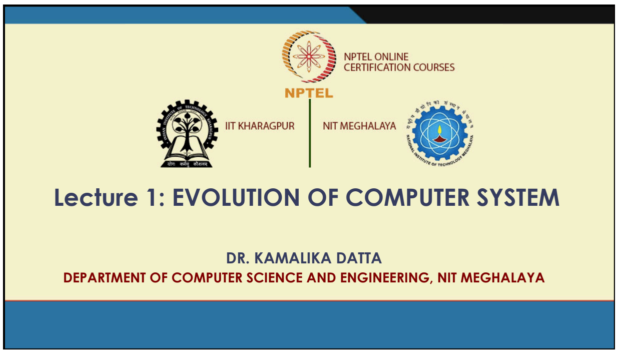


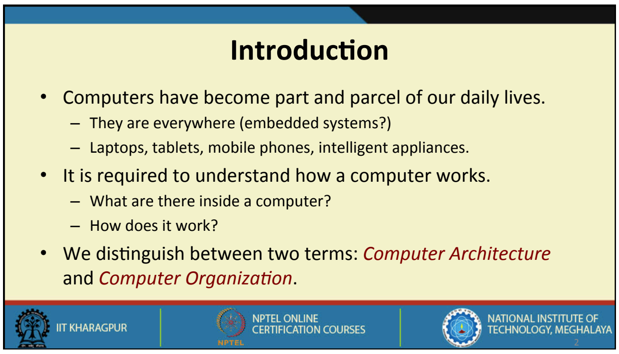
NPTEL ONLINE CERTIFICATION COURSES

NPTEL

IIT KHARAGPUR | NIT MEGHALAYA

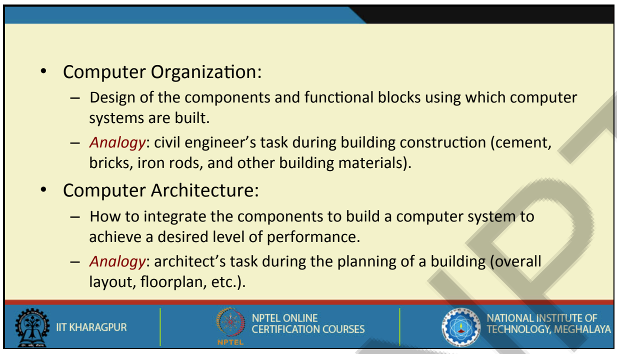
Lecture 1: EVOLUTION OF COMPUTER SYSTEM

DR. KAMALIKA DATTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, NIT MEGHALAYA

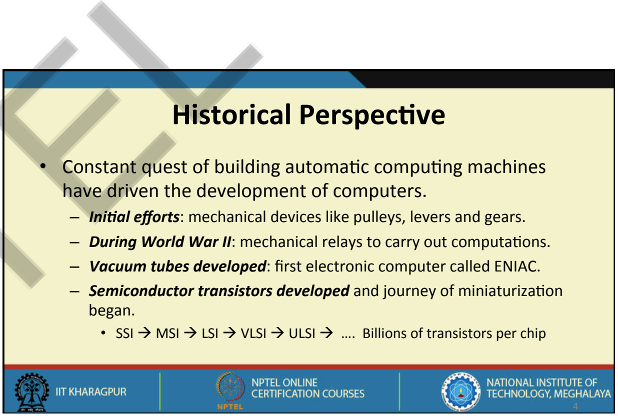


Introduction

- Computers have become part and parcel of our daily lives.
  - They are everywhere (embedded systems?)
  - Laptops, tablets, mobile phones, intelligent appliances.
- It is required to understand how a computer works.
  - What are there inside a computer?
  - How does it work?
- We distinguish between two terms: *Computer Architecture* and *Computer Organization*.

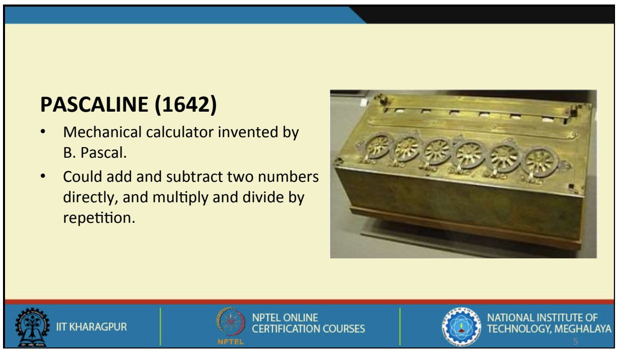


- Computer Organization:
  - Design of the components and functional blocks using which computer systems are built.
  - *Analogy*: civil engineer's task during building construction (cement, bricks, iron rods, and other building materials).
- Computer Architecture:
  - How to integrate the components to build a computer system to achieve a desired level of performance.
  - *Analogy*: architect's task during the planning of a building (overall layout, floorplan, etc.).



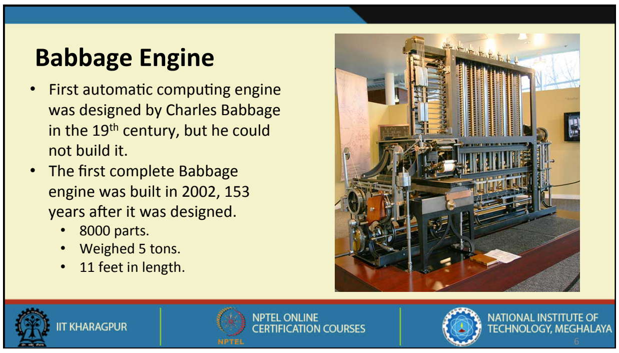
Historical Perspective

- Constant quest of building automatic computing machines have driven the development of computers.
  - ***Initial efforts***: mechanical devices like pulleys, levers and gears.
  - ***During World War II***: mechanical relays to carry out computations.
  - ***Vacuum tubes developed***: first electronic computer called ENIAC.
  - ***Semiconductor transistors developed*** and journey of miniaturization began.
    - SSI → MSI → LSI → VLSI → ULSI → .... Billions of transistors per chip



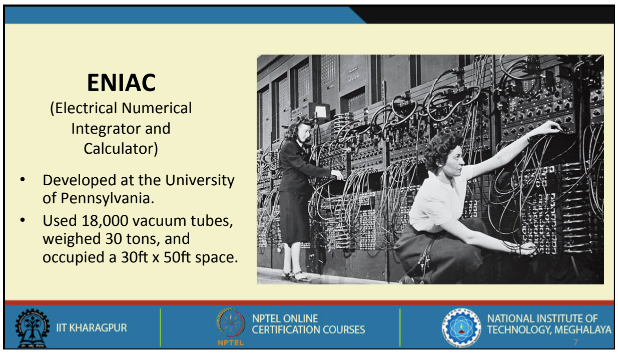
PASCALINE (1642)

- Mechanical calculator invented by B. Pascal.
- Could add and subtract two numbers directly, and multiply and divide by repetition.



Babbage Engine

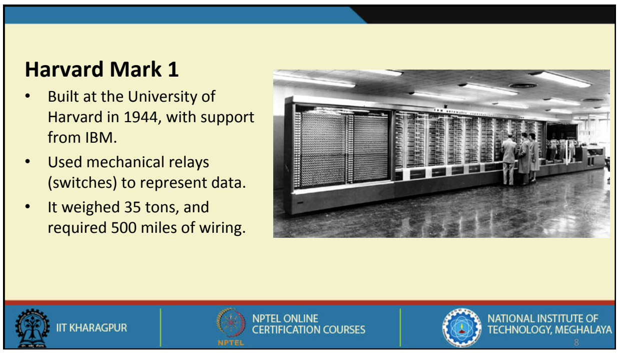
- First automatic computing engine was designed by Charles Babbage in the 19th century, but he could not build it.
- The first complete Babbage engine was built in 2002, 153 years after it was designed.
  - 8000 parts.
  - Weighed 5 tons.
  - 11 feet in length.



ENIAC

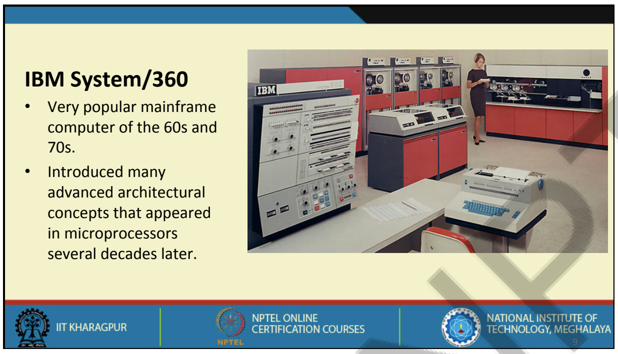
(Electrical Numerical Integrator and Calculator)

- Developed at the University of Pennsylvania.
- Used 18,000 vacuum tubes, weighed 30 tons, and occupied a 30ft x 50ft space.



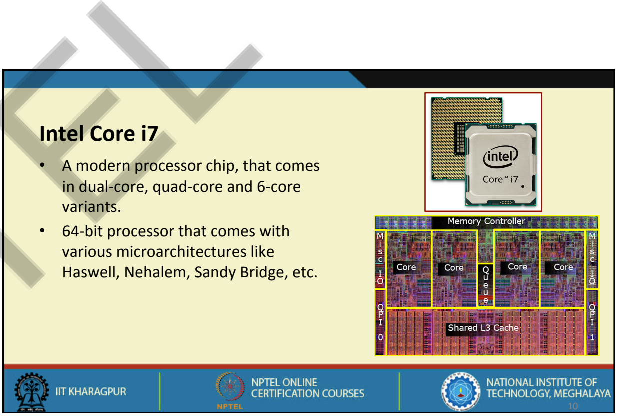
Harvard Mark 1

- Built at the University of Harvard in 1944, with support from IBM.
- Used mechanical relays (switches) to represent data.
- It weighed 35 tons, and required 500 miles of wiring.



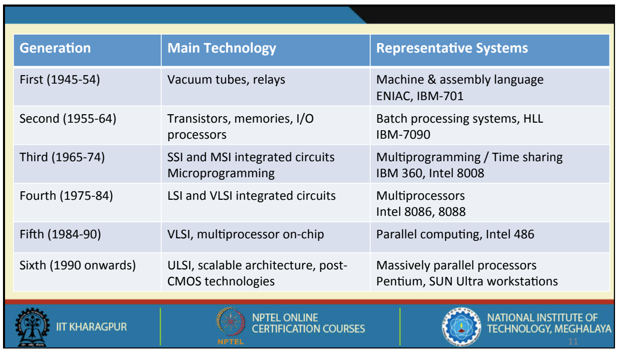
IBM System/360

- Very popular mainframe computer of the 60s and 70s.
- Introduced many advanced architectural concepts that appeared in microprocessors several decades later.

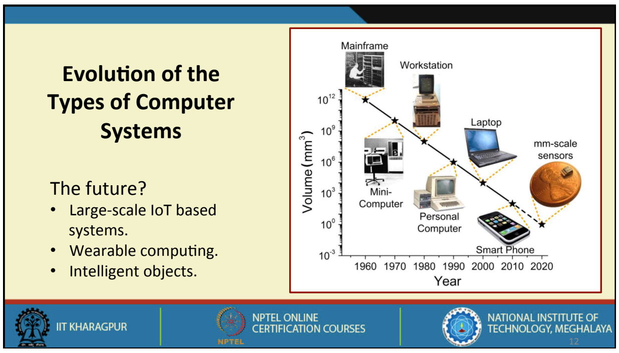


Intel Core i7

- A modern processor chip, that comes in dual-core, quad-core and 6-core variants.
- 64-bit processor that comes with various microarchitectures like Haswell, Nehalem, Sandy Bridge, etc.



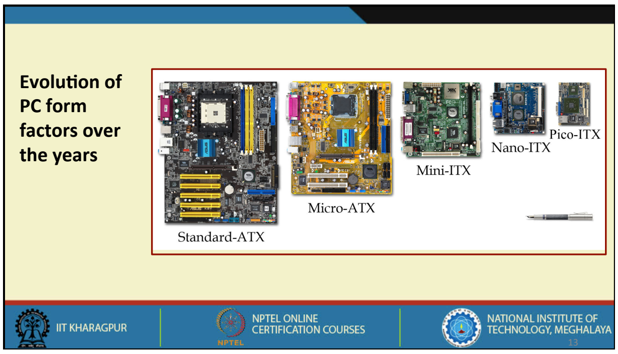
Generation	Main Technology	Representative Systems
First (1945-54)	Vacuum tubes, relays	Machine & assembly language ENIAC, IBM-701
Second (1955-64)	Transistors, memories, I/O processors	Batch processing systems, HLL IBM-7090
Third (1965-74)	SSI and MSI integrated circuits Microprogramming	Multiprogramming / Time sharing IBM 360, Intel 8008
Fourth (1975-84)	LSI and VLSI integrated circuits	Multiprocessors Intel 8086, 8088
Fifth (1984-90)	VLSI, multiprocessor on-chip	Parallel computing, Intel 486
Sixth (1990 onwards)	ULSI, scalable architecture, post-CMOS technologies	Massively parallel processors Pentium, SUN Ultra workstations



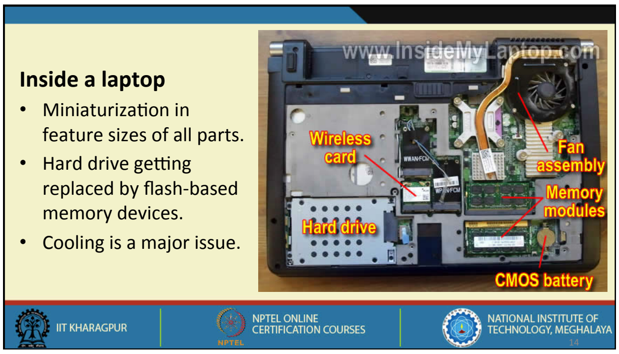
Evolution of the Types of Computer Systems

The future?

- Large-scale IoT based systems.
- Wearable computing.
- Intelligent objects.



Evolution of PC form factors over the years

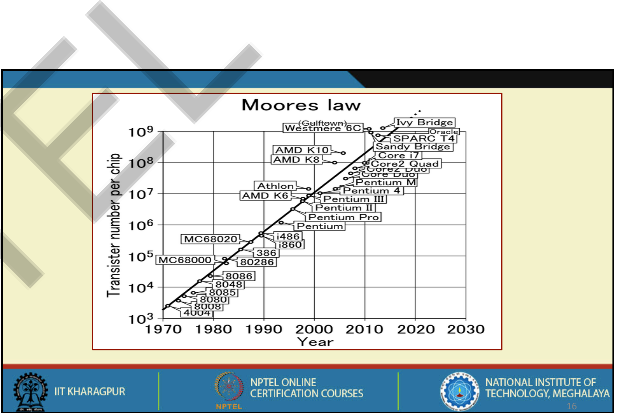


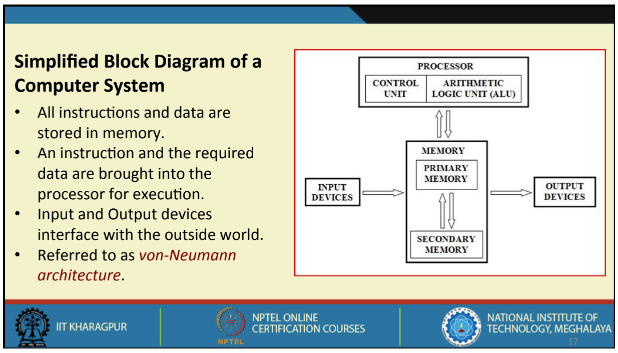
Inside a laptop

- Miniaturization in feature sizes of all parts.
- Hard drive getting replaced by flash-based memory devices.
- Cooling is a major issue.

Moore's Law

- Refers to an observation made by Intel co-founder Gordon Moore in 1965. He noticed that the number of transistors per square inch on integrated circuits had doubled every year since their invention.
- Moore's law predicts that this trend will continue into the foreseeable future.
- Although the pace has slowed, the number of transistors per square inch has since doubled approximately every 18 months. This is used as the current definition of Moore's law.





Simplified Block Diagram of a Computer System

- All instructions and data are stored in memory.
- An instruction and the required data are brought into the processor for execution.
- Input and Output devices interface with the outside world.
- Referred to as *von-Neumann architecture*.

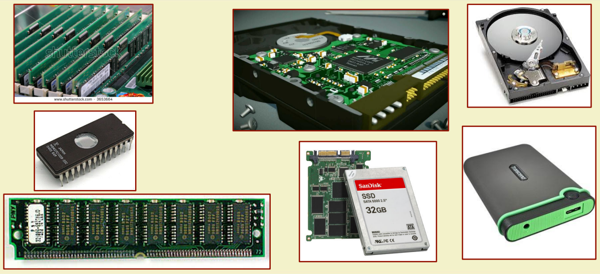
- **Inside the Processor**
  - Also called *Central Processing Unit* (CPU).
  - Consists of a *Control Unit* and an *Arithmetic Logic Unit* (ALU).
    - All calculations happen inside the ALU.
    - The Control Unit generates sequence of control signals to carry out all operations.
  - The processor fetches an instruction from memory for execution.
    - An instruction specifies the exact operation to be carried out.
    - It also specifies the data that are to be operated on.
    - A program refers to a set of instructions that are required to carry out some specific task (e.g. sorting a set of numbers).

- What is the role of ALU?
  - It contains several registers, some general-purpose and some special-purpose, for temporary storage of data.
  - It contains circuitry to carry out logic operations, like AND, OR, NOT, shift, compare, etc.
  - It contains circuitry to carry out arithmetic operations like addition, subtraction, multiplication, division, etc.
  - During instruction execution, the data (operands) are brought in and stored in some registers, the desired operation carried out, and the result stored back in some register or memory.

- What is the role of control unit?
  - Acts as the nerve center that senses the states of various functional units and sends control signals to control their states.
  - To carry out a specific operation (say, R1 ← R2 + R3), the control unit must generate control signals in a specific sequence.
    - Enable the outputs of registers R2 and R3.
    - Select the addition operation.
    - Store the output of the adder circuit into register R1.
  - When an instruction is fetched from memory, the operation (called *opcode*) is decoded by the control unit, and the control signals issued.

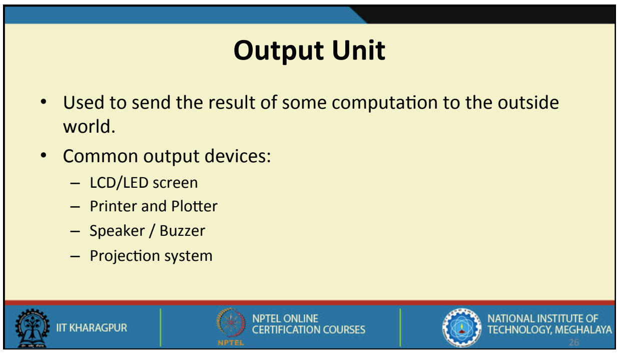
- **Inside the Memory Unit**
  - Two main types of memory subsystems.
    - *Primary or Main memory*, which stores the active instructions and data for the program being executed on the processor.
    - *Secondary memory*, which is used as a backup and stores all active and inactive programs and data, typically as *files*.
  - The processor only has direct access to the primary memory.
  - In reality, the memory system is implemented as a hierarchy of several levels.
    - L1 cache, L2 cache, L3 cache, primary memory, secondary memory.
    - Objective is to provide faster memory access at affordable cost.

- Various different types of memory are possible.
  a) Random Access Memory (RAM), which is used for the cache and primary memory sub-systems. Read and Write access times are independent of the location being accessed.
  b) Read Only Memory (ROM), which is used as part of the primary memory to store some fixed data that cannot be changed.
  c) Magnetic Disk, which uses direction of magnetization of tiny magnetic particles on a metallic surface to store data. Access times vary depending on the location being accessed, and is used as secondary memory.
  d) Flash Memory, which is replacing magnetic disks as secondary memory devices. They are faster, but smaller in size as compared to disk.



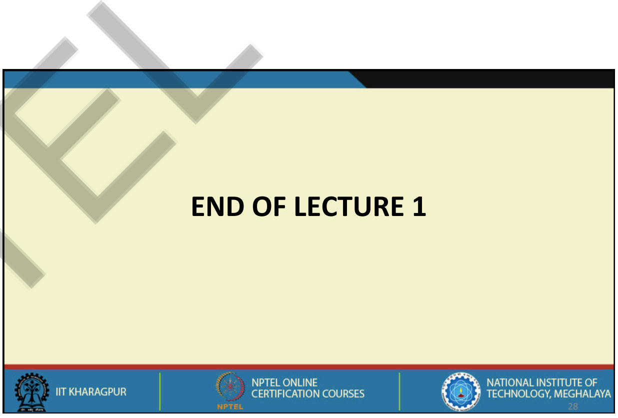
Input Unit

- Used to feed data to the computer system from the external environment.
  - Data are transferred to the processor/memory after appropriate encoding.
- Common input devices:
  - Keyboard
  - Mouse
  - Joystick
  - Camera

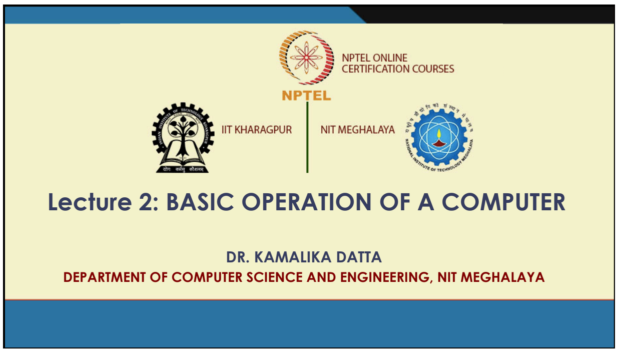


Output Unit

- Used to send the result of some computation to the outside world.
- Common output devices:
  - LCD/LED screen
  - Printer and Plotter
  - Speaker / Buzzer
  - Projection system



END OF LECTURE 1

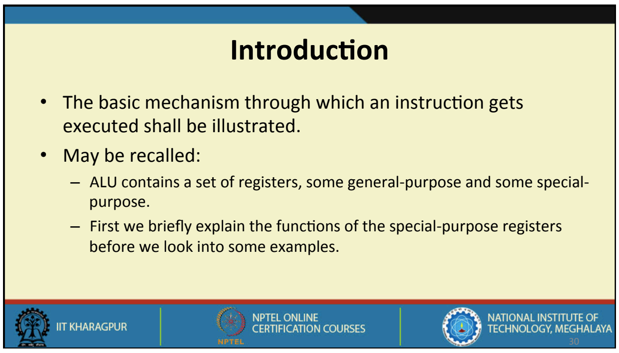


NPTEL ONLINE CERTIFICATION COURSES

IIT KHARAGPUR | NIT MEGHALAYA

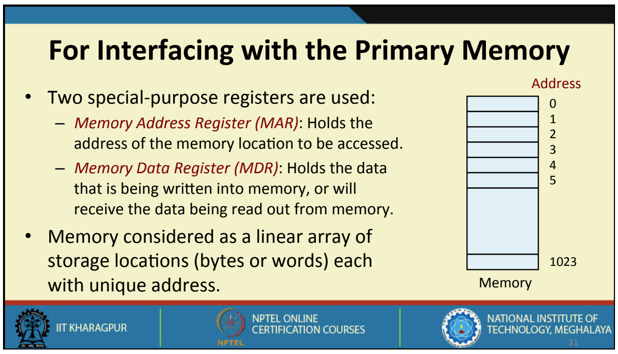
Lecture 2: BASIC OPERATION OF A COMPUTER

DR. KAMALIKA DATTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, NIT MEGHALAYA



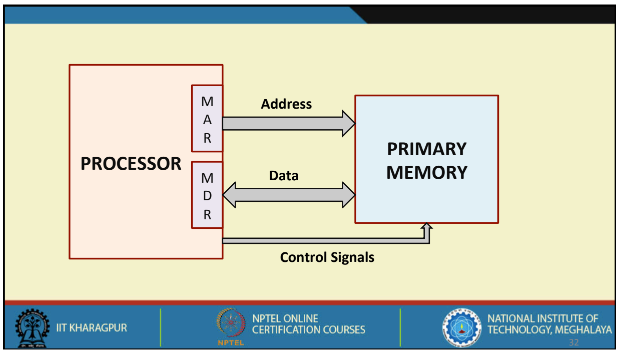
Introduction

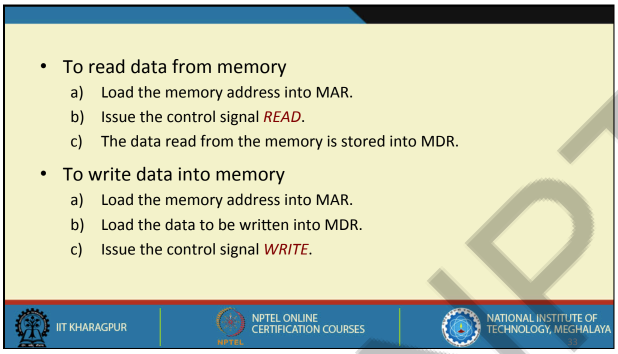
- The basic mechanism through which an instruction gets executed shall be illustrated.
- May be recalled:
  - ALU contains a set of registers, some general-purpose and some special-purpose.
  - First we briefly explain the functions of the special-purpose registers before we look into some examples.



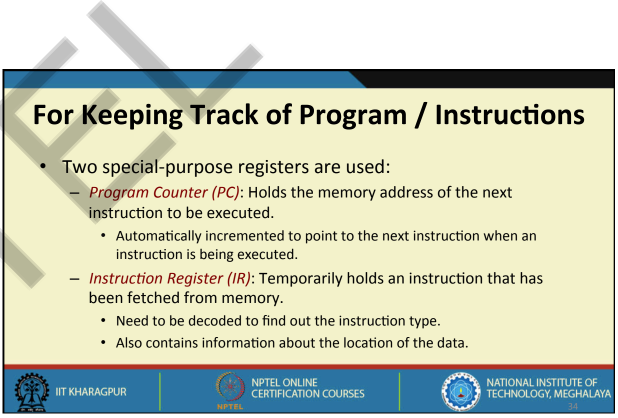
For Interfacing with the Primary Memory

- Two special-purpose registers are used:
  - *Memory Address Register (MAR)*: Holds the address of the memory location to be accessed.
  - *Memory Data Register (MDR)*: Holds the data that is being written into memory, or will receive the data being read out from memory.
- Memory considered as a linear array of storage locations (bytes or words) each with unique address.



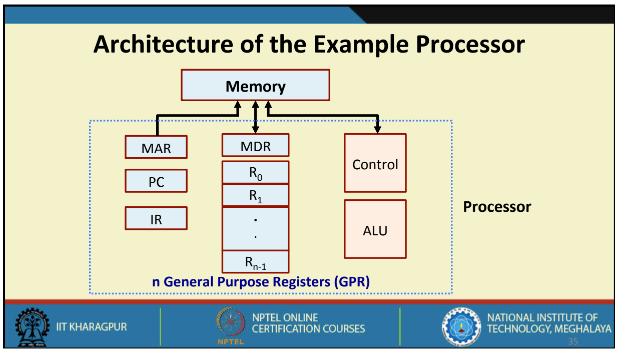


- To read data from memory
  - a) Load the memory address into MAR.
  - b) Issue the control signal *READ*.
  - c) The data read from the memory is stored into MDR.
- To write data into memory
  - a) Load the memory address into MAR.
  - b) Load the data to be written into MDR.
  - c) Issue the control signal *WRITE*.

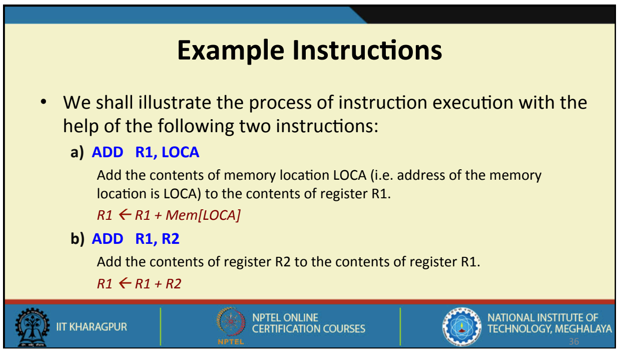


For Keeping Track of Program / Instructions

- Two special-purpose registers are used:
  - *Program Counter (PC)*: Holds the memory address of the next instruction to be executed.
    - Automatically incremented to point to the next instruction when an instruction is being executed.
  - *Instruction Register (IR)*: Temporarily holds an instruction that has been fetched from memory.
    - Need to be decoded to find out the instruction type.
    - Also contains information about the location of the data.



Architecture of the Example Processor



Example Instructions

- We shall illustrate the process of instruction execution with the help of the following two instructions:

a) ADD R1, LOCA

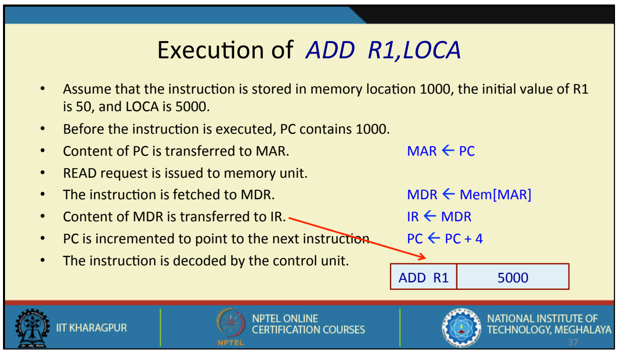
Add the contents of memory location LOCA (i.e. address of the memory location is LOCA) to the contents of register R1.

R1 ← R1 + Mem[LOCA]

b) ADD R1, R2

Add the contents of register R2 to the contents of register R1.

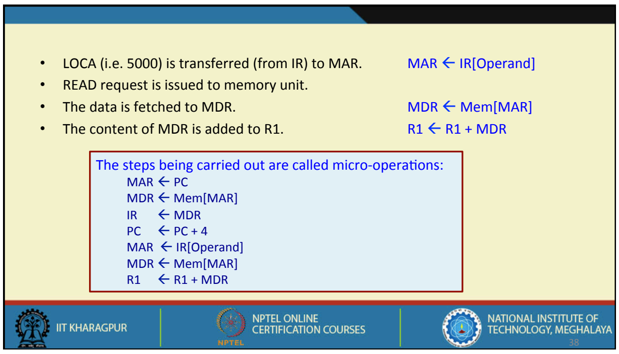
R1 ← R1 + R2



Execution of *ADD R1,LOCA*

- Assume that the instruction is stored in memory location 1000, the initial value of R1 is 50, and LOCA is 5000.
- Before the instruction is executed, PC contains 1000.
- Content of PC is transferred to MAR. MAR ← PC
- READ request is issued to memory unit.
- The instruction is fetched to MDR. MDR ← Mem[MAR]
- Content of MDR is transferred to IR. IR ← MDR
- PC is incremented to point to the next instruction. PC ← PC + 4
- The instruction is decoded by the control unit.

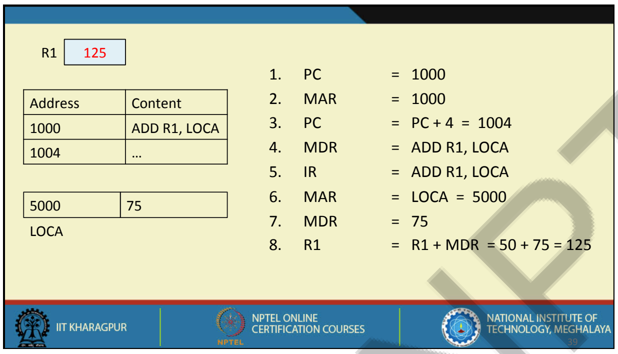
ADD R1	5000



- LOCA (i.e. 5000) is transferred (from IR) to MAR. MAR ← IR[Operand]
- READ request is issued to memory unit.
- The data is fetched to MDR. MDR ← Mem[MAR]
- The content of MDR is added to R1. R1 ← R1 + MDR

The steps being carried out are called micro-operations:

MAR ← PC
MDR ← Mem[MAR]
IR ← MDR
PC ← PC + 4
MAR ← IR[Operand]
MDR ← Mem[MAR]
R1 ← R1 + MDR



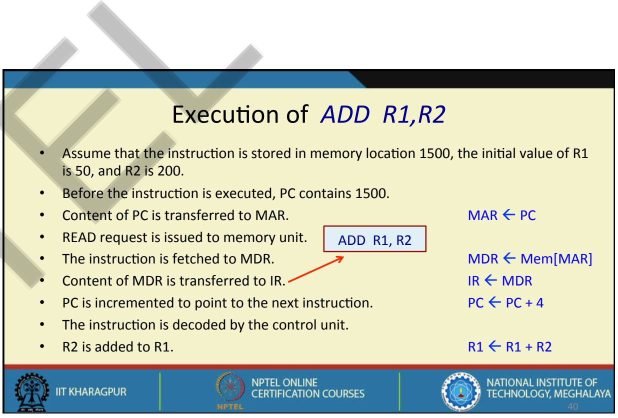
R1 125

Address	Content
1000	ADD R1, LOCA
1004	...

5000	75

LOCA

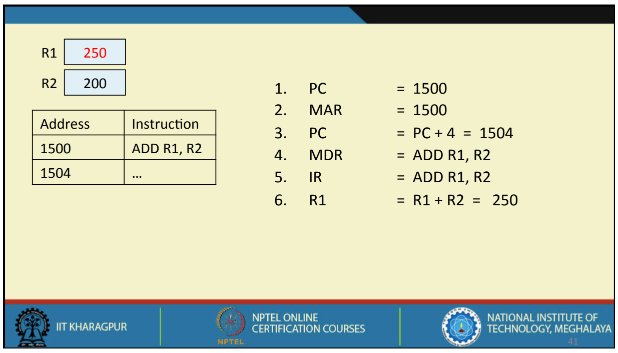
1. PC = 1000
2. MAR = 1000
3. PC = PC + 4 = 1004
4. MDR = ADD R1, LOCA
5. IR = ADD R1, LOCA
6. MAR = LOCA = 5000
7. MDR = 75
8. R1 = R1 + MDR = 50 + 75 = 125



Execution of *ADD R1,R2*

- Assume that the instruction is stored in memory location 1500, the initial value of R1 is 50, and R2 is 200.
- Before the instruction is executed, PC contains 1500.
- Content of PC is transferred to MAR. MAR ← PC
- READ request is issued to memory unit.
- The instruction is fetched to MDR. MDR ← Mem[MAR]
- Content of MDR is transferred to IR. IR ← MDR
- PC is incremented to point to the next instruction. PC ← PC + 4
- The instruction is decoded by the control unit.
- R2 is added to R1. R1 ← R1 + R2

ADD R1, R2

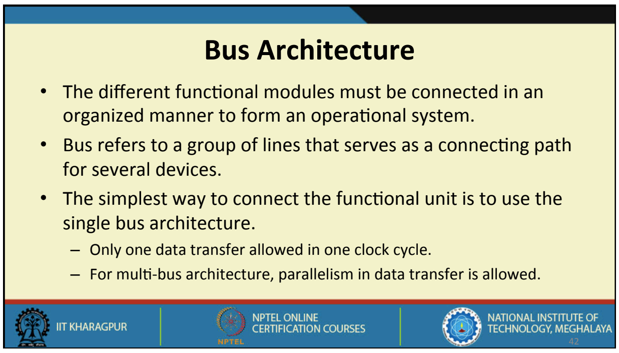


R1 250

R2 200

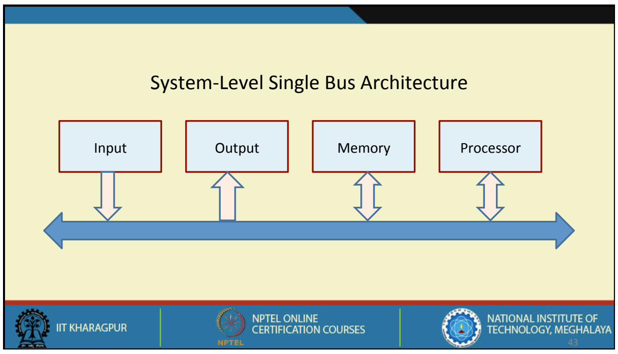
Address	Instruction
1500	ADD R1, R2
1504	...

1. PC = 1500
2. MAR = 1500
3. PC = PC + 4 = 1504
4. MDR = ADD R1, R2
5. IR = ADD R1, R2
6. R1 = R1 + R2 = 250

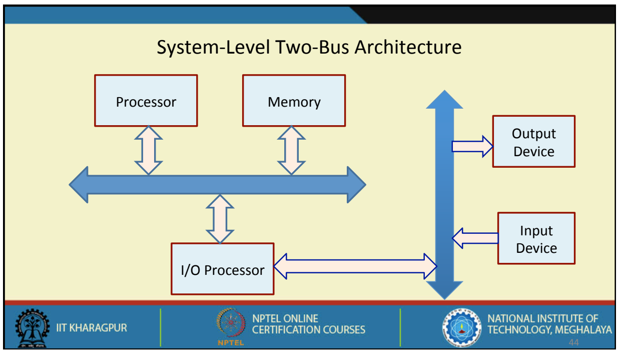


Bus Architecture

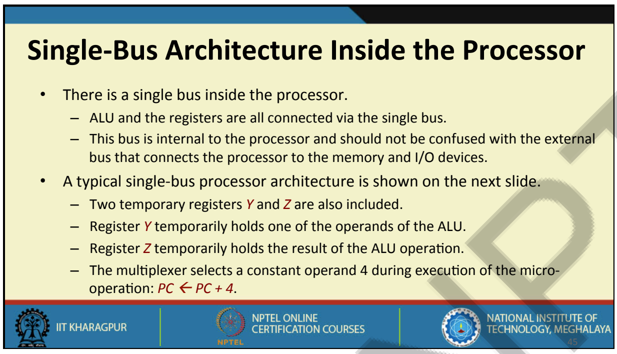
- The different functional modules must be connected in an organized manner to form an operational system.
- Bus refers to a group of lines that serves as a connecting path for several devices.
- The simplest way to connect the functional unit is to use the single bus architecture.
  - Only one data transfer allowed in one clock cycle.
  - For multi-bus architecture, parallelism in data transfer is allowed.



System-Level Single Bus Architecture

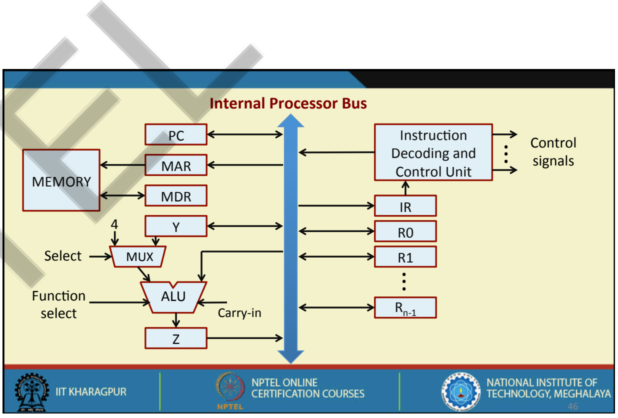


System-Level Two-Bus Architecture

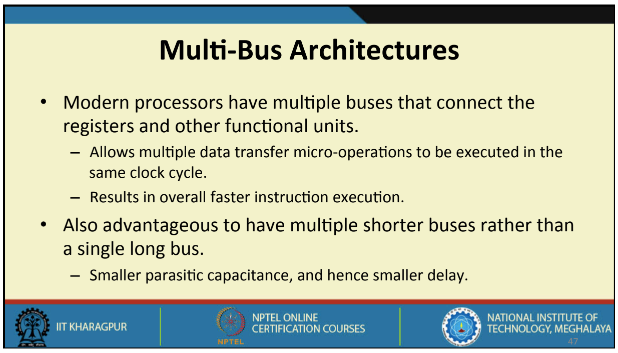


Single-Bus Architecture Inside the Processor

- There is a single bus inside the processor.
  - ALU and the registers are all connected via the single bus.
  - This bus is internal to the processor and should not be confused with the external bus that connects the processor to the memory and I/O devices.
- A typical single-bus processor architecture is shown on the next slide.
  - Two temporary registers *Y* and *Z* are also included.
  - Register *Y* temporarily holds one of the operands of the ALU.
  - Register *Z* temporarily holds the result of the ALU operation.
  - The multiplexer selects a constant operand 4 during execution of the micro-operation: *PC ← PC + 4*.

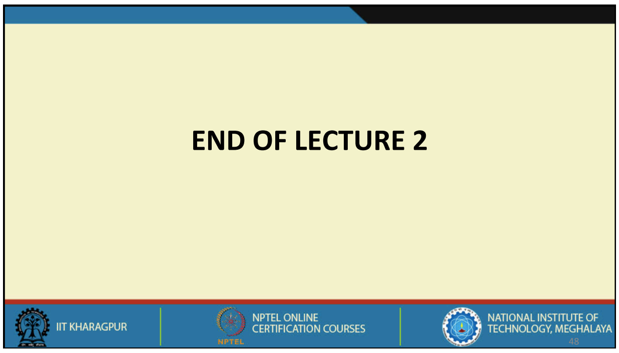


Internal Processor Bus

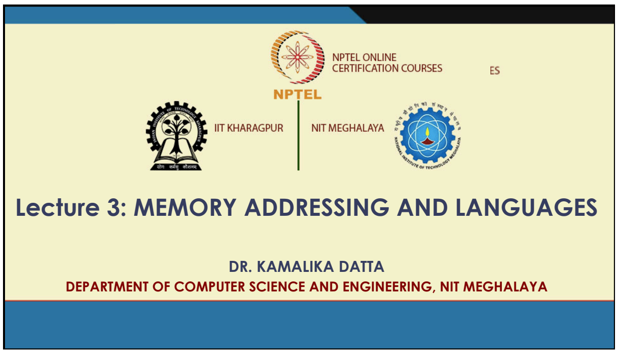


Multi-Bus Architectures

- Modern processors have multiple buses that connect the registers and other functional units.
  - Allows multiple data transfer micro-operations to be executed in the same clock cycle.
  - Results in overall faster instruction execution.
- Also advantageous to have multiple shorter buses rather than a single long bus.
  - Smaller parasitic capacitance, and hence smaller delay.



END OF LECTURE 2



NPTEL ONLINE CERTIFICATION COURSES

NPTEL

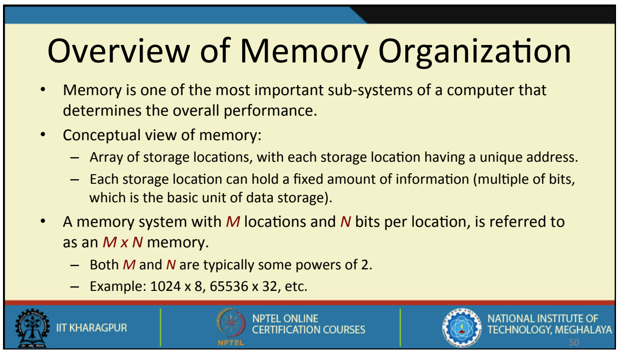
ES

IIT KHARAGPUR | NIT MEGHALAYA

Lecture 3: MEMORY ADDRESSING AND LANGUAGES

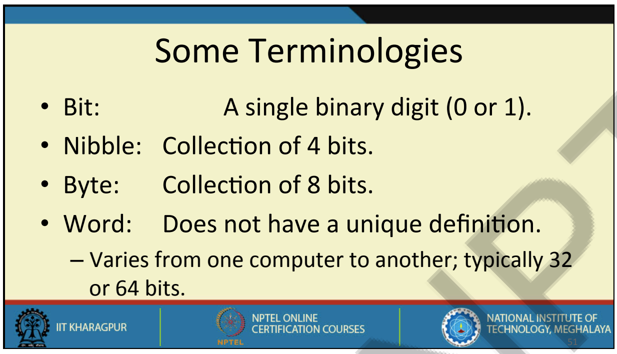
DR. KAMALIKA DATTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, NIT MEGHALAYA



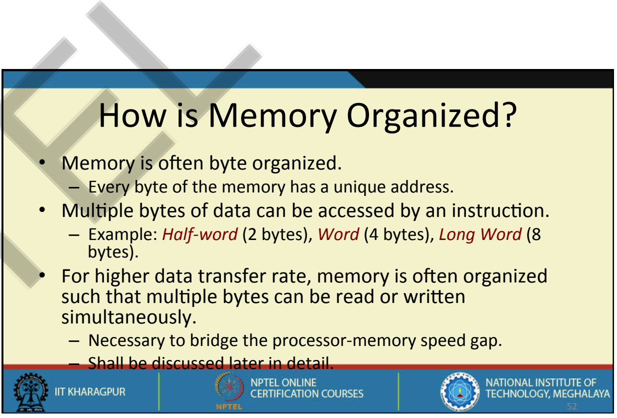
Overview of Memory Organization

- Memory is one of the most important sub-systems of a computer that determines the overall performance.
- Conceptual view of memory:
  - Array of storage locations, with each storage location having a unique address.
  - Each storage location can hold a fixed amount of information (multiple of bits, which is the basic unit of data storage).
- A memory system with *M* locations and *N* bits per location, is referred to as an *M x N* memory.
  - Both *M* and *N* are typically some powers of 2.
  - Example: 1024 x 8, 65536 x 32, etc.



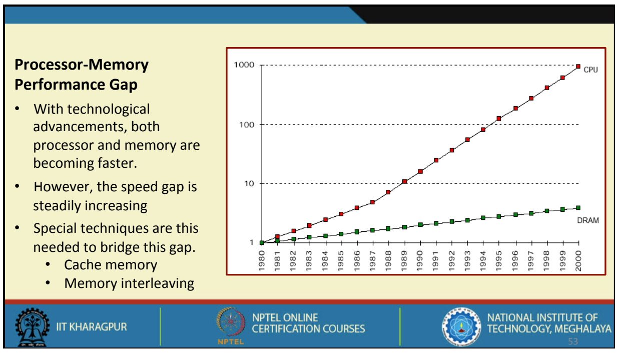
Some Terminologies

- Bit: A single binary digit (0 or 1).
- Nibble: Collection of 4 bits.
- Byte: Collection of 8 bits.
- Word: Does not have a unique definition.
  - Varies from one computer to another; typically 32 or 64 bits.



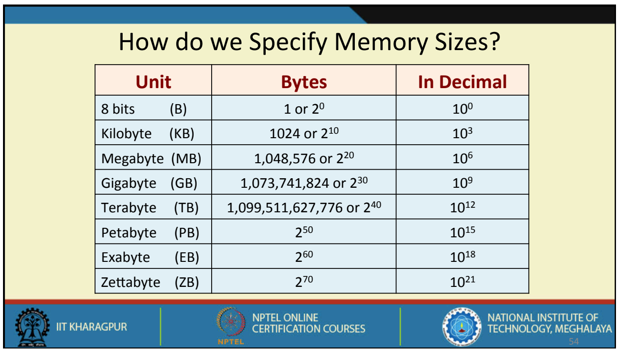
How is Memory Organized?

- Memory is often byte organized.
  - Every byte of the memory has a unique address.
- Multiple bytes of data can be accessed by an instruction.
  - Example: *Half-word* (2 bytes), *Word* (4 bytes), *Long Word* (8 bytes).
- For higher data transfer rate, memory is often organized such that multiple bytes can be read or written simultaneously.
  - Necessary to bridge the processor-memory speed gap.
  - Shall be discussed later in detail.



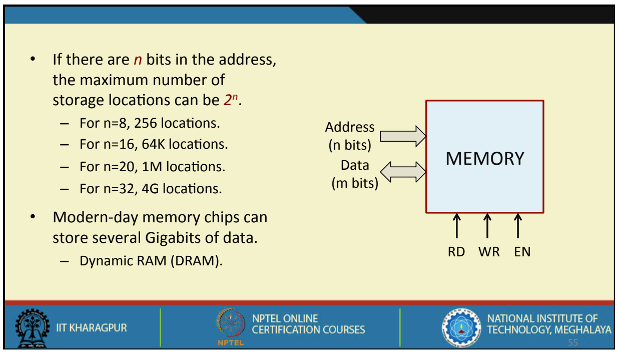
Processor-Memory Performance Gap

- With technological advancements, both processor and memory are becoming faster.
- However, the speed gap is steadily increasing
- Special techniques are this needed to bridge this gap.
  - Cache memory
  - Memory interleaving

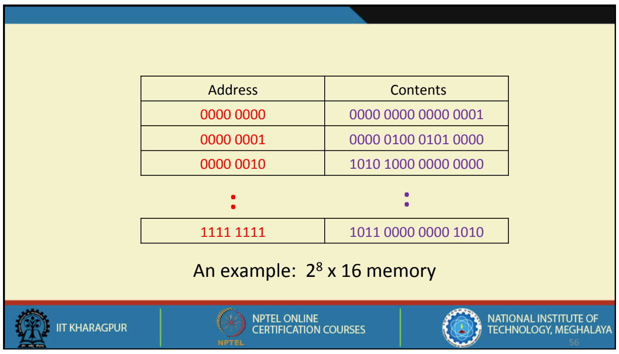


How do we Specify Memory Sizes?

Unit	Bytes	In Decimal
8 bits (B)	1 or 2^0	10^0
Kilobyte (KB)	1024 or 2^{10}	10^3
Megabyte (MB)	1,048,576 or 2^{20}	10^6
Gigabyte (GB)	1,073,741,824 or 2^{30}	10^9
Terabyte (TB)	1,099,511,627,776 or 2^{40}	10^{12}
Petabyte (PB)	2^{50}	10^{15}
Exabyte (EB)	2^{60}	10^{18}
Zettabyte (ZB)	2^{70}	10^{21}

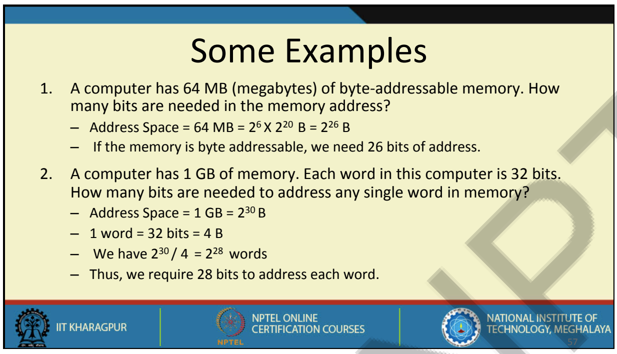


- If there are *n* bits in the address, the maximum number of storage locations can be 2^n.
  - For n=8, 256 locations.
  - For n=16, 64K locations.
  - For n=20, 1M locations.
  - For n=32, 4G locations.
- Modern-day memory chips can store several Gigabits of data.
  - Dynamic RAM (DRAM).



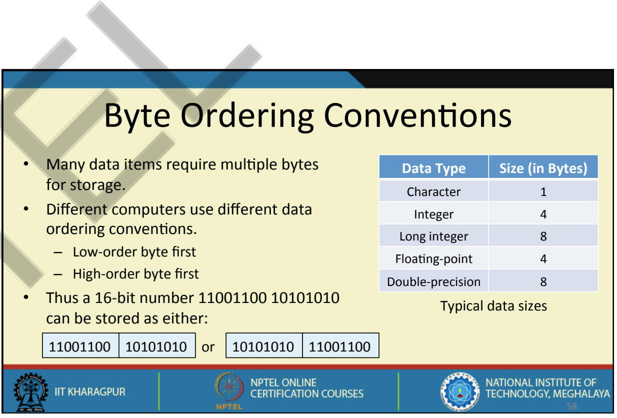
Address	Contents
0000 0000	0000 0000 0000 0001
0000 0001	0000 0100 0101 0000
0000 0010	1010 1000 0000 0000
:	:
1111 1111	1011 0000 0000 1010

An example: 2^8 x 16 memory



Some Examples

1. A computer has 64 MB (megabytes) of byte-addressable memory. How many bits are needed in the memory address?
   - Address Space = 64 MB = 2^6 X 2^{20} B = 2^{26} B
   - If the memory is byte addressable, we need 26 bits of address.
2. A computer has 1 GB of memory. Each word in this computer is 32 bits. How many bits are needed to address any single word in memory?
   - Address Space = 1 GB = 2^{30} B
   - 1 word = 32 bits = 4 B
   - We have 2^{30} / 4 = 2^{28} words
   - Thus, we require 28 bits to address each word.



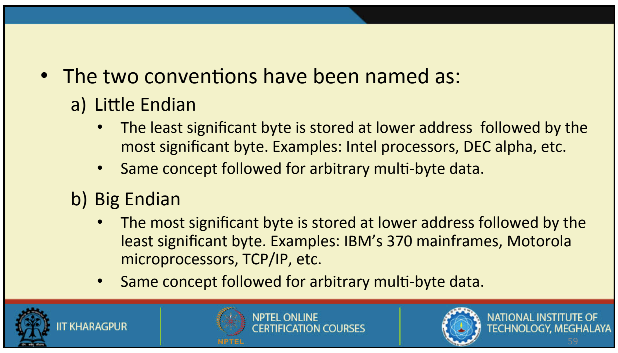
Byte Ordering Conventions

- Many data items require multiple bytes for storage.
- Different computers use different data ordering conventions.
  - Low-order byte first
  - High-order byte first
- Thus a 16-bit number 11001100 10101010 can be stored as either:

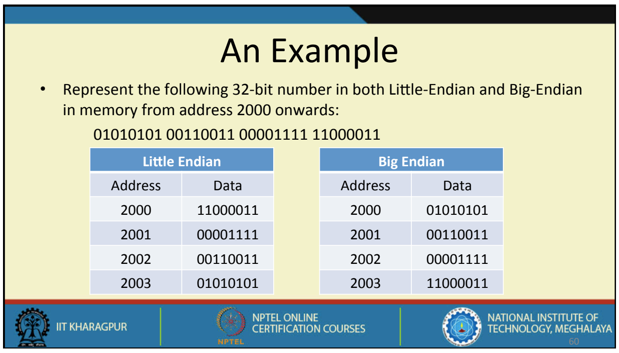
11001100	10101010	or	10101010	11001100

Data Type	Size (in Bytes)
Character	1
Integer	4
Long integer	8
Floating-point	4
Double-precision	8

Typical data sizes



- The two conventions have been named as:
  - a) Little Endian
    - The least significant byte is stored at lower address followed by the most significant byte. Examples: Intel processors, DEC alpha, etc.
    - Same concept followed for arbitrary multi-byte data.
  - b) Big Endian
    - The most significant byte is stored at lower address followed by the least significant byte. Examples: IBM's 370 mainframes, Motorola microprocessors, TCP/IP, etc.
    - Same concept followed for arbitrary multi-byte data.

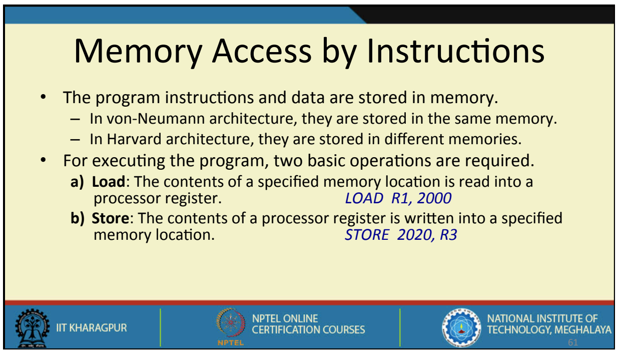


An Example

- Represent the following 32-bit number in both Little-Endian and Big-Endian in memory from address 2000 onwards:

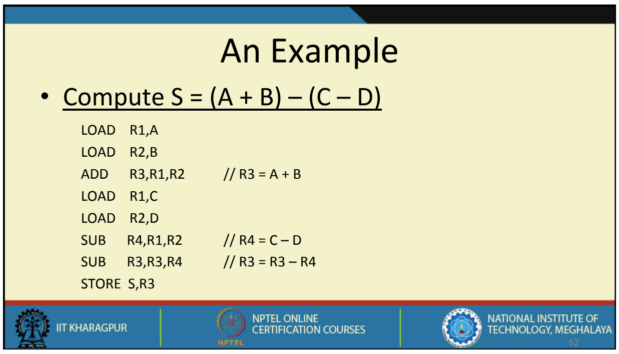
01010101 00110011 00001111 11000011

Little Endian		Big Endian	
Address	Data	Address	Data
2000	11000011	2000	01010101
2001	00001111	2001	00110011
2002	00110011	2002	00001111
2003	01010101	2003	11000011



Memory Access by Instructions

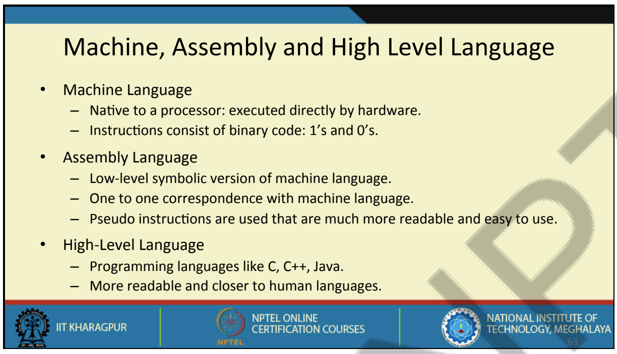
- The program instructions and data are stored in memory.
  - In von-Neumann architecture, they are stored in the same memory.
  - In Harvard architecture, they are stored in different memories.
- For executing the program, two basic operations are required.
  - **a) Load**: The contents of a specified memory location is read into a processor register. *LOAD R1, 2000*
  - **b) Store**: The contents of a processor register is written into a specified memory location. *STORE 2020, R3*



An Example

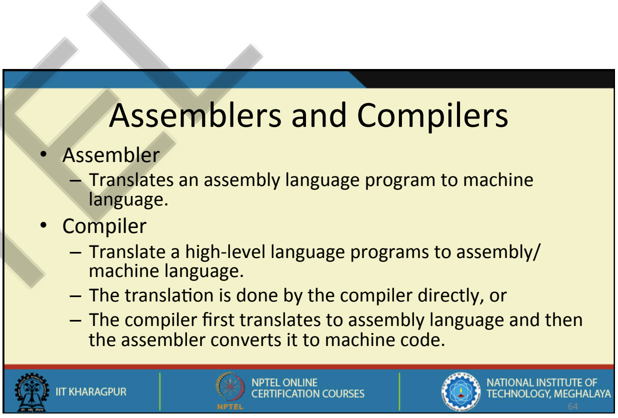
- Compute S = (A + B) – (C – D)

```
LOAD   R1,A
LOAD   R2,B
ADD    R3,R1,R2      // R3 = A + B
LOAD   R1,C
LOAD   R2,D
SUB    R4,R1,R2      // R4 = C – D
SUB    R3,R3,R4      // R3 = R3 – R4
STORE  S,R3
```



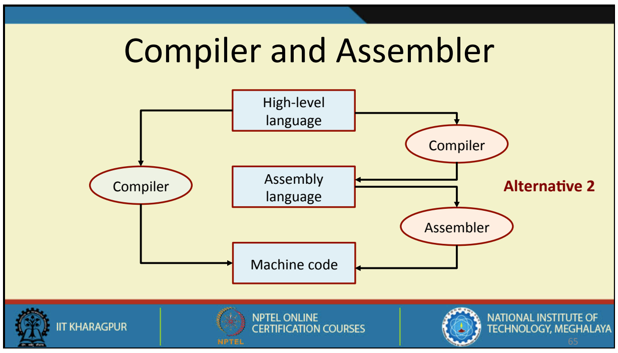
Machine, Assembly and High Level Language

- Machine Language
  - Native to a processor: executed directly by hardware.
  - Instructions consist of binary code: 1's and 0's.
- Assembly Language
  - Low-level symbolic version of machine language.
  - One to one correspondence with machine language.
  - Pseudo instructions are used that are much more readable and easy to use.
- High-Level Language
  - Programming languages like C, C++, Java.
  - More readable and closer to human languages.

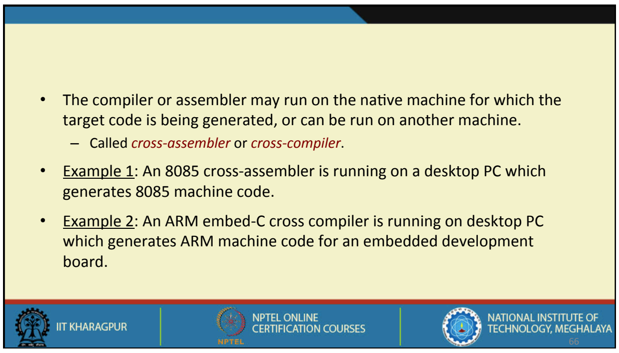


Assemblers and Compilers

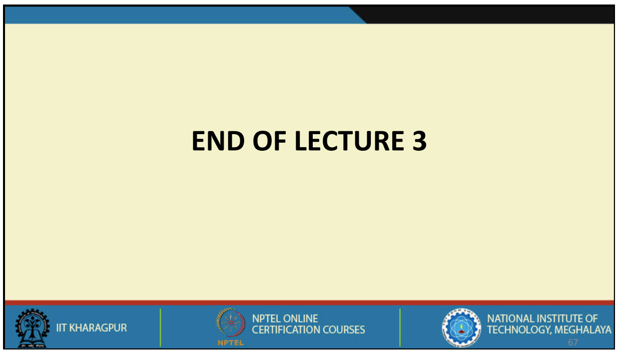
- Assembler
  - Translates an assembly language program to machine language.
- Compiler
  - Translate a high-level language programs to assembly/ machine language.
  - The translation is done by the compiler directly, or
  - The compiler first translates to assembly language and then the assembler converts it to machine code.



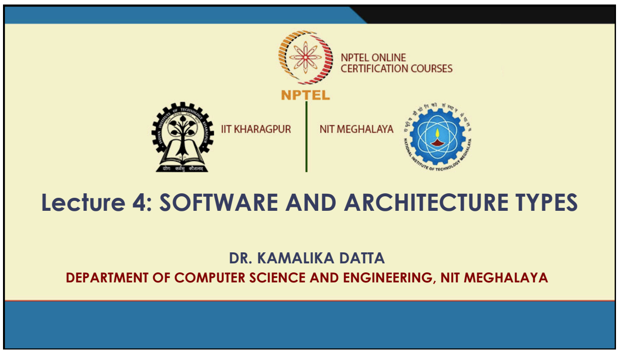
Compiler and Assembler



- The compiler or assembler may run on the native machine for which the target code is being generated, or can be run on another machine.
  - Called *cross-assembler* or *cross-compiler*.
- Example 1: An 8085 cross-assembler is running on a desktop PC which generates 8085 machine code.
- Example 2: An ARM embed-C cross compiler is running on desktop PC which generates ARM machine code for an embedded development board.



END OF LECTURE 3



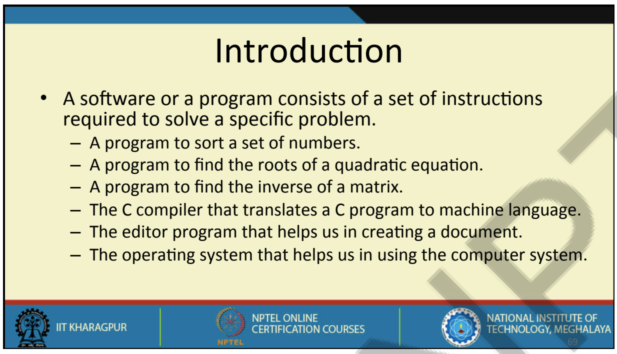
NPTEL ONLINE CERTIFICATION COURSES

NPTEL

IIT KHARAGPUR | NIT MEGHALAYA

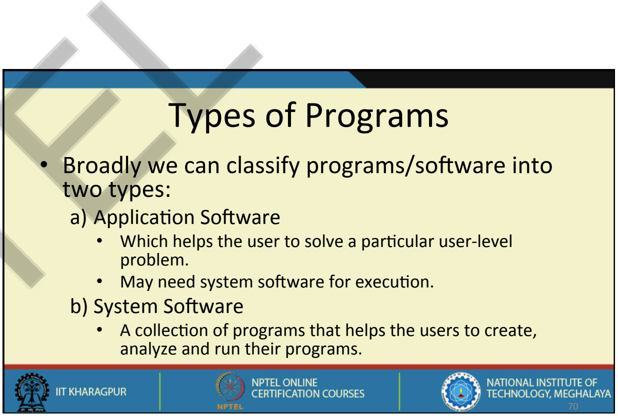
Lecture 4: SOFTWARE AND ARCHITECTURE TYPES

DR. KAMALIKA DATTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, NIT MEGHALAYA



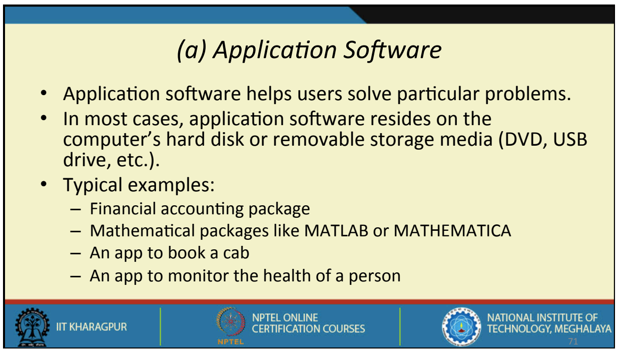
Introduction

- A software or a program consists of a set of instructions required to solve a specific problem.
  – A program to sort a set of numbers.
  – A program to find the roots of a quadratic equation.
  – A program to find the inverse of a matrix.
  – The C compiler that translates a C program to machine language.
  – The editor program that helps us in creating a document.
  – The operating system that helps us in using the computer system.



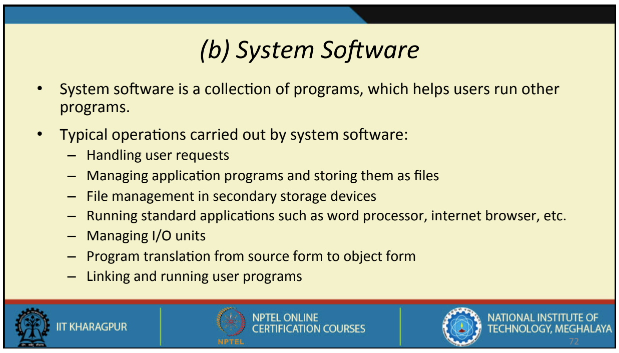
Types of Programs

- Broadly we can classify programs/software into two types:
  a) Application Software
    - Which helps the user to solve a particular user-level problem.
    - May need system software for execution.
  b) System Software
    - A collection of programs that helps the users to create, analyze and run their programs.



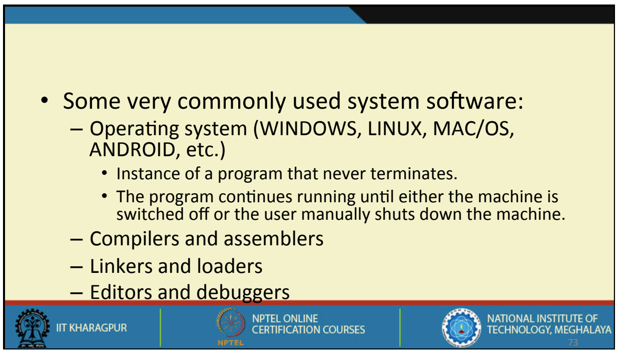
(a) Application Software

- Application software helps users solve particular problems.
- In most cases, application software resides on the computer's hard disk or removable storage media (DVD, USB drive, etc.).
- Typical examples:
  – Financial accounting package
  – Mathematical packages like MATLAB or MATHEMATICA
  – An app to book a cab
  – An app to monitor the health of a person

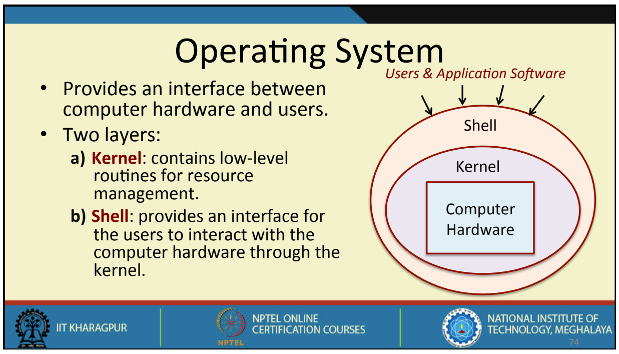


(b) System Software

- System software is a collection of programs, which helps users run other programs.
- Typical operations carried out by system software:
  – Handling user requests
  – Managing application programs and storing them as files
  – File management in secondary storage devices
  – Running standard applications such as word processor, internet browser, etc.
  – Managing I/O units
  – Program translation from source form to object form
  – Linking and running user programs

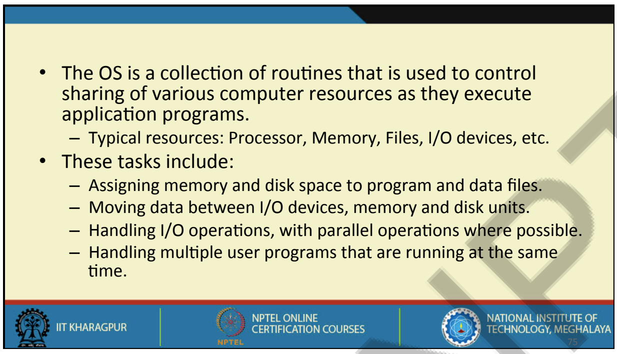


- Some very commonly used system software:
  - Operating system (WINDOWS, LINUX, MAC/OS, ANDROID, etc.)
    - Instance of a program that never terminates.
    - The program continues running until either the machine is switched off or the user manually shuts down the machine.
  - Compilers and assemblers
  - Linkers and loaders
  - Editors and debuggers

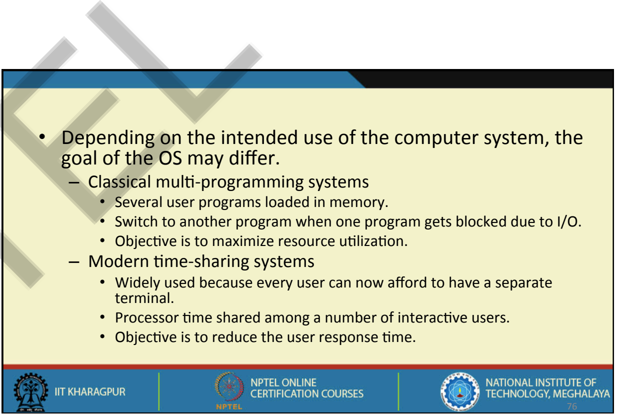


Operating System

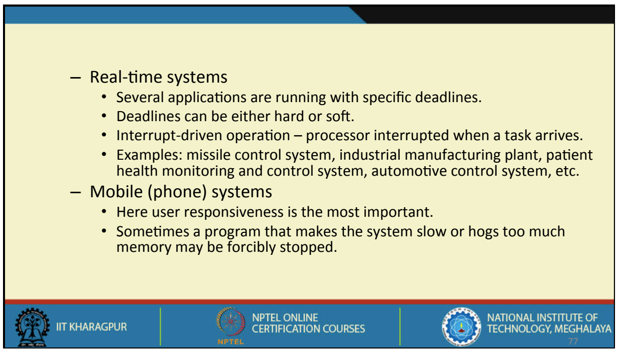
- Provides an interface between computer hardware and users.
- Two layers:
  - **a) Kernel**: contains low-level routines for resource management.
  - **b) Shell**: provides an interface for the users to interact with the computer hardware through the kernel.



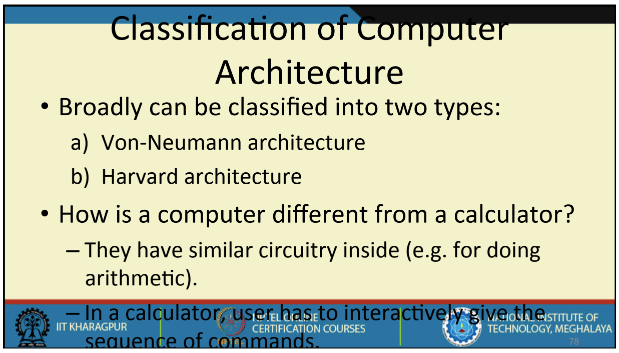
- The OS is a collection of routines that is used to control sharing of various computer resources as they execute application programs.
  - Typical resources: Processor, Memory, Files, I/O devices, etc.
- These tasks include:
  - Assigning memory and disk space to program and data files.
  - Moving data between I/O devices, memory and disk units.
  - Handling I/O operations, with parallel operations where possible.
  - Handling multiple user programs that are running at the same time.



- Depending on the intended use of the computer system, the goal of the OS may differ.
  - Classical multi-programming systems
    - Several user programs loaded in memory.
    - Switch to another program when one program gets blocked due to I/O.
    - Objective is to maximize resource utilization.
  - Modern time-sharing systems
    - Widely used because every user can now afford to have a separate terminal.
    - Processor time shared among a number of interactive users.
    - Objective is to reduce the user response time.

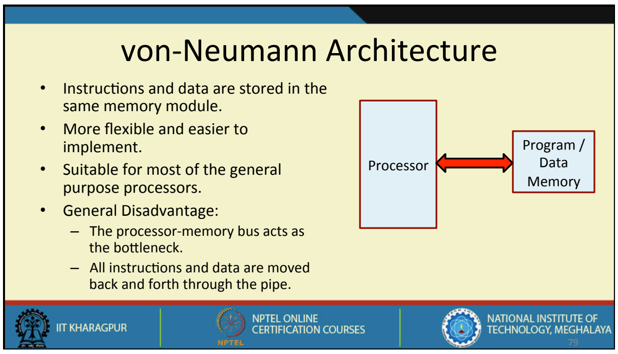


- Real-time systems
  - Several applications are running with specific deadlines.
  - Deadlines can be either hard or soft.
  - Interrupt-driven operation – processor interrupted when a task arrives.
  - Examples: missile control system, industrial manufacturing plant, patient health monitoring and control system, automotive control system, etc.
- Mobile (phone) systems
  - Here user responsiveness is the most important.
  - Sometimes a program that makes the system slow or hogs too much memory may be forcibly stopped.



Classification of Computer Architecture

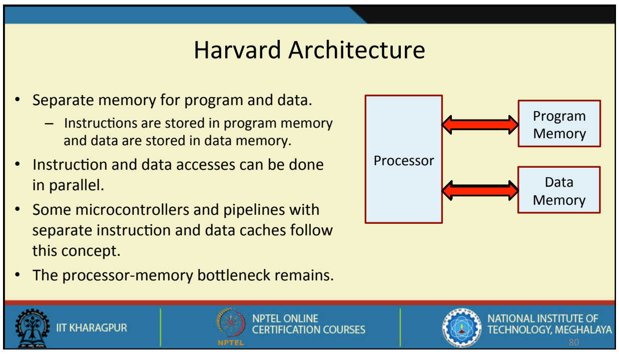
- Broadly can be classified into two types:
  - a) Von-Neumann architecture
  - b) Harvard architecture
- How is a computer different from a calculator?
  - They have similar circuitry inside (e.g. for doing arithmetic).
  - In a calculator, user has to interactively give the sequence of commands.



von-Neumann Architecture

- Instructions and data are stored in the same memory module.
- More flexible and easier to implement.
- Suitable for most of the general purpose processors.
- General Disadvantage:
  - The processor-memory bus acts as the bottleneck.
  - All instructions and data are moved back and forth through the pipe.

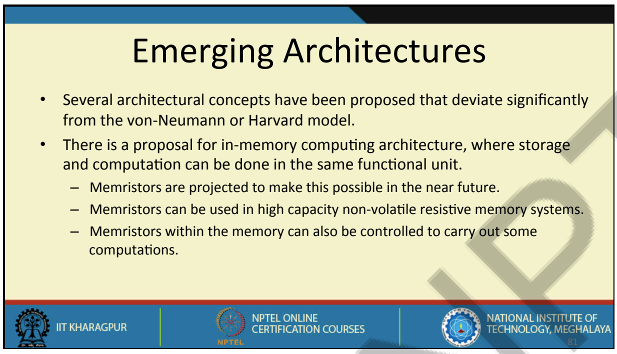
Processor
Program / Data Memory



Harvard Architecture

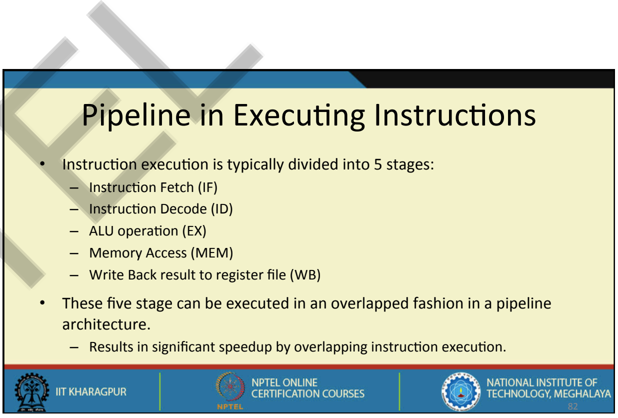
- Separate memory for program and data.
  - Instructions are stored in program memory and data are stored in data memory.
- Instruction and data accesses can be done in parallel.
- Some microcontrollers and pipelines with separate instruction and data caches follow this concept.
- The processor-memory bottleneck remains.

Processor
Program Memory
Data Memory



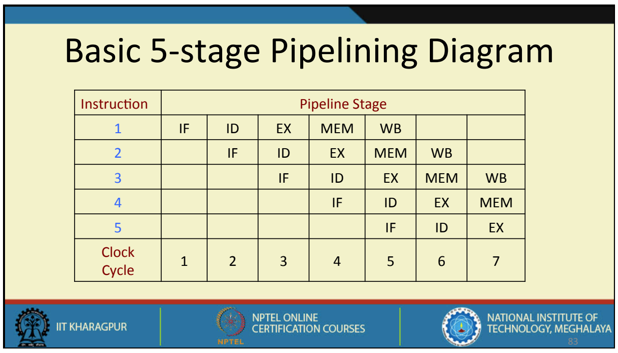
Emerging Architectures

- Several architectural concepts have been proposed that deviate significantly from the von-Neumann or Harvard model.
- There is a proposal for in-memory computing architecture, where storage and computation can be done in the same functional unit.
  - Memristors are projected to make this possible in the near future.
  - Memristors can be used in high capacity non-volatile resistive memory systems.
  - Memristors within the memory can also be controlled to carry out some computations.



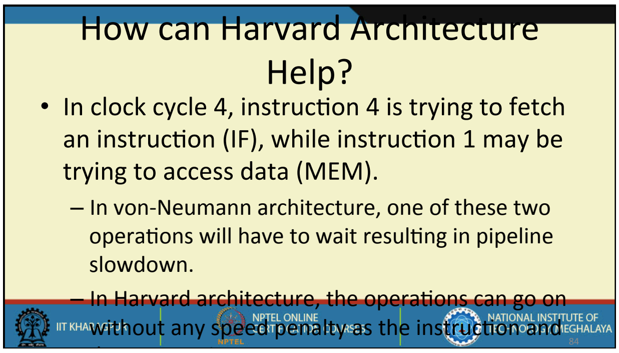
Pipeline in Executing Instructions

- Instruction execution is typically divided into 5 stages:
  - Instruction Fetch (IF)
  - Instruction Decode (ID)
  - ALU operation (EX)
  - Memory Access (MEM)
  - Write Back result to register file (WB)
- These five stage can be executed in an overlapped fashion in a pipeline architecture.
  - Results in significant speedup by overlapping instruction execution.



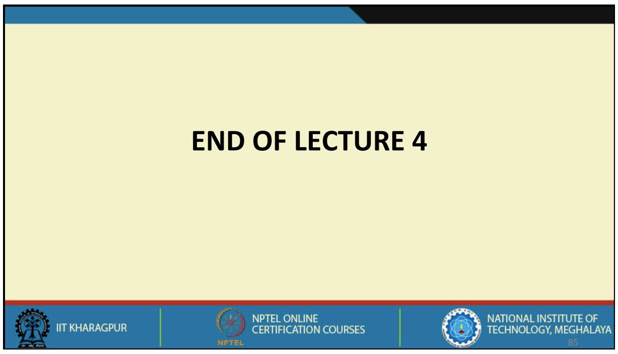
Basic 5-stage Pipelining Diagram

Instruction	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

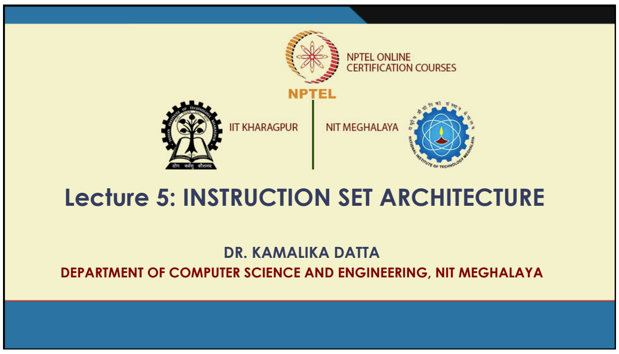


How can Harvard Architecture Help?

- In clock cycle 4, instruction 4 is trying to fetch an instruction (IF), while instruction 1 may be trying to access data (MEM).
  - In von-Neumann architecture, one of these two operations will have to wait resulting in pipeline slowdown.
  - In Harvard architecture, the operations can go on without any speed penalty as the instruction and



END OF LECTURE 4

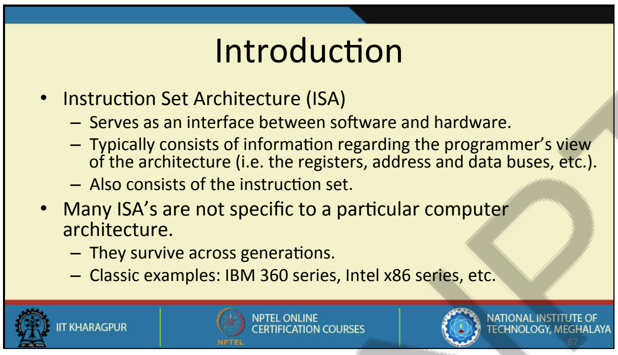


NPTEL ONLINE CERTIFICATION COURSES

IIT KHARAGPUR | NIT MEGHALAYA

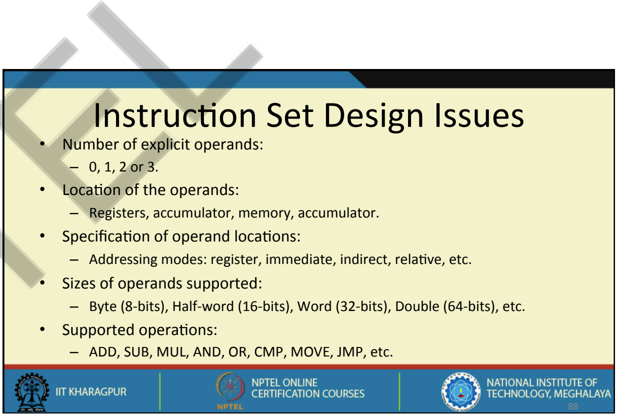
Lecture 5: INSTRUCTION SET ARCHITECTURE

DR. KAMALIKA DATTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, NIT MEGHALAYA



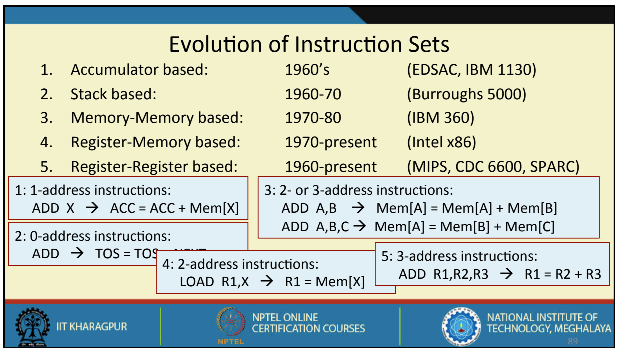
Introduction

- Instruction Set Architecture (ISA)
  - Serves as an interface between software and hardware.
  - Typically consists of information regarding the programmer's view of the architecture (i.e. the registers, address and data buses, etc.).
  - Also consists of the instruction set.
- Many ISA's are not specific to a particular computer architecture.
  - They survive across generations.
  - Classic examples: IBM 360 series, Intel x86 series, etc.



Instruction Set Design Issues

- Number of explicit operands:
  - 0, 1, 2 or 3.
- Location of the operands:
  - Registers, accumulator, memory, accumulator.
- Specification of operand locations:
  - Addressing modes: register, immediate, indirect, relative, etc.
- Sizes of operands supported:
  - Byte (8-bits), Half-word (16-bits), Word (32-bits), Double (64-bits), etc.
- Supported operations:
  - ADD, SUB, MUL, AND, OR, CMP, MOVE, JMP, etc.



Evolution of Instruction Sets

1.	Accumulator based:	1960's	(EDSAC, IBM 1130)
2.	Stack based:	1960-70	(Burroughs 5000)
3.	Memory-Memory based:	1970-80	(IBM 360)
4.	Register-Memory based:	1970-present	(Intel x86)
5.	Register-Register based:	1960-present	(MIPS, CDC 6600, SPARC)

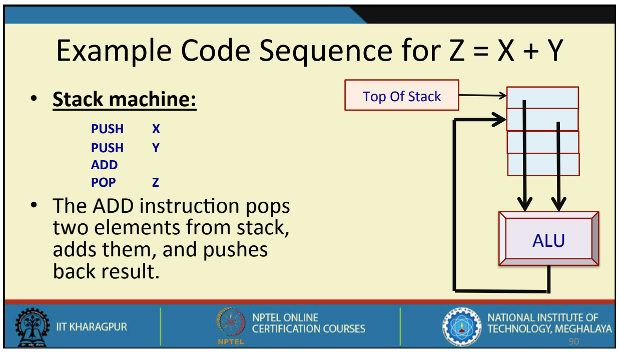
1: 1-address instructions:
ADD X → ACC = ACC + Mem[X]

2: 0-address instructions:
ADD → TOS = TOS

3: 2- or 3-address instructions:
ADD A,B → Mem[A] = Mem[A] + Mem[B]
ADD A,B,C → Mem[A] = Mem[B] + Mem[C]

4: 2-address instructions:
LOAD R1,X → R1 = Mem[X]

5: 3-address instructions:
ADD R1,R2,R3 → R1 = R2 + R3

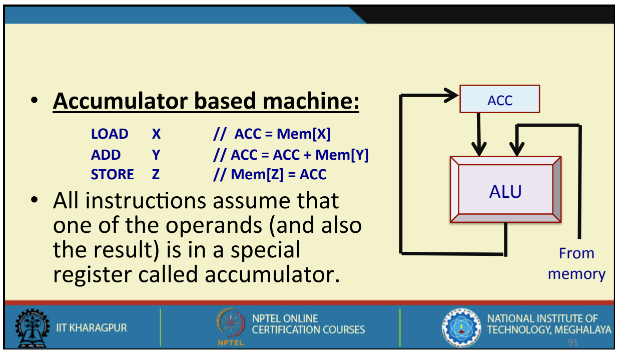


Example Code Sequence for Z = X + Y

- **Stack machine:**

```
PUSH   X
PUSH   Y
ADD
POP    Z
```

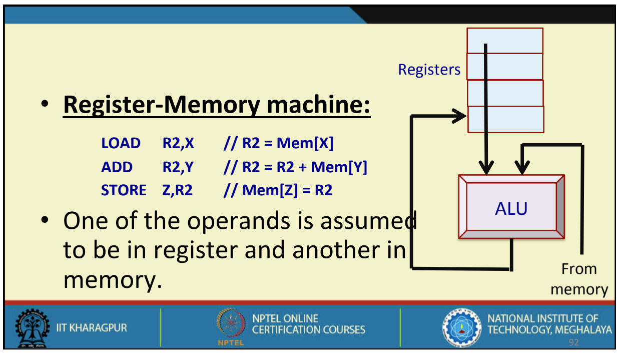
- The ADD instruction pops two elements from stack, adds them, and pushes back result.



- **Accumulator based machine:**

LOAD	X	// ACC = Mem[X]
ADD	Y	// ACC = ACC + Mem[Y]
STORE	Z	// Mem[Z] = ACC

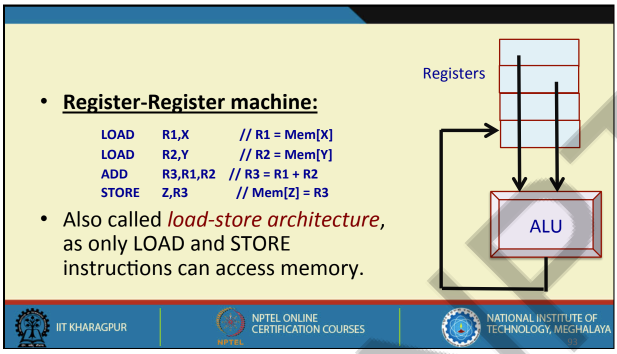
- All instructions assume that one of the operands (and also the result) is in a special register called accumulator.



- **Register-Memory machine:**

LOAD	R2,X	// R2 = Mem[X]
ADD	R2,Y	// R2 = R2 + Mem[Y]
STORE	Z,R2	// Mem[Z] = R2

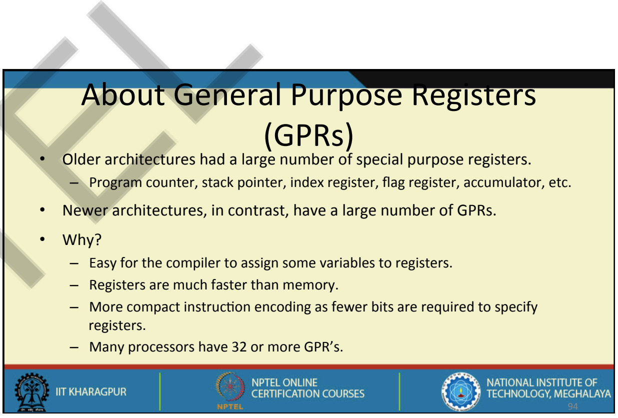
- One of the operands is assumed to be in register and another in memory.



- **Register-Register machine:**

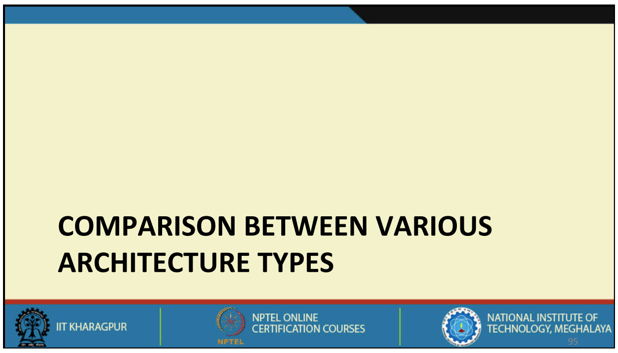
LOAD	R1,X	// R1 = Mem[X]
LOAD	R2,Y	// R2 = Mem[Y]
ADD	R3,R1,R2	// R3 = R1 + R2
STORE	Z,R3	// Mem[Z] = R3

- Also called *load-store architecture*, as only LOAD and STORE instructions can access memory.

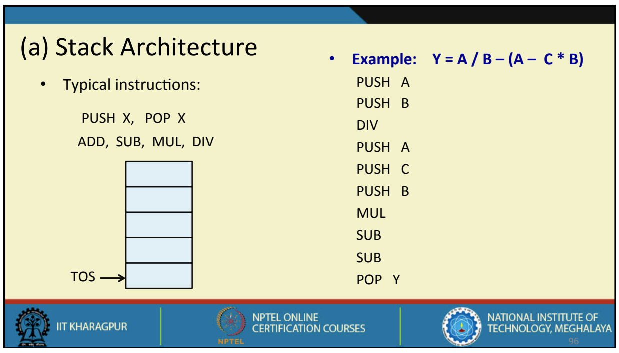


About General Purpose Registers (GPRs)

- Older architectures had a large number of special purpose registers.
  - Program counter, stack pointer, index register, flag register, accumulator, etc.
- Newer architectures, in contrast, have a large number of GPRs.
- Why?
  - Easy for the compiler to assign some variables to registers.
  - Registers are much faster than memory.
  - More compact instruction encoding as fewer bits are required to specify registers.
  - Many processors have 32 or more GPR's.



COMPARISON BETWEEN VARIOUS ARCHITECTURE TYPES



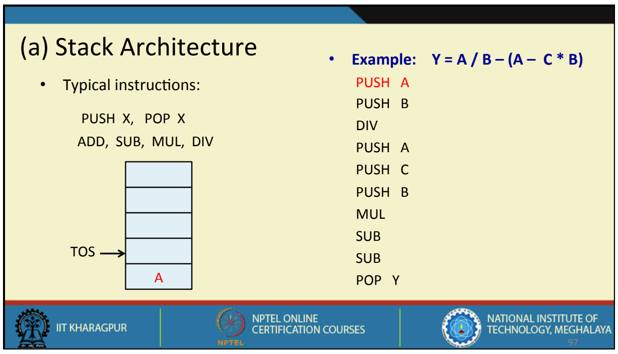
(a) Stack Architecture

- Typical instructions:

PUSH X, POP X
ADD, SUB, MUL, DIV

- **Example: Y = A / B – (A – C * B)**

```
PUSH  A
PUSH  B
DIV
PUSH  A
PUSH  C
PUSH  B
MUL
SUB
SUB
POP   Y
```



(a) Stack Architecture

- Typical instructions:

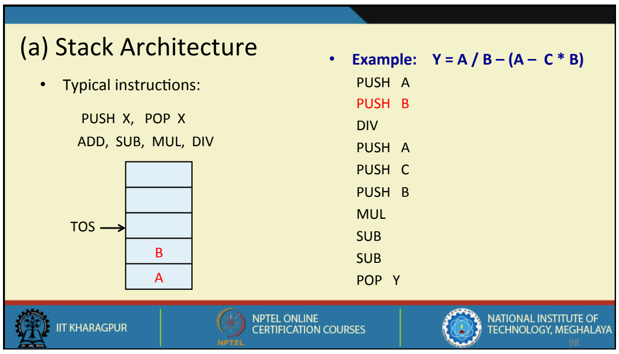
PUSH X, POP X
ADD, SUB, MUL, DIV

TOS →
A

- **Example: Y = A / B – (A – C * B)**

```
PUSH  A
PUSH  B
DIV
PUSH  A
PUSH  C
PUSH  B
MUL
SUB
SUB
POP  Y
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA 97



(a) Stack Architecture

- Typical instructions:

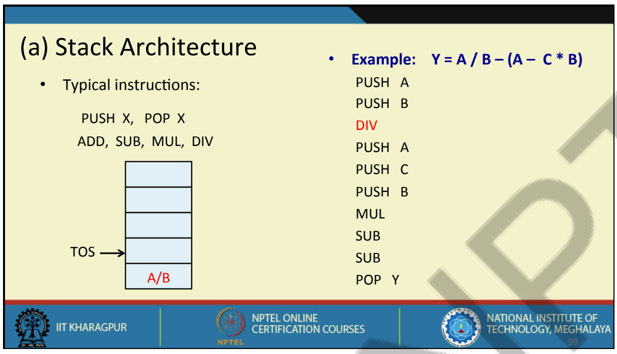
PUSH X, POP X
ADD, SUB, MUL, DIV

TOS →
B
A

- **Example: Y = A / B – (A – C * B)**

```
PUSH  A
PUSH  B
DIV
PUSH  A
PUSH  C
PUSH  B
MUL
SUB
SUB
POP  Y
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA 98



(a) Stack Architecture

- Typical instructions:

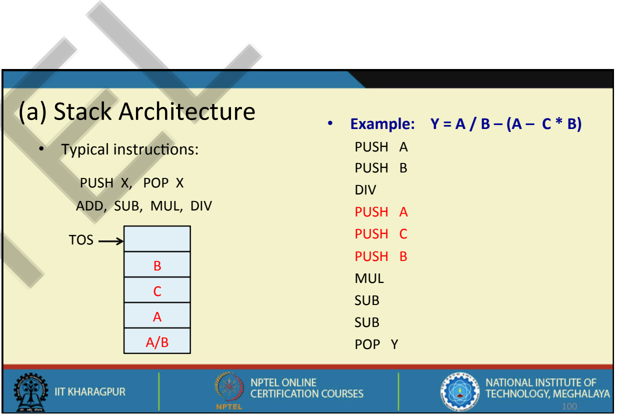
PUSH X, POP X
ADD, SUB, MUL, DIV

TOS →
A/B

- **Example: Y = A / B – (A – C * B)**

```
PUSH  A
PUSH  B
DIV
PUSH  A
PUSH  C
PUSH  B
MUL
SUB
SUB
POP  Y
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA 99



(a) Stack Architecture

- Typical instructions:

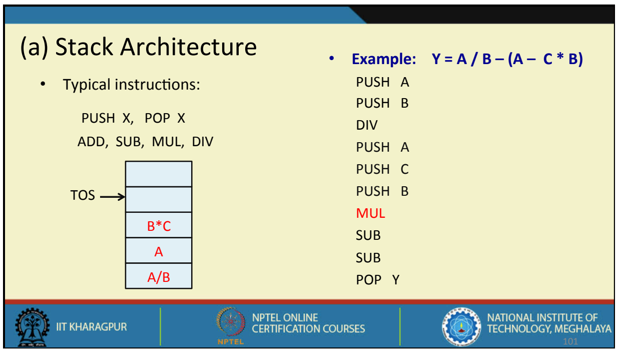
PUSH X, POP X
ADD, SUB, MUL, DIV

TOS →
B
C
A
A/B

- **Example: Y = A / B – (A – C * B)**

```
PUSH  A
PUSH  B
DIV
PUSH  A
PUSH  C
PUSH  B
MUL
SUB
SUB
POP  Y
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA 100



(a) Stack Architecture

- Typical instructions:

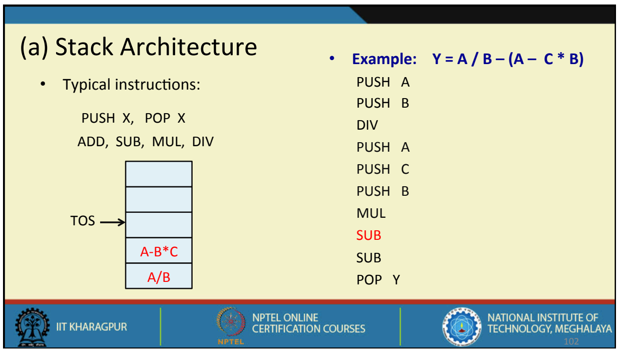
PUSH X, POP X
ADD, SUB, MUL, DIV

TOS →
B*C
A
A/B

- **Example: Y = A / B – (A – C * B)**

```
PUSH  A
PUSH  B
DIV
PUSH  A
PUSH  C
PUSH  B
MUL
SUB
SUB
POP  Y
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA 101



(a) Stack Architecture

- Typical instructions:

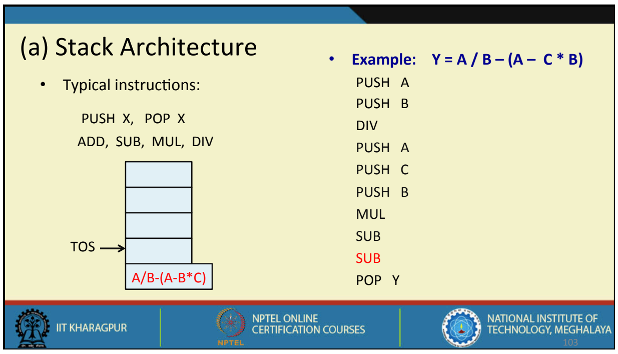
PUSH X, POP X
ADD, SUB, MUL, DIV

TOS →
A-B*C
A/B

- **Example: Y = A / B – (A – C * B)**

```
PUSH  A
PUSH  B
DIV
PUSH  A
PUSH  C
PUSH  B
MUL
SUB
SUB
POP  Y
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA 102



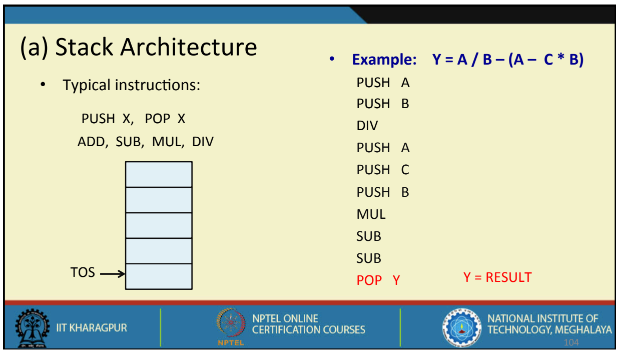
(a) Stack Architecture

- Typical instructions:

```
PUSH X,  POP X
ADD, SUB, MUL, DIV
```

- **Example: Y = A / B – (A – C * B)**

```
PUSH  A
PUSH  B
DIV
PUSH  A
PUSH  C
PUSH  B
MUL
SUB
SUB
POP  Y
```



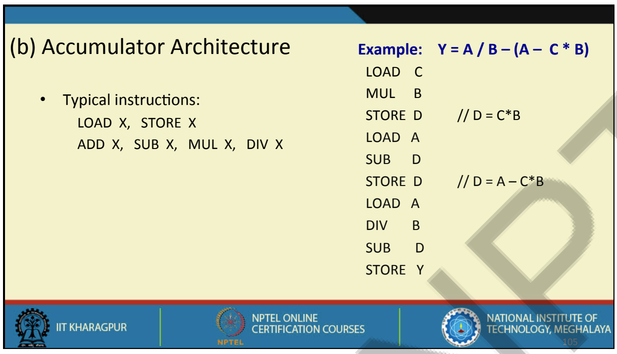
(a) Stack Architecture

- Typical instructions:

```
PUSH X,  POP X
ADD, SUB, MUL, DIV
```

- **Example: Y = A / B – (A – C * B)**

```
PUSH  A
PUSH  B
DIV
PUSH  A
PUSH  C
PUSH  B
MUL
SUB
SUB
POP  Y          Y = RESULT
```



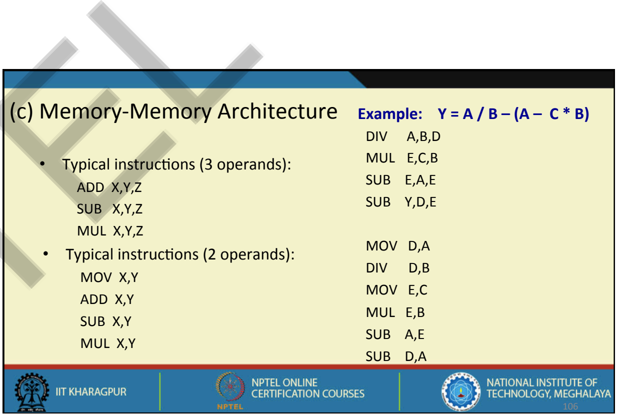
(b) Accumulator Architecture

- Typical instructions:

```
LOAD X,  STORE X
ADD X,  SUB X,  MUL X,  DIV X
```

Example: Y = A / B – (A – C * B)

```
LOAD   C
MUL    B
STORE  D       // D = C*B
LOAD   A
SUB    D
STORE  D       // D = A – C*B
LOAD   A
DIV    B
SUB    D
STORE  Y
```



(c) Memory-Memory Architecture

- Typical instructions (3 operands):

```
ADD  X,Y,Z
SUB  X,Y,Z
MUL  X,Y,Z
```

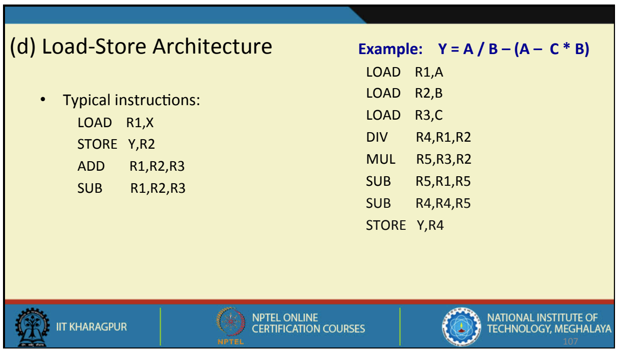
- Typical instructions (2 operands):

```
MOV  X,Y
ADD  X,Y
SUB  X,Y
MUL  X,Y
```

Example: Y = A / B – (A – C * B)

```
DIV    A,B,D
MUL    E,C,B
SUB    E,A,E
SUB    Y,D,E

MOV    D,A
DIV    D,B
MOV    E,C
MUL    E,B
SUB    A,E
SUB    D,A
```



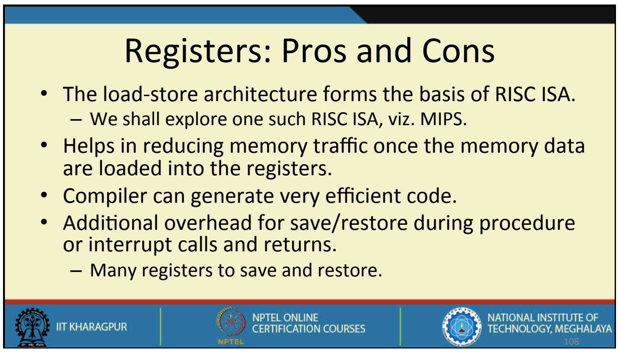
(d) Load-Store Architecture

- Typical instructions:

```
LOAD   R1,X
STORE  Y,R2
ADD    R1,R2,R3
SUB    R1,R2,R3
```

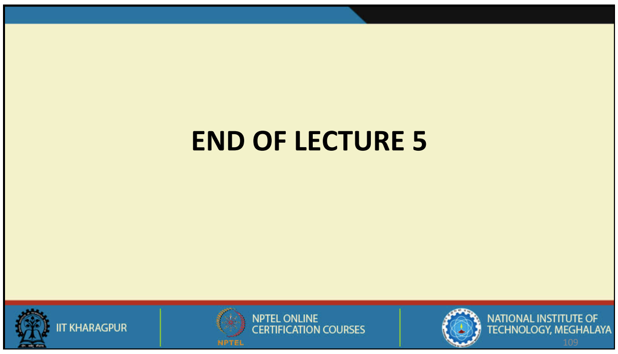
Example: Y = A / B – (A – C * B)

```
LOAD   R1,A
LOAD   R2,B
LOAD   R3,C
DIV    R4,R1,R2
MUL    R5,R3,R2
SUB    R5,R1,R5
SUB    R4,R4,R5
STORE  Y,R4
```



Registers: Pros and Cons

- The load-store architecture forms the basis of RISC ISA.
  - We shall explore one such RISC ISA, viz. MIPS.
- Helps in reducing memory traffic once the memory data are loaded into the registers.
- Compiler can generate very efficient code.
- Additional overhead for save/restore during procedure or interrupt calls and returns.
  - Many registers to save and restore.



END OF LECTURE 5

IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

109