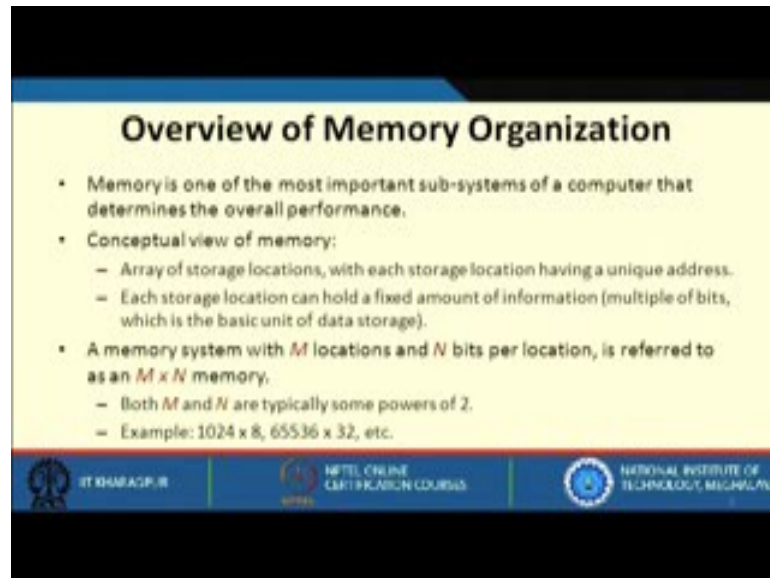


Computer Architecture and Organization
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya


Lecture - 03
Memory Addressing and Languages

(Refer Slide Time: 00:29)



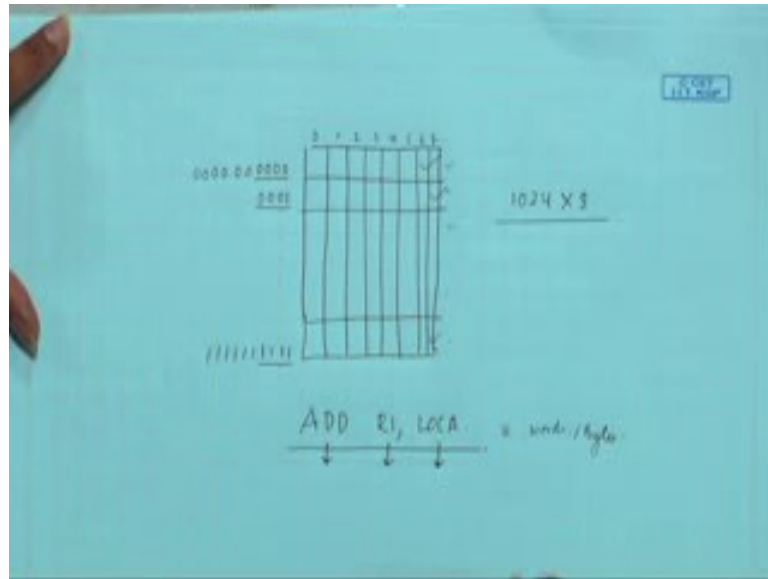
Overview of Memory Organization

- 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 $M \times N$ memory.
 - Both M and N are typically some powers of 2.
 - Example: 1024×8 , 65536×32 , etc.



Welcome to the third lecture on memory addressing and languages. So, let us know about the overview of memory organization. What is memory? Memory is one of the most important subsystems of a computer that determines the overall performance. What do you mean by that? See as you have seen in the previous lecture that we are storing the instruction and data in the memory. If your memory is slower then loading the data from the memory will be slower. So, in that case we need to have a good speed memory. The conceptual view of memory is it is an array of storage locations with each storage location having a unique address. So, it is an array of memory locations.

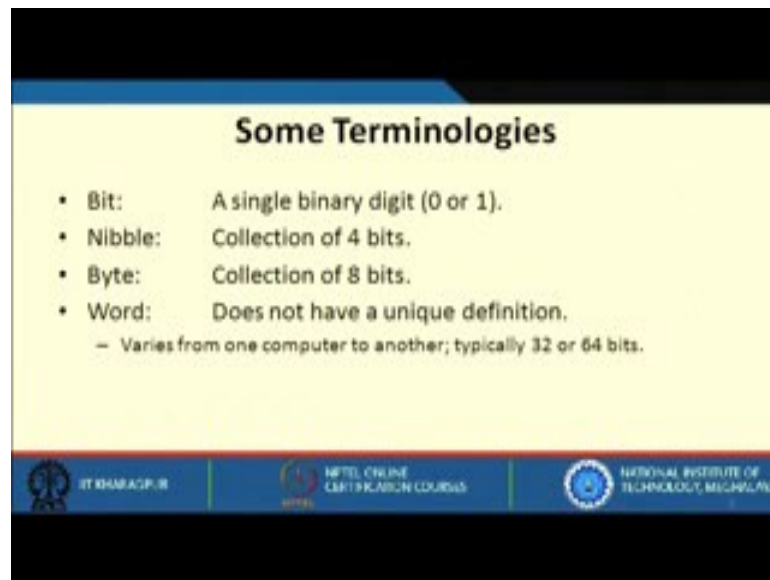
(Refer Slide Time: 01:30)



So, as I said it is an array of memory locations. So, we have first location as 0 0 0 0, next location as 0 0 0 1 and so on, maybe the last location is 1 1 1 1. So, it is an array of storage location each with a unique address. So, these are individual locations and this is the address associated with each location. And each storage location can hold a fixed amount of information, which can be 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, where both M and N are typically some powers of 2. An example: 1024 x 8.

So, if I say 1024 x 8, it means we have 10-bit in the address, and each location is having 8-bit. So, you can store these many locations in these many locations; this shows how a memory will look like.

(Refer Slide Time: 03:53)



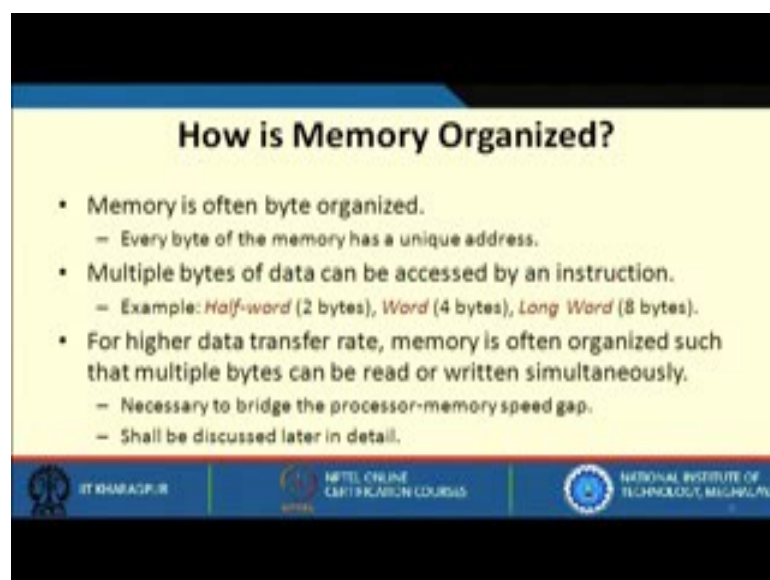
Some Terminologies

- **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.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, BHUBANESWAR

Now, some terminologies you must know when we talk about memory. What is a bit, we all know a bit is a single binary digit either 0 or 1. Nibble is a collection of 4 bits. Byte is a collection of 8 bits. And word does not have a unique definition because we can either have a 32 bit word length or 64 bit word length. So, word does not have a unique definition.

(Refer Slide Time: 04:29)



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.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, BHUBANESWAR

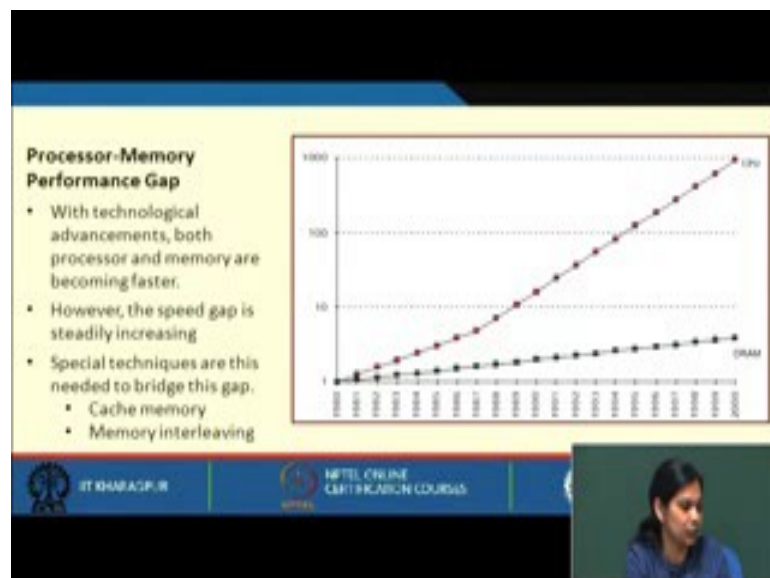
Now let us see how is memory organized. Memory is often byte organized. So, we never say that each bit is having an address, rather we say each byte is having an address, that

means every byte of the memory has a unique address. And multiple bytes of a data can be accessed by an instruction. I will just take an example: ADD R1,LOCA. So, if you consider this instruction, it is depending on how many bits this ADD will have, how many bits this register will have, and how many bits this location will have; this will define that how many words this instruction will have or how many bytes this instruction will have.

So, in that sense what I am trying to say is that how many bytes this instruction will take is dependent on various other factors like the total number of instructions available in your computer. The total number of registers present in your computer, and also the number of locations you are having based on which you can determine the number of bytes required to represent this particular instruction.

For higher data transfer rate, memory is often organized such that multiple bytes can be read or written simultaneously. This is basically needed to bridge the processor memory speed gap; we shall of course discuss this later, but I will just tell very briefly about this memory processor speed gap. So, as you know that processor speed is increasing memory speed is also increasing, but not at this pace the processor is increasing.

(Refer Slide Time: 07:06)

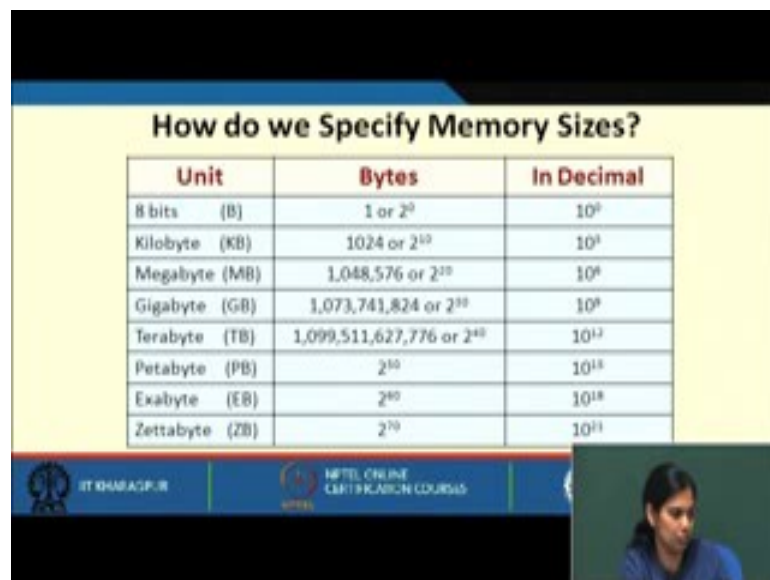


So, this picture will show you the processor memory performance gap. See with technological advancement both processor speed is increasing and also memory speed is increasing; however, there is a speed gap which is steadily increasing. So, earlier the

speed gap was much less, but now with technological advancement CPU speed has increased to a greater extent; memory speed has also increased, but not at the same pace as the CPU. So, we can see this.

So, some special techniques are used to bridge this gap. We will see this in the memory module design, where the concept of cache memory and memory interleaving will be talked about, but from this we can see what we can say is that there is a still huge gap between the speed of a processor and speed of your memory. So, this is where we have to agree upon that. We are still in the phase we are growing we are trying to make certain techniques to bridge this gap, but still this gap exists.

(Refer Slide Time: 08:40)



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}

Now, how do you specify memory sizes. Memory sizes can be 8 bit which is a byte. It can be kilobyte 2^{10} ; it can be megabyte 2^{20} ; gigabyte 2^{30} ; terabytes 2^{40} , and many more like petabyte, exabyte and zettabyte.

(Refer Slide Time: 09:09)

• 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 (n bits)
Data (m bits)

RD WR EN

IT KANAKPUR | NETAJI ONLINE CERTIFICATION COURSES

Now, you see if there are n bits in an address, the maximum number of storage locations that can be accessed is 2^n .

(Refer Slide Time: 09:31)

$n=3$ $2^3 = 8$

000
001
010
011
100
101
110
111

3x8

This is a small example, we will take $n = 3$. So we can say how many locations we can access; $2^3 = 8$. So, the first location will be 0 0 0, the next location will be 0 0 1, next will be 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0 and 1 1 1. So, with n bits we can have 2^n locations that can be accessed.

So, if we have 3 bit in the address, so maximum location that can be accessed is 8. So, for $n = 8$, 256 locations (2^8); for $n = 16$, $2^{16} = 64K$ locations; for $n = 20$, 1M locations, etc. can be accessed. So, this diagram shows the address bits; if you have n bit address we can have 2 to the power n locations that can be accessed. And modern-day memory chips can store several gigabytes of data that is our dynamic RAM. We will be looking into more details about each and every aspect of memory module.

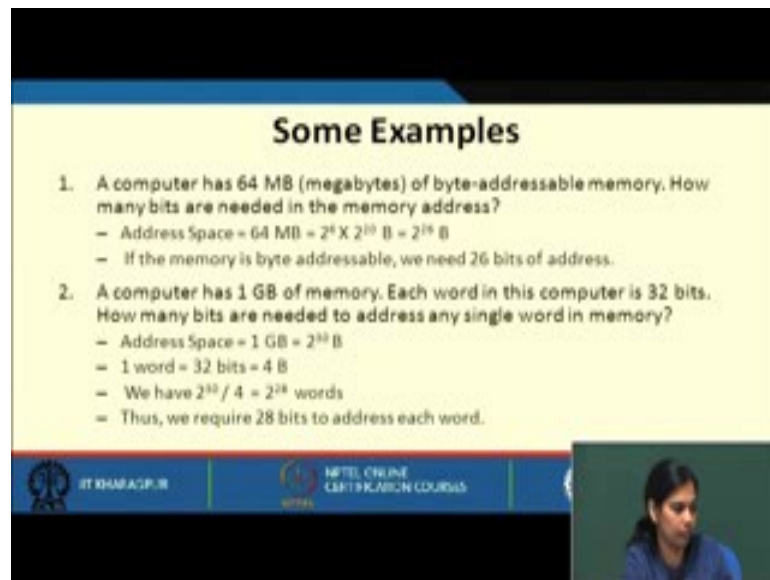
(Refer Slide Time: 11:33)

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: $2^8 \times 16$ memory

Now, as I said for an 8 bit address, 2 to the power 8 unique locations will be there. The first locations will be all 0s, and the last location will be all 1s; and each of these locations again will have some content. So, consider an example of $2^8 \times 16$ memory. So, in each of these locations we will have some data which is 16 bits.

(Refer Slide Time: 12:02)



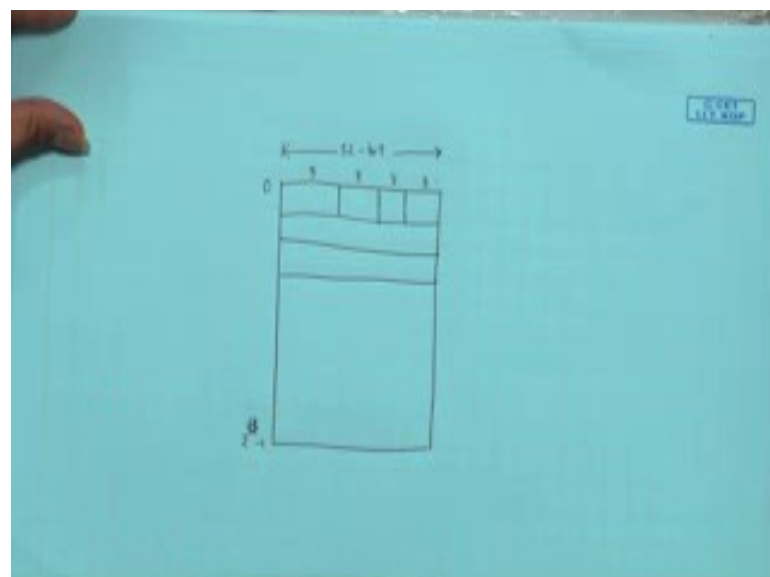
Some Examples

1. A computer has 64 MB (megabytes) of byte-addressable memory. How many bits are needed in the memory address?
 - Address Space = 64 MB = $2^4 \times 2^{20}$ B = 2^{24} 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.

IT BHARATPUR | NITEL ONLINE LECTURE COURSES

Let us see a computer with 64 MB of byte addressable memory. How many bits are needed in the memory address? As I already said, that $64 \text{ MB} = 2^{26}$; that is, we need 26 bits to represent the address. Now, let us take another example where we say a computer has 1 GB of memory. So, we are saying total of 1 GB of memory, each word in this computer is 32 bit.

(Refer Slide Time: 13:18)



So, $1 \text{ GB} = 2^{30}$. If each word is 32 bits, that means 8, 8, 8, 8. So, total words possible will be $2^{30} / 4 = 2^{28}$. So, we require 28 bits, with address from 0 to $2^{28}-1$. If it is byte addressable, each byte can be accessed with address from 0 to $2^{30}-1$.

(Refer Slide Time: 15:02)


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

 IIT KHARAGPUR

 NPTEL ONLINE CERTIFICATION COURSES

 NATIONAL INSTITUTE OF TECHNOLOGY, WARDHA

Now, let us also understand what is byte ordering convention. Many data items require multiple bytes for storage. And different computers use different data ordering convention, it is known as low order byte first and high order byte first. So, these two are called basically Little Endian and Big Endian. So, you see this data type character is 1 byte, integer is 4 byte, long integer is 8, floating point 4 and double precision is 8. Thus if you have a 16 bit number which is represented like this, so in one way this is the total number high order bit is stored in high order address, and the low order is stored here and so on here it is stored differently. This is stored first and this is stored next.

(Refer Slide Time: 16:13)

The slide contains the following text:

- 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.

The slide footer includes logos for IIT Kharagpur and NPTEL Online Certification Course.

So, let us see the convention that has been named as little endian. The least significant byte is stored at lower address followed by most significant byte. So, Intel processors DEC alpha they all use little endian method, where again I repeat least significant byte is stored at lower address. Now, same concept follows for arbitrary multi-byte. So, if you also want to store multi byte then the same thing will follow there also. Now, in big endian the most significant byte is stored at lower address followed by least significant byte. So, the most significant byte is stored at lower address followed by least significant byte. IBM's 370 mainframe uses big endian concept.

(Refer Slide Time: 17:14)

The slide contains the following text:

An Example

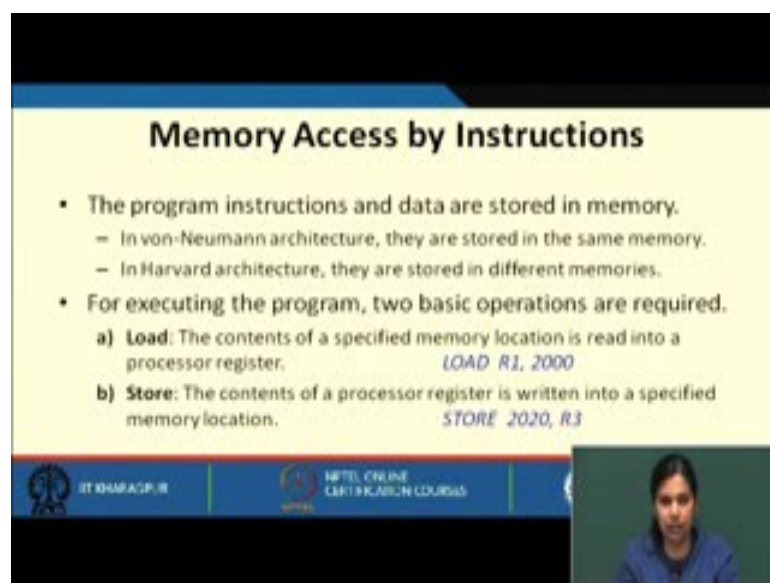
- Represent the following 32-bit number in both Little-Endian and Big-Endian in memory from address 2000 onwards:
01010101 00110011 00001111 11000011

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

The slide footer includes logos for IIT Kharagpur and NPTEL Online Certification Course.

Now, let us see this representation with this example of a 32-bit number both in little endian and big endian. So, this is the number and lower byte is stored in lower address, then the next byte, then the next byte, and then the next byte. And here the higher order address higher order byte is stored lower address, then the next byte then the next byte and then this one. So, just see the difference between these two in little endian what we are storing; this is the least significant byte, we are storing in the least address that is 2000 and then the next one, next one, next one. In a similar way, the most significant byte we are storing it in the lower address and so on.

(Refer Slide Time: 18:23)



Memory Access by Instructions

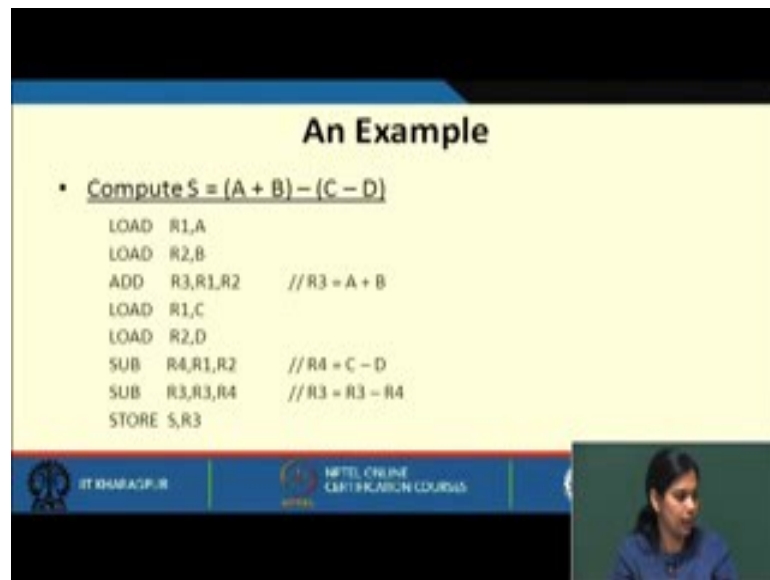
- The program instructions and data are stored in memory.
 - In von-Neumann architecture, they are stored in the same memory.
 - In Harvard architecture, they are stored in different memories.
- For executing the program, two basic operations are required.
 - a) **Load:** The contents of a specified 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`

IT KANMAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let us see memory access by instructions. Now, as I said that the program and instruction in the program instructions and the data are stored in memory. So, there are two basic ways how this can be stored. One is von-Neumann architecture, they are stored same in the same memory both the program and the data are stored in same memory. In Harvard architecture, they are stored in different memories.

So, for executing the program two basic operations are required. What are the two basic operation we need to know this. Load the content of specified memory location. So, `LOAD R1,2000` means load the content from memory location 2000 into processor register R1. Another one is `STORE R3,2010` that means store the content of R3 into this specific location 2020. So, either we load a data from memory into one of the processor registers, or we store the data from processor register to some location in memory.

(Refer Slide Time: 20:11)



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
```

IT BHARATPUR | NIEL ONLINE CERTIFICATION COURSES

Now, let us take an example. Suppose we need to execute this particular instruction. We need to compute this. S, A, B, C and D are stored in memory. So, what we need to do is we need to load individual data, that is A and B and then only we can execute it. So, let us see what are the steps that are required to execute this. First of all let us do $A + B$. For this, I need to load first A into some processor register, B into some processor register and then add them. So, LOAD R1,A will load the content of A into R1; LOAD R2,B will load the content of B into R2. The ADD instruction will add the content of R1 and R2, and store the result in R3. So, this portion is done.

Next I load C into R1, D into R2, then I subtract C, I subtract D from C, and then I store the result in R4. Now, $A + B$ is stored in R3, $C - D$ is stored in R4, and now I need to subtract them. SUB R3,R3,R4 will cause the result to be stored in R3. But finally, I have to store the result in another location, that is S. So, what I should do now I have to store the content of R3 into S.

(Refer Slide Time: 22:58)

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.

IT KHAMARPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us also understand what is machine language, assembly language, and high level language. Machine language is native to a processor and it is executed directly by in hardware. So, it only consists of binary code 1s and 0s.

(Refer Slide Time: 23:29)

ADD R1, LOCA

01001, 0010, 000001001000 → 0515

ADD R1, LOCA

ADD R1, R2
MUL R2, R3

A = [1, 2, 3, 4, 5, 6, 7, 8]

```
for (i=0; i<10; i++)  
{  
    Sum = Sum + A[i];  
}
```

So, I have talked about this if you remember that let say this is my instruction. Again I take the same example ADD R1,LOCA and as I said this can be 01001 (5-bit), this can be a 4-bit number, and this can be a 12-bit number. So, what I am specifying I am specifying the entire instruction in a sequence of 0s and 1s. So, when I specify an

instruction in the form of 0s and 1s, is called machine language, which is native to processor executed directly by hardware.

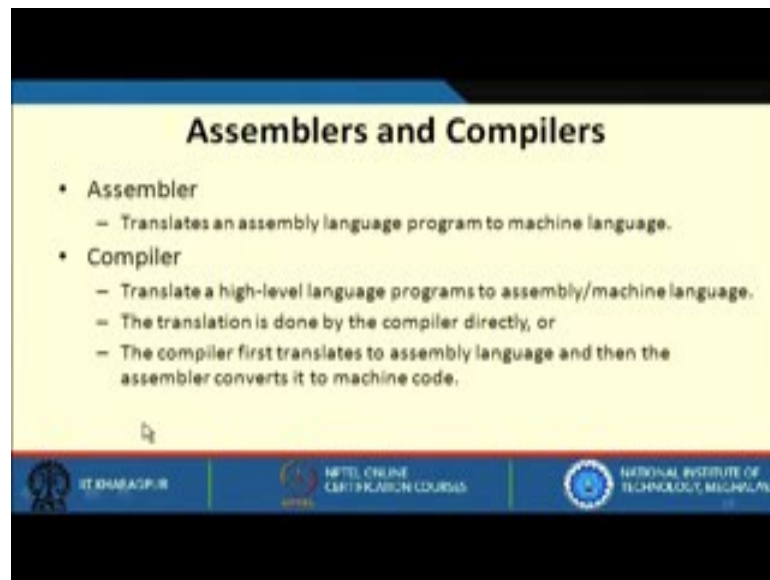
Next is assembly language. It is also a low-level symbolic version of machine language. Instead of 0s and 1s, I write it symbolically as `ADD R1,LOCA`. When I represent something symbolically; instead of writing 0s and 1s, I am writing it with some mnemonics.

So, this language is a low-level symbolic version of machine language; one to one correspondence with machine language is there. Pseudo instructions are used that are much more readable and easier to use. What do you mean by much more readable and easier to use? That means it will be difficult for me to remember `0110` is add, but it will be very much easy for me to remember the mnemonic `ADD`. So, I can write an instruction `ADD R1,R2`. So, this is much easier way to represent a program. So, assembly language is nothing but some kind of one-to-one correspondence with machine language.

Next comes to high-level language. Now, you see if you have to add ten numbers or you have to sort some list in an array, you need to perform certain operation. Again writing such kind of language where we say that first sum of 10 numbers you have to initialize it and then you have to repeatedly add it. So, there will be set of more instruction that are required to perform an add operation; rather for adding 10 numbers, I can very easily write let say for `(i=0; i<10; i++)`, `sum = sum + a[i]`. So, this is much more easier to code.

So, generally high-level languages basically the programming languages like C or C++ or Java are used which are much more readable and closer to human languages. So, we can we can write a program in a high-level language and that can be executed by your machine.

(Refer Slide Time: 29:04)



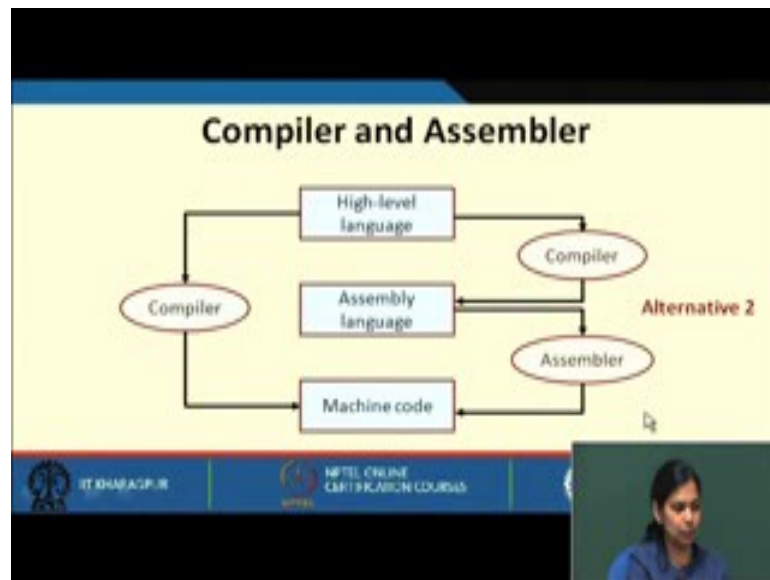
Assemblers and Compilers

- **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.

IT BHARATPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, BHOPAL

Now, if you have to execute a high-level language then you need some kind of translator that should translate your high-level language to assembly language or machine level language. So, these are the two things which are required, one is assembler that translates an assembly language program to machine language; and compiler that translates a high level language program to assembly or machine language. So, the translation is done by compiler directly or the compiler first translates to assembly language, and then an assembler can convert to a machine language, but ultimately you need to have machine language code to execute your instruction. So, any high-level language you write ultimately you have to boil down it to machine language that can only be understood by your computer.

(Refer Slide Time: 30:10)



So, as I said this compiler you have high-level language and then you can directly have a compiler that will generate an assembly language. And then that assembly language will be fed to an assembler that will generate a machine language or directly you can have a compiler that will generate your machine language, these are the best two alternatives that happens.

(Refer Slide Time: 30:39)

- 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.

Now, we you can also have something called cross-assembler or cross-compiler. What it does like the compiler or assembler may run on a native machine for which the target

code is being generated or can run on any other machine that means. Take an example where you have an 8085 cross-assembler which is running in your desktop machine. And what it generates is some 8085 machine codes. Similarly, an ARM embed-C compiler which is available online may be running on a desktop PC which generates ARM machine code for the embedded development board.

So, by this I will end lecture 3. So, in this lecture what we have discussed is how memory is organized, what are the ways to store the data in the memory. And we also looked into what are the steps that are required to transfer an instruction from one level to another like from high-level language, how it is converting into machine language through some compiler, how from assembly language it is converted into machine language and what is a machine language.

Thank you.