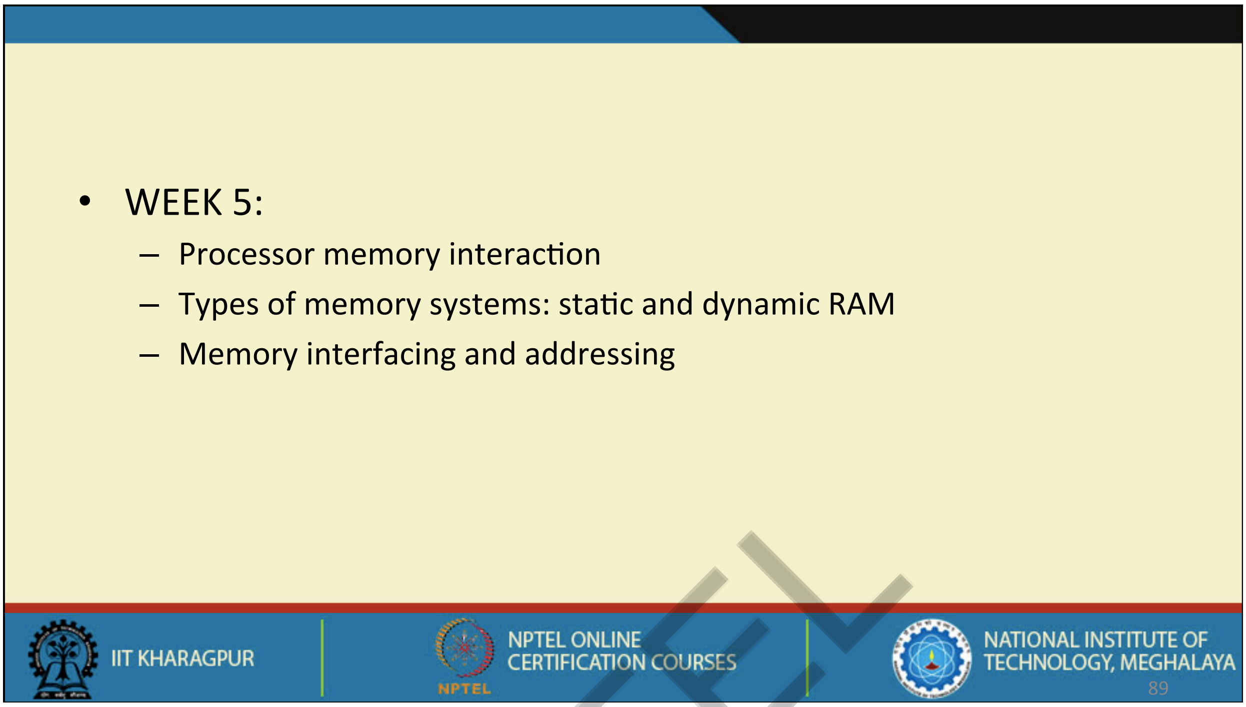
86



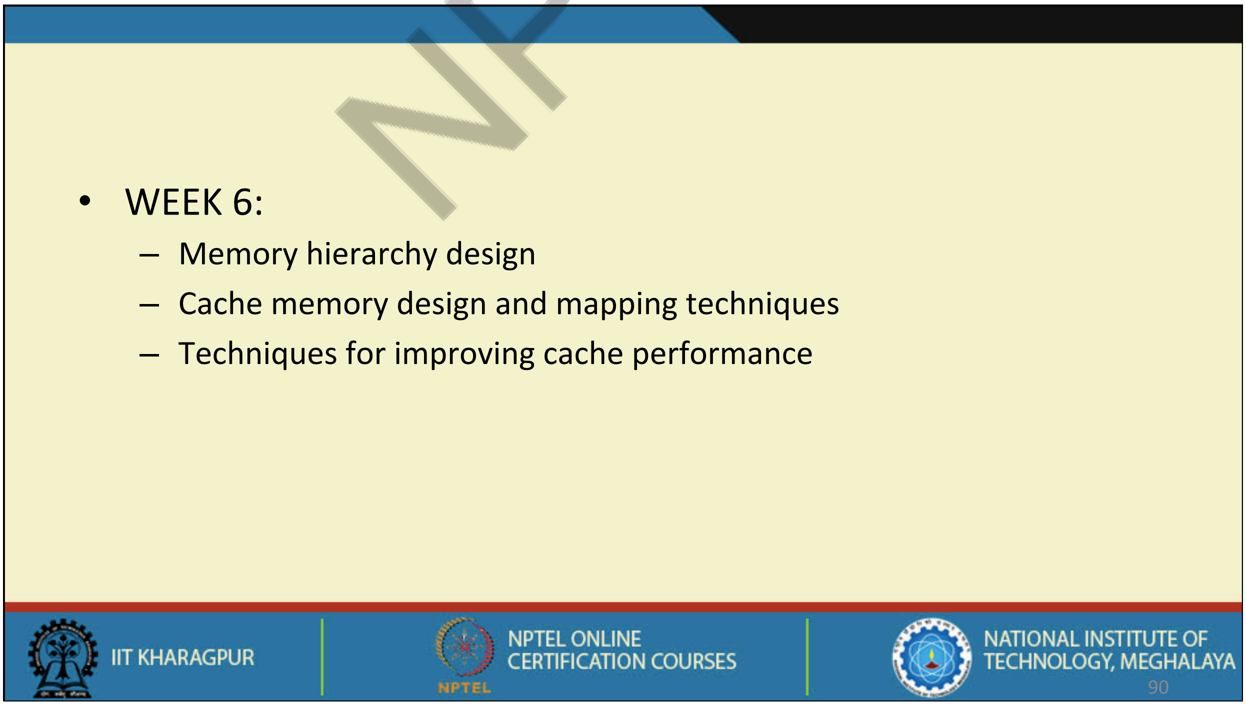
- WEEK 3:
  - Measuring CPU performance
  - Choice of benchmarks
  - Summarizing performance results
  - Amadahl’s law and applications



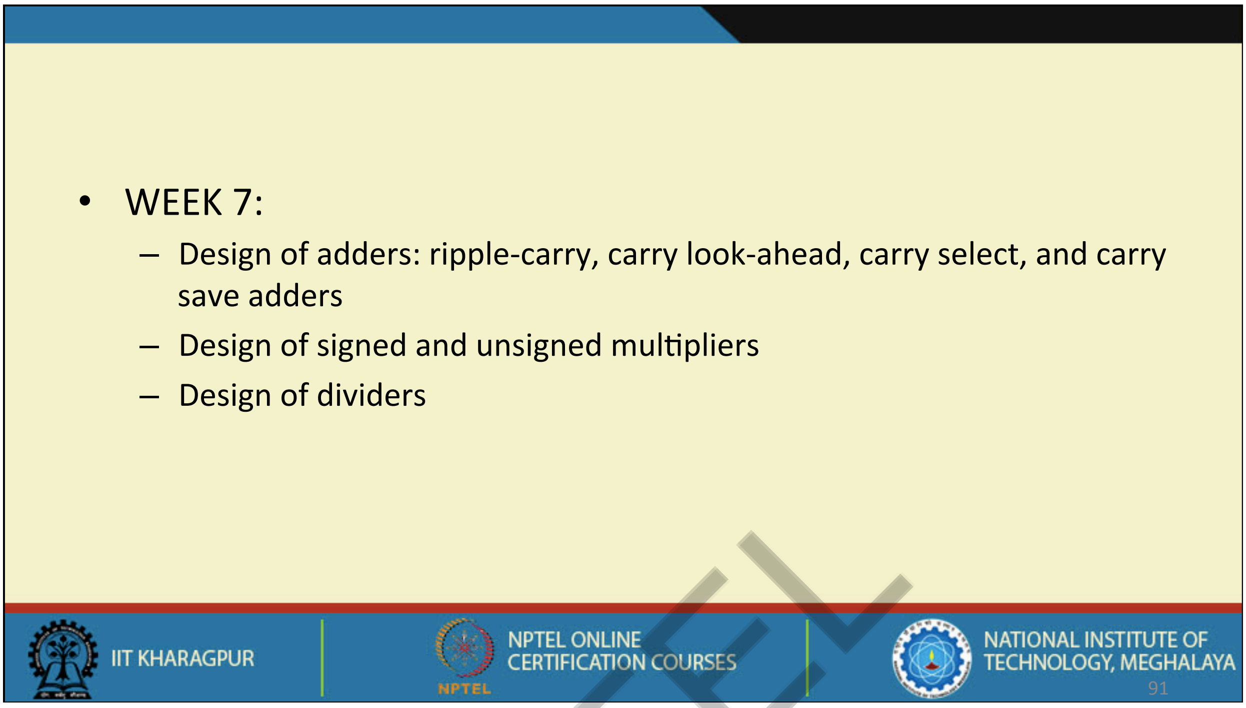
- WEEK 4:
  - Design of control unit
  - Hardwired and microprogrammed control
  - Non-pipelined implementation of MIPS32 ISA



- WEEK 5:
  - Processor memory interaction
  - Types of memory systems: static and dynamic RAM
  - Memory interfacing and addressing



- WEEK 6:
  - Memory hierarchy design
  - Cache memory design and mapping techniques
  - Techniques for improving cache performance



- WEEK 7:
  - Design of adders: ripple-carry, carry look-ahead, carry select, and carry save adders
  - Design of signed and unsigned multipliers
  - Design of dividers

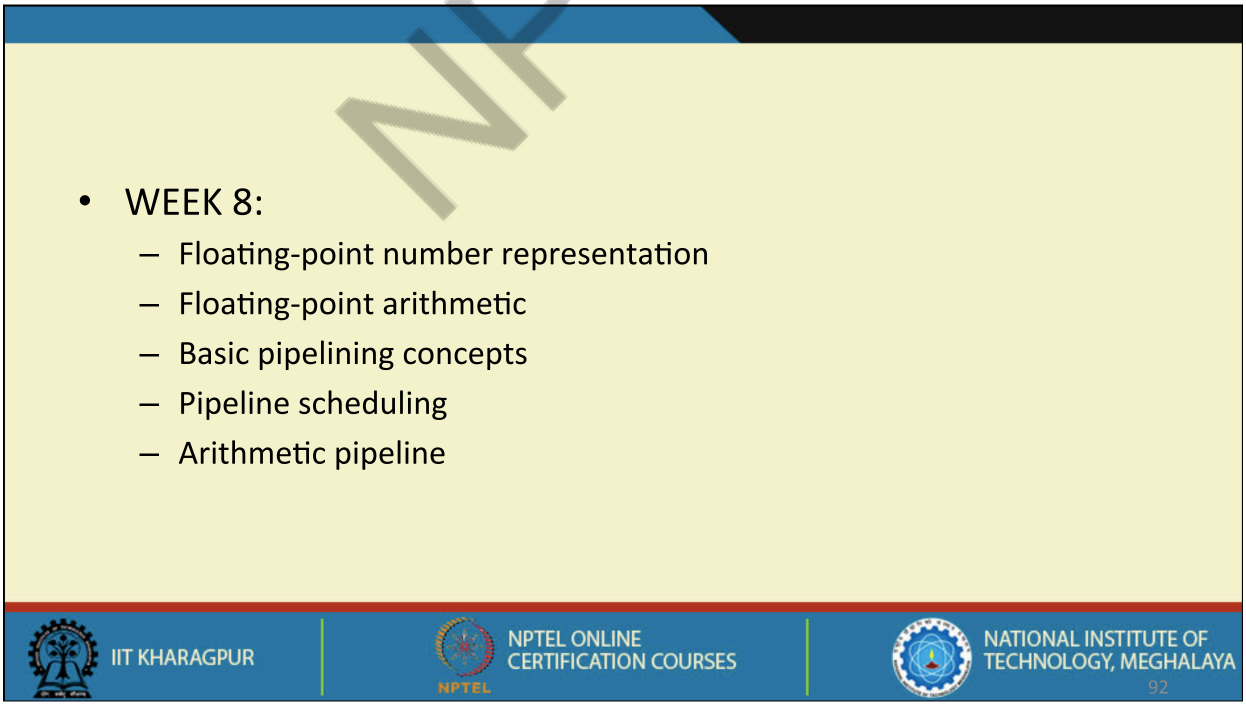
IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

NATIONAL INSTITUTE OF
TECHNOLOGY, MEGHALAYA

91



- WEEK 8:
  - Floating-point number representation
  - Floating-point arithmetic
  - Basic pipelining concepts
  - Pipeline scheduling
  - Arithmetic pipeline

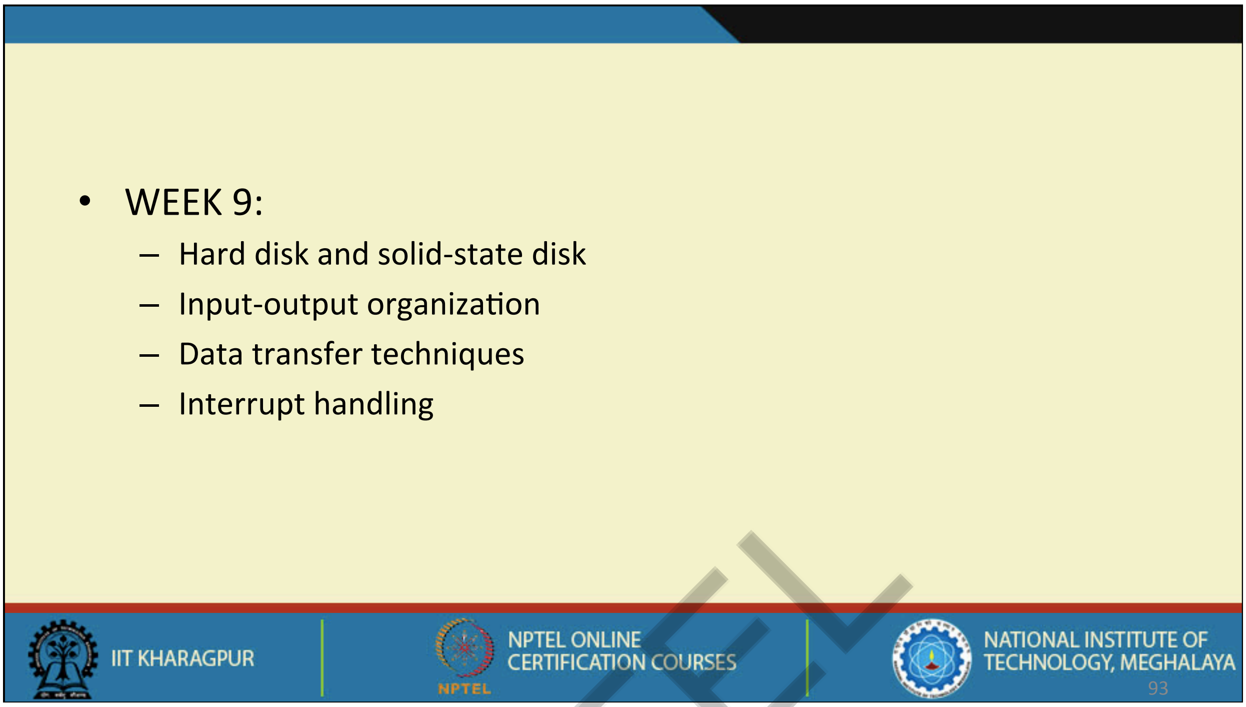
IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

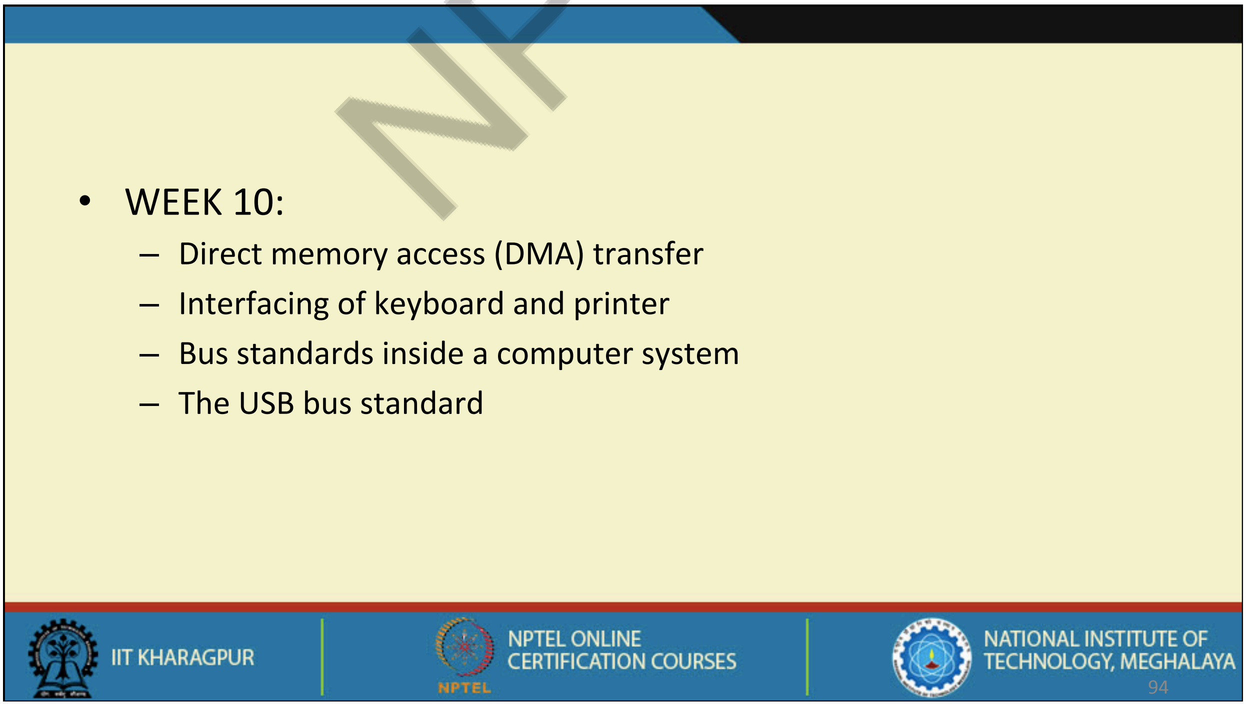
NPTEL

NATIONAL INSTITUTE OF
TECHNOLOGY, MEGHALAYA

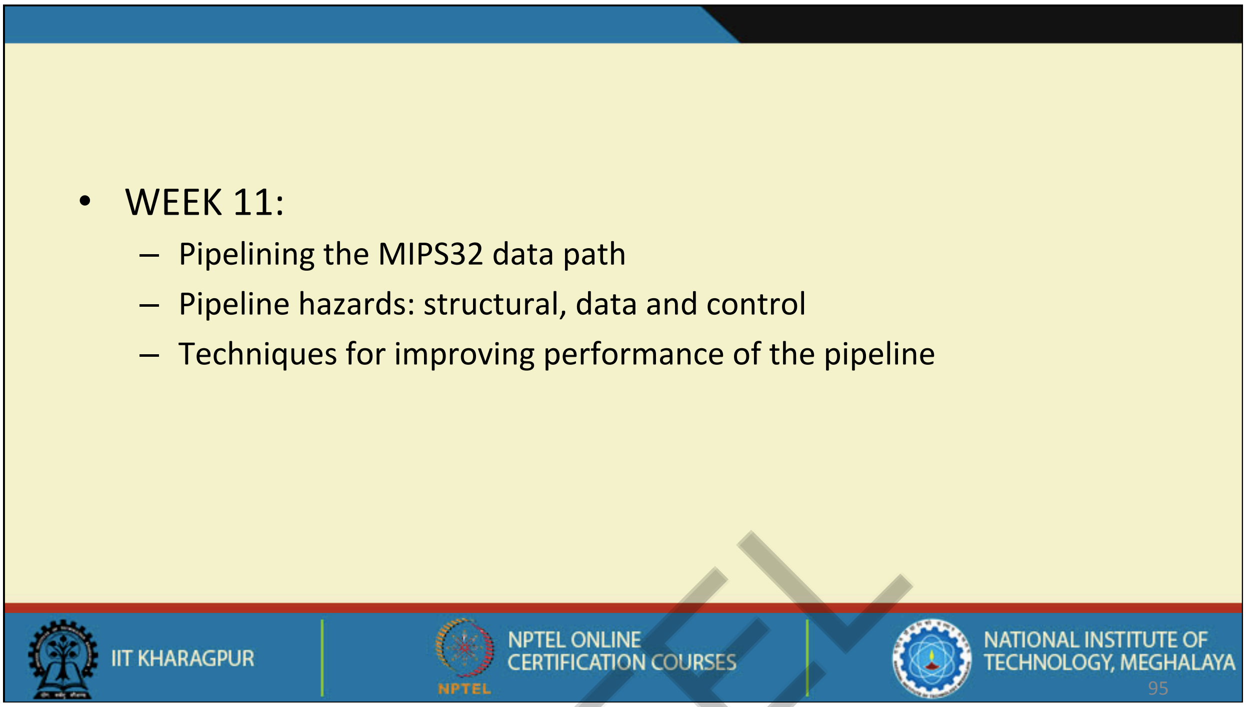
92



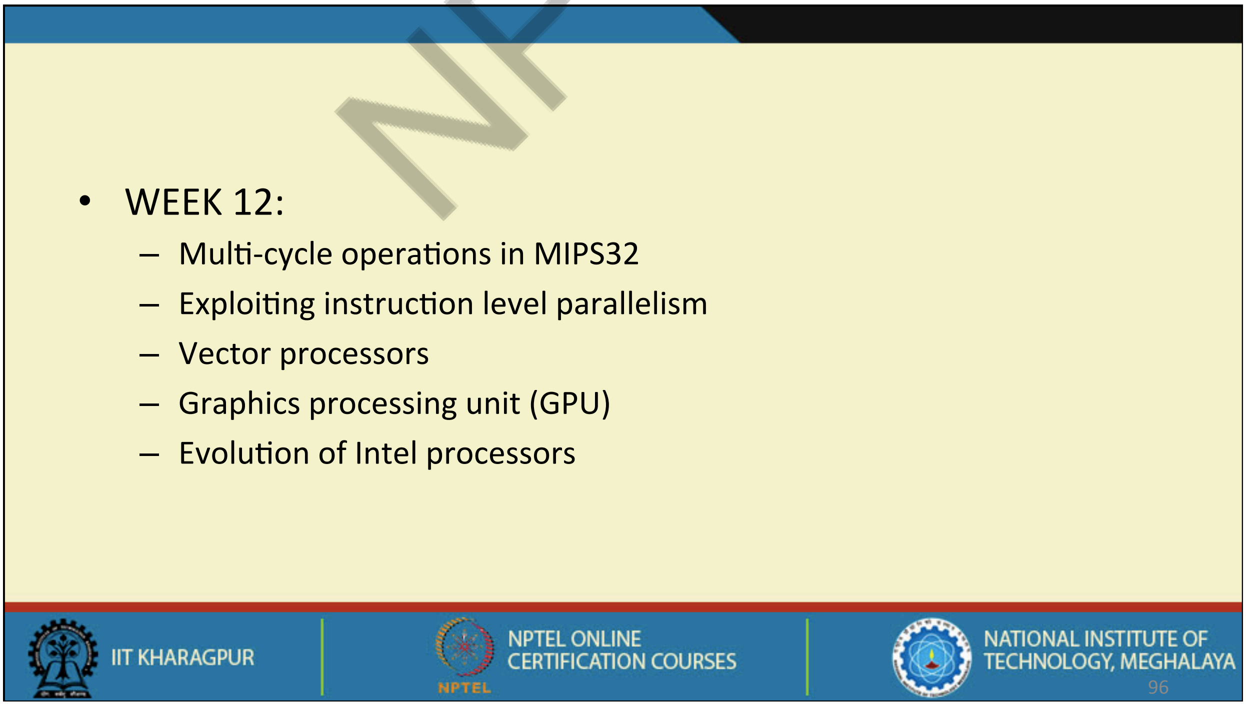
- WEEK 9:
  - Hard disk and solid-state disk
  - Input-output organization
  - Data transfer techniques
  - Interrupt handling



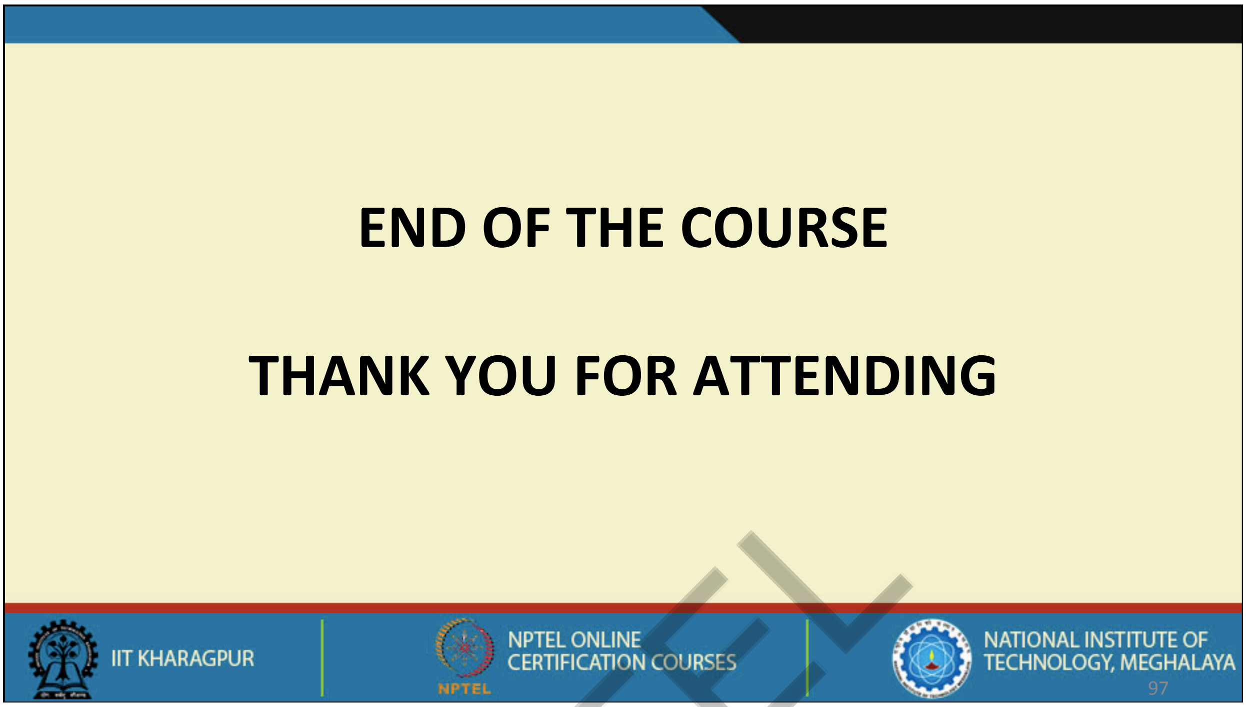
- WEEK 10:
  - Direct memory access (DMA) transfer
  - Interfacing of keyboard and printer
  - Bus standards inside a computer system
  - The USB bus standard



- WEEK 11:
  - Pipelining the MIPS32 data path
  - Pipeline hazards: structural, data and control
  - Techniques for improving performance of the pipeline



- WEEK 12:
  - Multi-cycle operations in MIPS32
  - Exploiting instruction level parallelism
  - Vector processors
  - Graphics processing unit (GPU)
  - Evolution of Intel processors



END OF THE COURSE

THANK YOU FOR ATTENDING

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NATIONAL INSTITUTE OF
TECHNOLOGY, MEGHALAYA

97