




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

Lecture - 07
Instruction Format And Addressing Modes

(Refer Slide Time: 00:27)

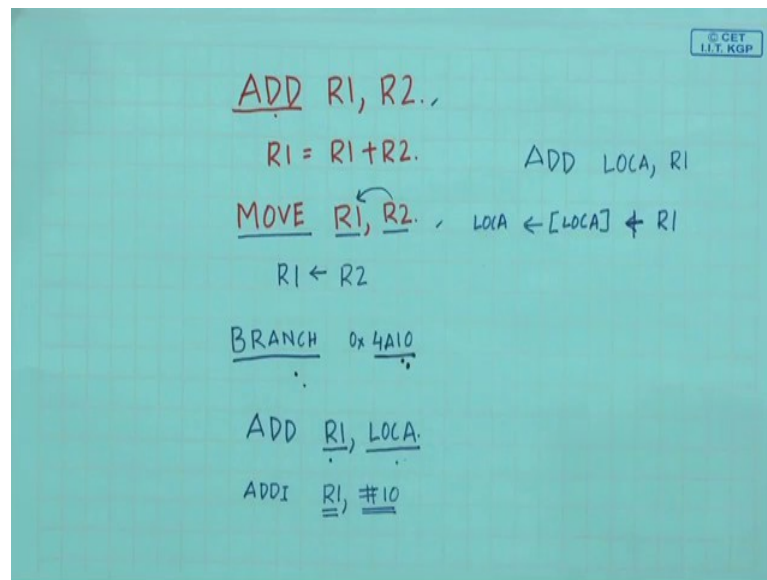
Instruction Format

- An instruction consists of two parts:-
 - a) **Operation Code or Opcode**
 - Specifies the operation to be performed by the instruction.
 - Various categories of instructions: data transfer, arithmetic and logical, control, I/O and special machine control.
 - b) **Operand(s)**
 - Specifies the source(s) and destination of the operation.
 - Source operand can be specified by an immediate data, by naming a register, or specifying the address of memory.
 - Destination can be specified by a register or memory address.

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

Welcome to the next lecture on instruction format and addressing modes. So, what do you mean by instruction format? We know what is an instruction. And instruction format is what it comprises of: two parts --- first part is the Opcode, and the next part is the Operand.

(Refer Slide Time: 00:57)



So, what is an opcode? Take an example `ADD R1,R2`. The opcode specifies the operation to be performed. The operation here is adding two register values, and result stored back in some register. So, R1 will store $R1 + R2$. `ADD` is the operation code that specifies the operation to be performed by the instruction and we can have various categories of instruction. This is an arithmetic instruction.

We can have an instruction called `MOVE`. What this instruction will do? This instruction will move the data from R2 to R1. So, here R1 will have the value of R2. Such kind of instruction is called data transfer instructions. We can have other instructions; this is arithmetic instruction, this is data transfer instruction, we can have other branching instruction. What is branching instruction? We can have an instruction called branch to some location, say, 16-bit hexadecimal number 4A10.

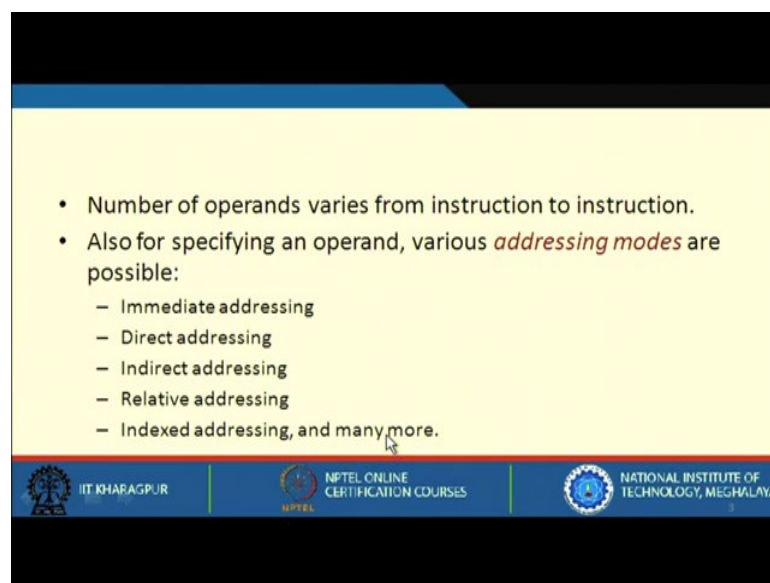
So, branch to this particular location will move to this particular location, and whatever data is there in this particular location it will be added with PC and it will calculate the next instruction that needs to be executed because branch to this location means in some particular location some instruction is present which we need to execute. So, this part of the instruction we call it an opcode, and this part is the operand.

Now, see what can be an operand; it specifies either a single source or there can be two source and a destination of the operation. And source operand can be specified by an immediate data or by naming a register. Just now I have shown how we can just give the

name of the register or specify a memory address like ADD R1,LOCA. Here we are specifying one operand is a register, another operand a memory location. So, an operand can be a register, a memory location or an immediate value (here 10).

So, this kind of operand can also be specified, but this operand cannot be the destination. A destination operand should always be either a register or a memory location like LOCA. So, instruction consists of two parts; one is operation code, another will be operand.

(Refer Slide Time: 06:13)

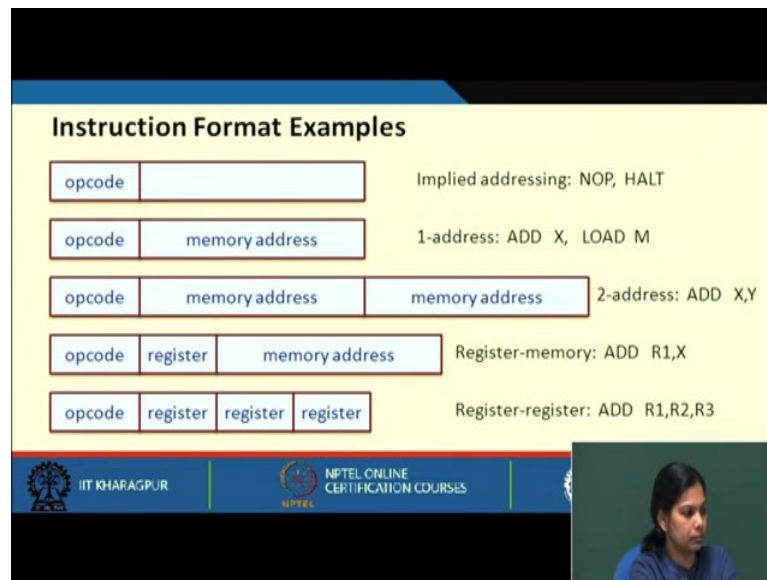


The slide features a yellow background with a blue header and footer. The main content area contains a bulleted list of addressing modes. The footer includes logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA.

- Number of operands varies from instruction to instruction.
- Also for specifying an operand, various *addressing modes* are possible:
 - Immediate addressing
 - Direct addressing
 - Indirect addressing
 - Relative addressing
 - Indexed addressing, and many more.

The number of operands varies from instruction to instruction. We can have a zero address instruction, we can have a one address instruction, we can even have a two address or even have a three address instruction. So, number of operands that are present in an instruction may vary. Also while specifying an operand, we need to know the various addressing modes. So, coming to what is addressing modes we will be looking in more details. Addressing mode actually is a way by which the location of the operand is specified in the instruction. There can be many possible addressing modes: immediate, direct, indirect, relative and index, and many more. We will be seeing a few of them.

(Refer Slide Time: 07:15)



Now, let us see this instruction format. If we have just the opcode, let us take some example of NOP, NOP means no operation. No operation instruction specifies the processor that no operation will be performed at this particular cycle. HALT will specify to halt the execution. We can have one-address instruction where only one address is specified along with the opcode. We can have two-address instruction where we can specify two operands. We can have two-address instruction where both operands can be memory locations. We can have another instruction where one can be register another can be memory operation, or we can have a three-address instruction where all are registers. So, these are various instruction formats.

(Refer Slide Time: 08:39)

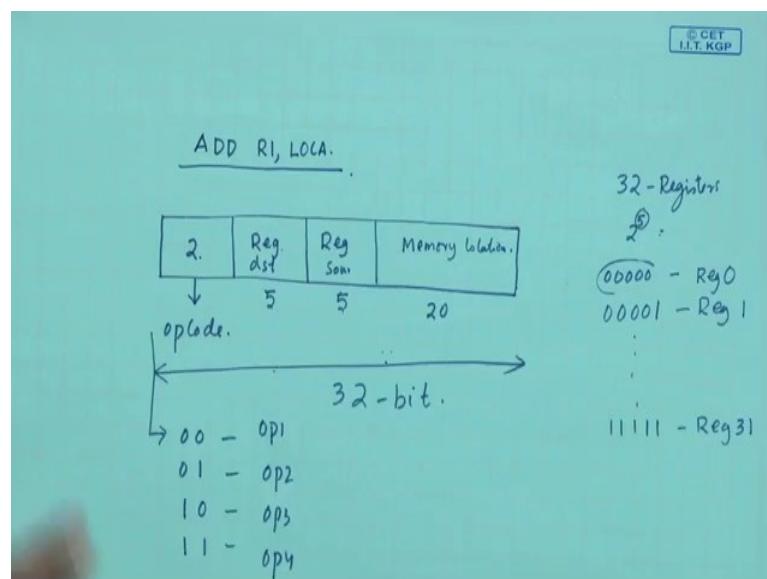
A 32-bit Instruction Example

- Suppose we have an ISA with 32-bit instructions only.
 - Fixed size instructions make the decoding easier.
- Some instruction encoding examples are shown.
 - Assume that there are 32 registers R0 to R31, all of 32-bits.
 - 5-bits are required to specify a register.

HIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

Now, consider a 32-bit instruction example. Suppose our instruction set architecture is having only 32-bit instructions; fixed size instructions make the decoding easier. Let us understand this statement.

(Refer Slide Time: 09:17)



I am giving an example; this is not corresponding to any real machine. Let us say we have an instruction ADD R1, LOCA. So, in this 32-bit instruction, some bits will be reserved for opcode, some bits will be reserved for register, etc. So, this can be register destination, this can be register source, and this can be your memory location. If this is so, the total is

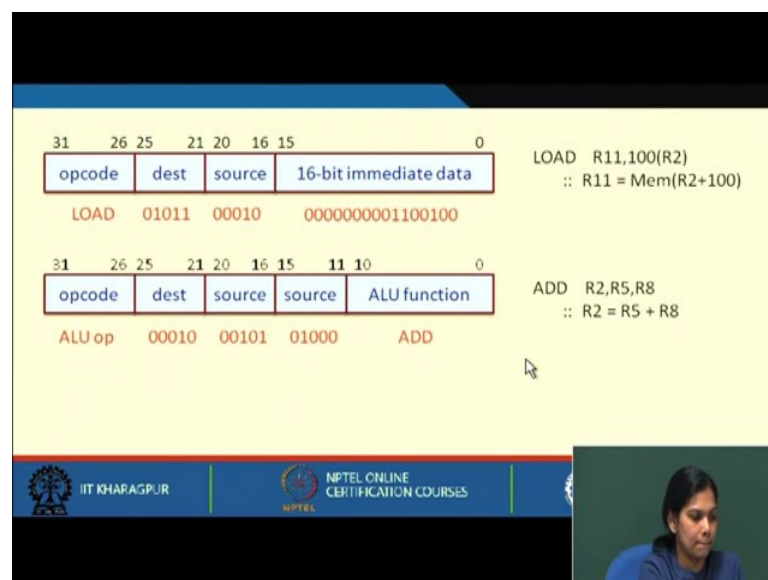
32-bit, and now we have to specify the bits. Let us say we have a total of 32 registers. Then 5-bits we will be required to represent a single register.

So, this 00000 will be register 0, 00001 will be register 1 and so on, and the last register will be register 31. So, 5-bits will be required to specify one of the registers. Also suppose we can specify a memory location in 20 bits. And then how many bits are remaining for the opcode? -- we have 2 bits left for opcode.

So, if we have two bit left for opcode we can have maximum of four possibilities 00, 01, 10 and 11. There can be four operations only. I am just giving you as example and this does not correspond to a real machine, but rather just to give you an idea that how the instruction format will looks like.

Fixed size instructions make the decoding easier. We can check the opcode to know the operation. We can have various kinds of instruction like here we have only one memory location; if we afford we can have two memory locations.

(Refer Slide Time: 14:11)



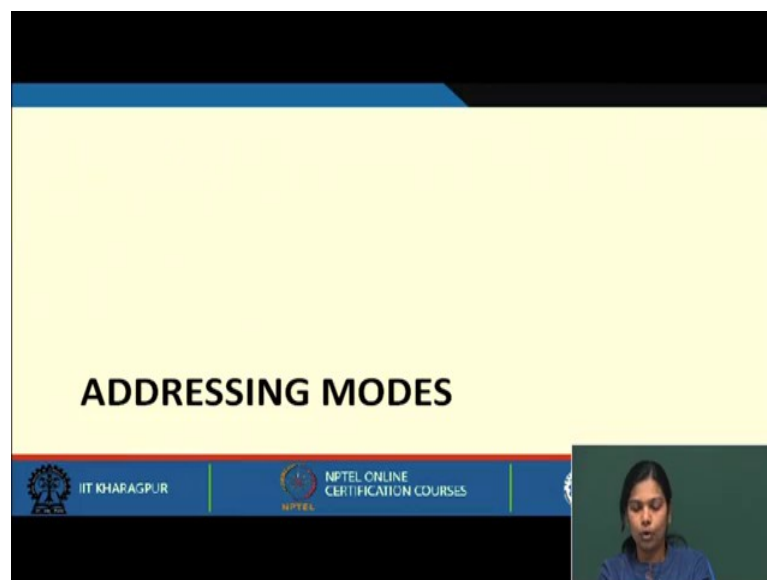
Now let us see some instruction formats. Let say this is the format where you have these bits for opcode 26, 27, 28, 29, 30 and 31 --- 6-bits. And 5-bit for register and 16-bit immediate data. Let us take an example LOAD R11,100(R2). We are loading the content from memory location pointed by 100 + R2. So, first we have to add 100 plus R2, and then we have to put this in MAR, activate the read control signal, get the value of it. And

then we load it into R11. These are the following steps that will be required to execute this instruction.

So, let us see where all it will be placed. This is the destination register R11. So, R11 will be placed here. R11 is 01011. Opcode for load will be loaded in first 6 bits. This is the source operand; one of the source operand is R2; it is loaded here. And 100 is the immediate value which is loaded here. So, this is how we encode this instruction into binary. Again load can be having some value, say 000001.

Let us move on. Now, if we have limited opcode facility, say, we can only have 64 operations that are possible with 6-bit. So, if you want to increase that, what we can do is that here this opcode will give you that what kind of function it will be taking care of like it can be an ALU function, it can be a data transfer operation, etc. And the exact function will be specified by these 11 bits. This is ADD R2,R5,R8. So, the contents of R8 and R5 will be added and will be stored in R2. Now, we see that, this is an ALU operation, this is the destination R2. here are two sources --- first source is R5 and the next source is R8 and this ALU function ADD.

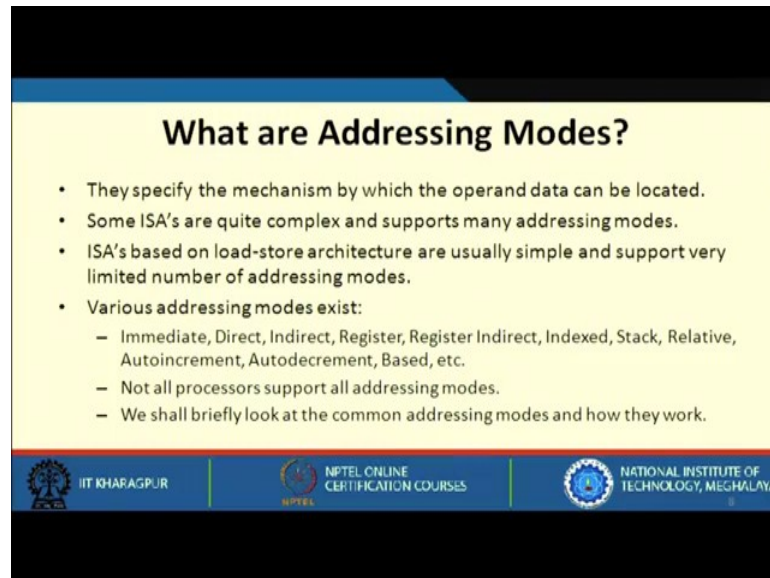
(Refer Slide Time: 17:44)



Now, moving on to addressing modes. Now, see what we have understood till now that we can see this that we have an instruction, and by seeing we are saying that this is a register, this is a memory operation, memory location. But again you have to instruct the computer that see this is a register, this is a memory location then only the processor will

do the required thing, it will go to the memory location, get the data, it will go to a particular register and get the data, etc.

(Refer Slide Time: 18:43)



What are Addressing Modes?

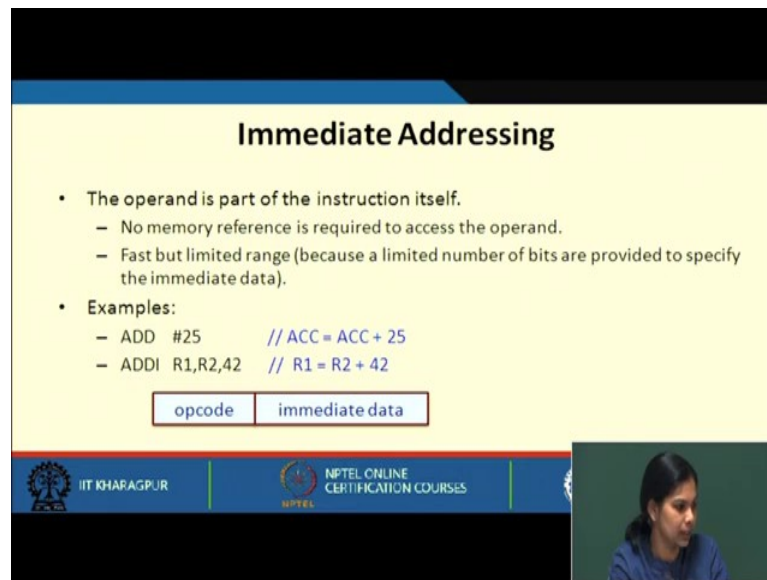
- They specify the mechanism by which the operand data can be located.
- Some ISA's are quite complex and supports many addressing modes.
- ISA's based on load-store architecture are usually simple and support very limited number of addressing modes.
- Various addressing modes exist:
 - Immediate, Direct, Indirect, Register, Register Indirect, Indexed, Stack, Relative, Autoincrement, Autodecrement, Based, etc.
 - Not all processors support all addressing modes.
 - We shall briefly look at the common addressing modes and how they work.

Logos at the bottom: IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

So, what are addressing modes? Addressing modes are the ways by which the location of an operand is specified in an instruction. So, it specifies that the location of an operand in the instruction. How the location of the operand is specified, whether we have to get it from a register or from memory location, etc. Some instruction set architectures are quite complex and supports many addressing modes. But instruction set architectures that are based on load-store usually support very simple addressing modes. So, this is very important. If you want to have complex addressing modes, some of the instructions or architecture do have it, but the load-store architecture basically supports very limited number of addressing modes.

There are various addressing modes that exist: immediate, direct, indirect, register, register indirect, indexed, stack, relative, auto increment, auto decrement, based, etc. However, all architectures will not have all the addressing modes. So, we shall first look into some common addressing modes, and how do they work.

(Refer Slide Time: 20:34)



Immediate Addressing

- The operand is part of the instruction itself.
 - No memory reference is required to access the operand.
 - Fast but limited range (because a limited number of bits are provided to specify the immediate data).
- Examples:
 - ADD #25 // ACC = ACC + 25
 - ADDI R1,R2,42 // R1 = R2 + 42

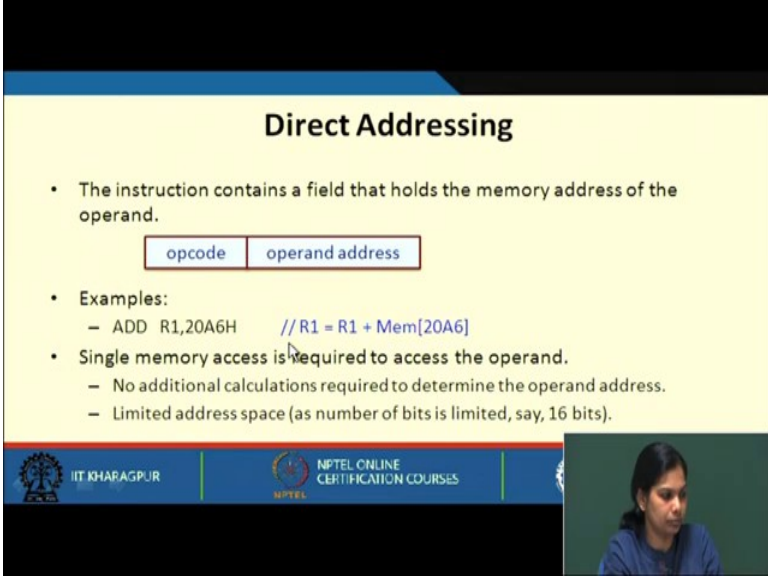
opcode immediate data

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Coming to immediate addressing mode, here the operand is part of the instruction itself. So, you need not have to go anywhere to get the operand, rather your operand is a part of your instruction. So, no memory access is required to get the operand and it is fast, but limited range because you can only specify a limited number using immediate mode like ADD #25. When we write #, it means is an immediate data. So, when we write ADD #25 that means, 25 will be added with accumulator and the result will be stored back in the accumulator.

Here, we have an immediate data 42, which is added to R2 and result stored in R1.

(Refer Slide Time: 22:10)





Direct Addressing


- The instruction contains a field that holds the memory address of the operand.

opcode operand address

- Examples:
 - ADD R1, 20A6H // $R1 = R1 + \text{Mem}[20A6]$
- Single memory access is required to access the operand.
 - No additional calculations required to determine the operand address.
 - Limited address space (as number of bits is limited, say, 16 bits).


IIT KHARAGPUR

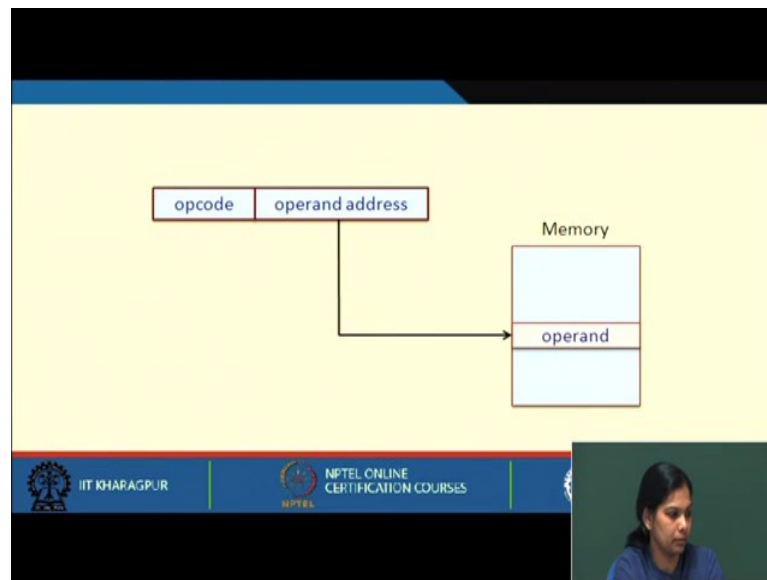

NPTEL ONLINE
CERTIFICATION COURSES



Moving on with direct addressing mode, here the instruction contains a field that holds the memory address of the operand. So, here the content of 20A6 will be the operand that we are looking for. Like here ADD R1, 20A6 means whatever content is in 20A6 will be added with R1 and result stored back in R1.

Now, here how many memory operations are required? So, we have to go to this particular address and fetch the instruction. So, going to this particular address we need one more memory access to access the operand, no additional calculation is required to determine the operand address and limited address space. So, if this address space is 16-bit, it is limited. So, we can only have direct addressing within that 16-bit address span.

(Refer Slide Time: 23:45)



So, this is how pictorially we can show -- this is the opcode, this is the operand address. You go to that address you get that operand, this is direct addressing.

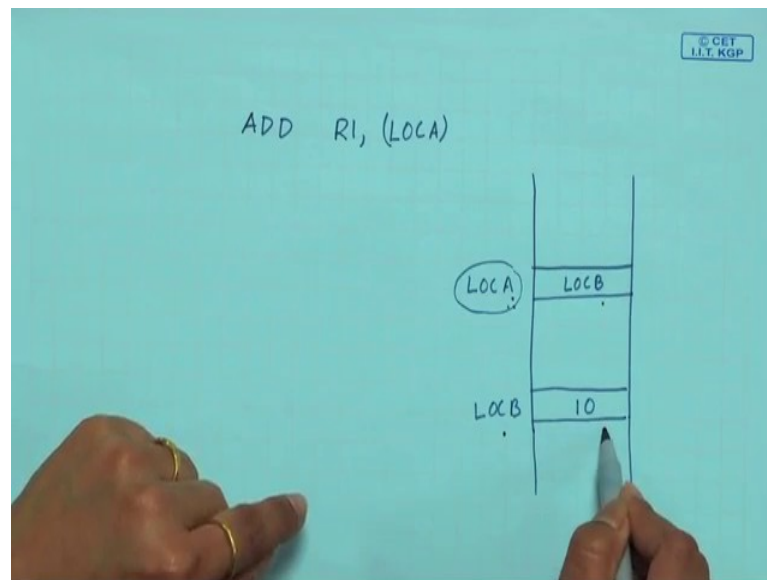
(Refer Slide Time: 24:01)

Indirect Addressing

- The instruction contains a field that holds the memory address, which in turn holds the memory address of the operand.
- Two memory accesses are required to get the operand value.
- Slower but can access large address space.
 - Not limited by the number of bits in operand address like direct addressing.
- Examples:
 - `ADD R1, (20A6H)` `// R1 = R1 + (Mem[20A6])`

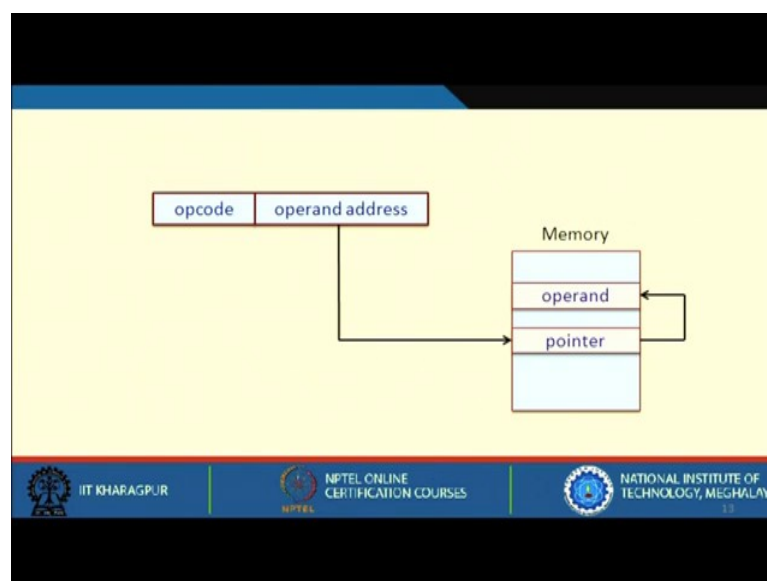
Let us move on indirect addressing. The name itself suggests when it is indirect that means, in the instruction what it contains, it contains a field that holds the memory address which in turn holds the memory address of the operand. So, let us see this with an example.

(Refer Slide Time: 21:41)



Let us say we have an instruction ADD R1,(LOCA). LOCA contain another address, say LOCB. And now you will not get your operand from LOCA, rather you will get your operand from LOCB. In this particular case, you have to go to this location, this location will give you another location and you go to that particular location that will give you the value. So, in this case you if you can see that you are requiring two memory accesses to get the operand value. This is slower, but can access larger address space. It is not limited to number of bits in the operand address like direct addressing.

(Refer Slide Time: 26:27)



So, this is the operand address. First this is a pointer as I explained and then from there you go to another address which will give you the exact operand.

(Refer Slide Time: 26:40)

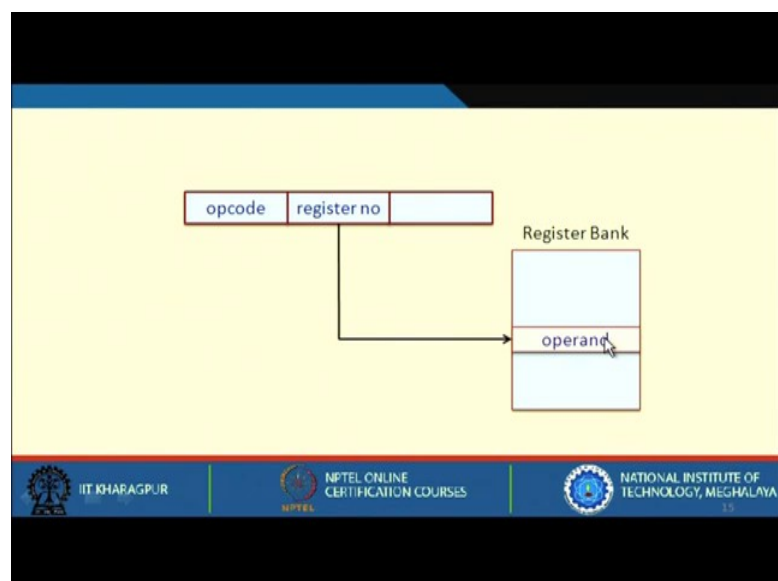
Register Addressing

- The operand is held in a register, and the instruction specifies the register number.
 - Very few number of bits needed, as the number of registers is limited.
 - Faster execution, since no memory access is required for getting the operand.
- Modern load-store architectures support large number of registers.
- Examples:
 - `ADD R1,R2,R3` `// R1 = R2 + R3`
 - `MOV R2,R5` `// R2 = R5`

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

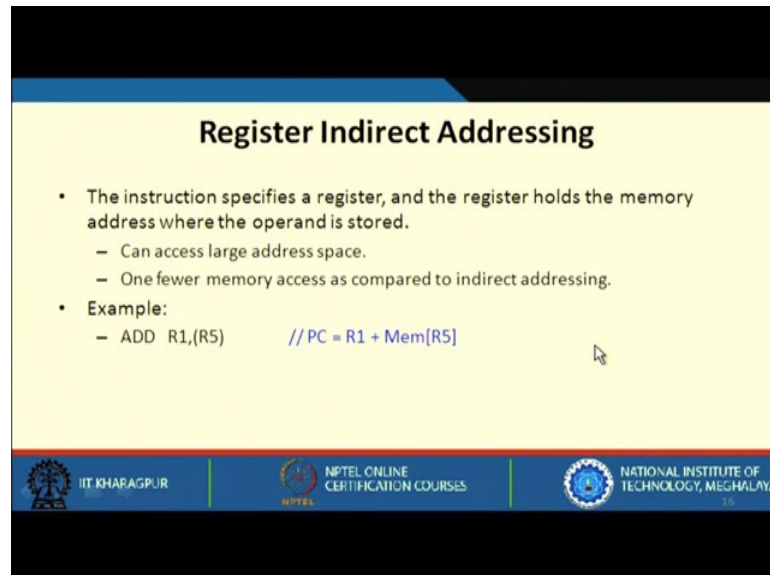
Moving on with register addressing. The operand is held in a register, and the instruction specifies the register number. Very few bits are needed, as the number of registers is limited. Faster execution is possible, and no memory access is required for getting the operand. The load-store architecture supports large number of registers.

(Refer Slide Time: 27:31)



So, as I said this is the register bank. The register number is specified in the instruction and you go to that particular register to get the operand.

(Refer Slide Time: 27:44)



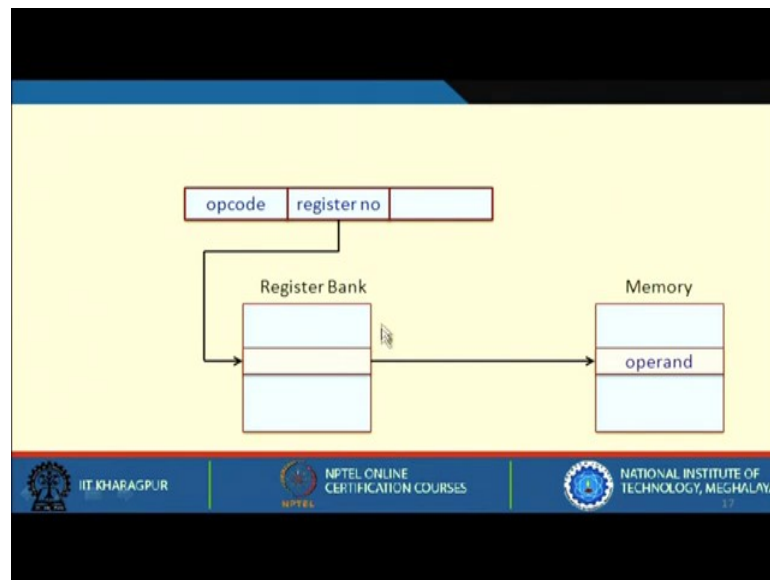
Register Indirect Addressing

- The instruction specifies a register, and the register holds the memory address where the operand is stored.
 - Can access large address space.
 - One fewer memory access as compared to indirect addressing.
- Example:
 - `ADD R1,(R5)` `// PC = R1 + Mem[R5]`

The slide features a yellow background with a blue header and footer. The footer contains logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA.

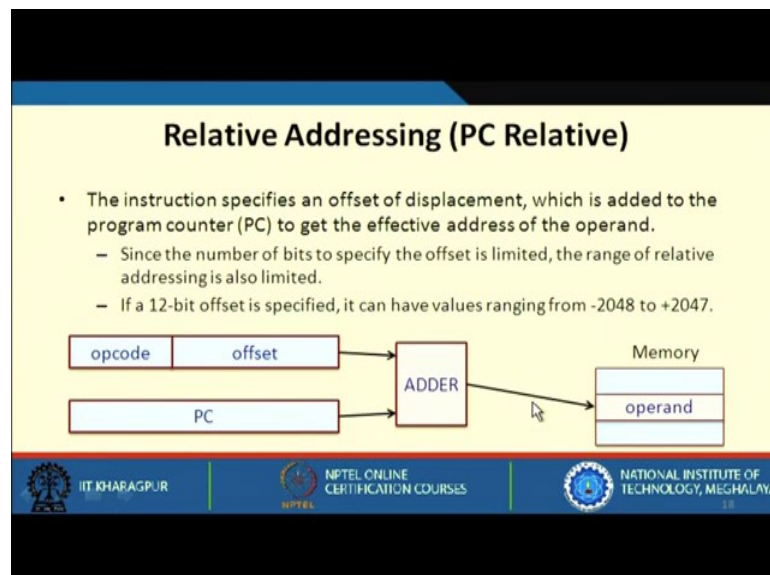
Moving on with register indirect addressing mode, here the instruction specifies a register and the register holds the memory address where the operand is stored. So, this is also a kind of indirect addressing, where instead of a memory location here we are putting the memory address in a register. And then this register holds what it holds a memory address, but not the operand, you have to go to that memory address to access the operand. One fewer memory access is required as compared to indirect addressing mode.

(Refer Slide Time: 28:59)



So, just see here this register will give you a memory location, and this memory location is fetched from the memory, the data from this memory location that is in the register is fetched from the memory, and we get the operand. So, this is register indirect addressing. In the register, we are putting an address and that address stores the operand.

(Refer Slide Time: 29:31)



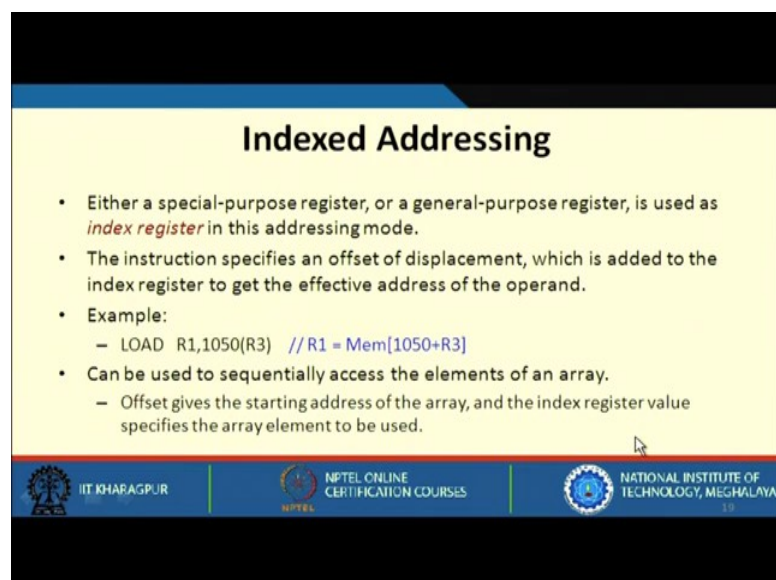
Relative addressing is always with respect to PC. In this kind of addressing modes the instruction specifies an offset or displacement, which is added to the program counter to get the effective address of the operand. Since the number of bits to specify the offset is

limited, the range of relative addressing is also limited. So, if a 12-bit offset is specified, it can have values ranging from -2048 to +2047.

Let us understand this relative addressing. With respect to PC that means, relative to PC how much you can go. So, in branch instruction we specify a branch address. To go to that particular location, you have to load that particular address in PC. So, this is an offset that is given in the instruction that is added or subtracted depending on where you are branching. That particular branch address will be added with the content of the PC. So, in such kind of cases we require relative addressing mode.

So, here you have an opcode, this is the offset. The offset is added with the content of PC and then where you go and you fetch the operand.

(Refer Slide Time: 31:45)



Indexed Addressing

- Either a special-purpose register, or a general-purpose register, is used as *index register* in this addressing mode.
- The instruction specifies an offset of displacement, which is added to the index register to get the effective address of the operand.
- Example:
 - `LOAD R1,1050(R3) // R1 = Mem[1050+R3]`
- Can be used to sequentially access the elements of an array.
 - Offset gives the starting address of the array, and the index register value specifies the array element to be used.

The slide footer contains logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA, along with the number 19.

Moving on with indexed addressing mode. In the previous case, we have seen in relative addressing modes, the content of PC is added with the offset value. Now, here either a special-purpose register or a general-purpose register is used as index register.

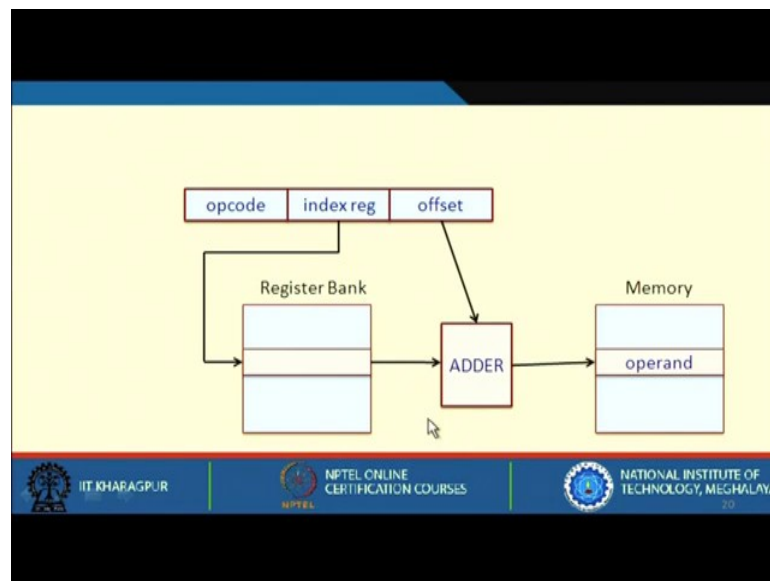
In this addressing mode, we can access an array. Array is a set of consecutive memory location. So, if you load a particular address, you know the first address of an array, how will you go to the next address, next address, and so on. In a similar fashion here in index addressing mode, you add that general-purpose register value it can be the used as index

register and this instruction specifies an offset or displacement that is added to the index register to get the effective address of the operand.

So, let us see with this example. Now, see 1050(R3); that means, content of R3 will be added with 1050, and then that location will give the operand. So, 1050 is added with R3, we get a value that value from which address in memory we get the operand, and it can be used to sequentially access the elements of an array.

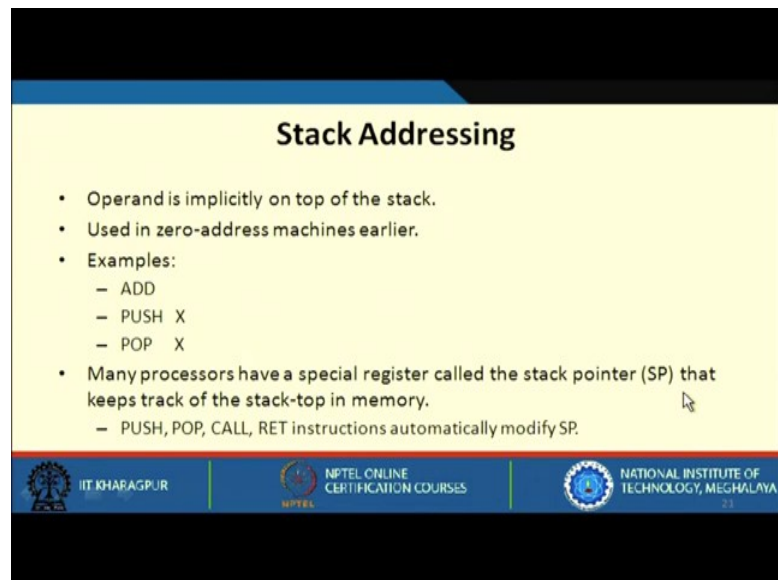
So, we load the first address and then we move to the next, next, next address by adding an offset to it. So, offset gives the starting address of the array and the index register value specifies the array element to be used; the first can be 0th element, then the next, then next and so on.

(Refer Slide Time: 34:11)



So, here this index register you get this added with this offset, this particular address is searched and we get the operand from there.

(Refer Slide Time: 34:23)



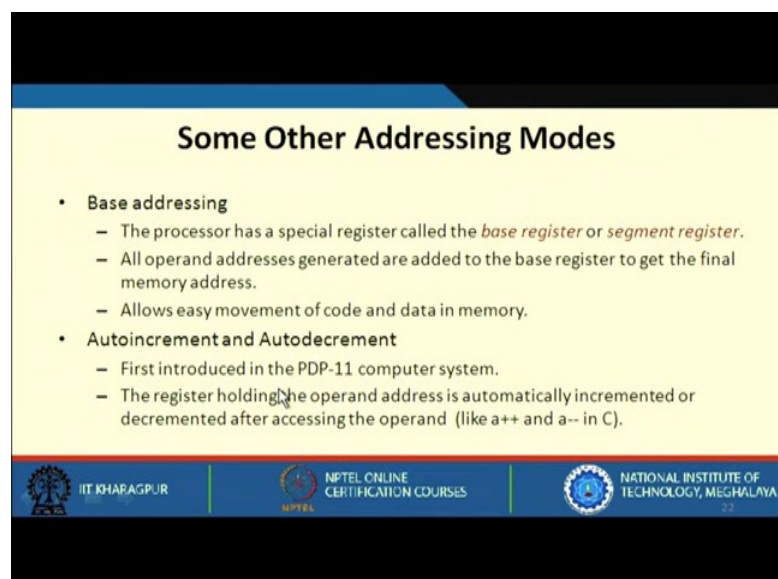
Stack Addressing

- Operand is implicitly on top of the stack.
- Used in zero-address machines earlier.
- Examples:
 - ADD
 - PUSH X
 - POP X
- Many processors have a special register called the stack pointer (SP) that keeps track of the stack-top in memory.
 - PUSH, POP, CALL, RET instructions automatically modify SP.

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

Next, come to stack addressing. In stack addressing, we already know the operand is implicitly on the top of the stack and it is used in zero-address machines much earlier. The first two elements on top of the stack will be taken out, will be added, and stored back there. PUSH X will push the X value into top of the stack. POP X will take out the top value to this location and store in X. Many processors have a special register called a stack pointer that keeps track of the top of the stack.

(Refer Slide Time: 35:14)



Some Other Addressing Modes

- **Base addressing**
 - The processor has a special register called the *base register* or *segment register*.
 - All operand addresses generated are added to the base register to get the final memory address.
 - Allows easy movement of code and data in memory.
- **Autoincrement and Autodecrement**
 - First introduced in the PDP-11 computer system.
 - The register holding the operand address is automatically incremented or decremented after accessing the operand (like `a++` and `a--` in C).

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

There are some other addressing modes as well, like base addressing mode. So, in base addressing mode, the processor has a special register called base register or segment register and then what happens here is all operand addresses generated are added to the base register to get the final memory address. Let us say the processor generates the address from 0, 1, 2, 3 and then you have stored some address in base register let say 1024, so 1024 will be added with that particular address. So, this is what base addressing means and it allows easy movement of code and data in memory.

We can also have another addressing mode, autoincrement and autodecrement addressing mode. It was first introduced in a PDP-11 computer, which was one of the most popular minicomputers in the 1980s. Autoincrement and autodecrement means that if you load a register with some address you can auto increment it you can access that value then you increment it, or auto decrement means you access the value then you decrement it. So, either way you can implement this. So, auto increment, auto decrement we have also seen in `a++` and `a--`, auto decrement and auto increment operators. So, in the similar way we can have such kind of addressing modes also.

So, we come to the end of lecture 7. What we have seen in this lecture is that addressing modes which are very important and the instruction format. We will move on in the next lecture where we will see the types of architectures.

Thank you.