

Computer Organization and Architecture: A Pedagogical Aspect

Prof. Jatindra Kr. Deka

Dr. Santosh Biswas

Dr. Arnab Sarkar

Department of Computer Science and Engineering

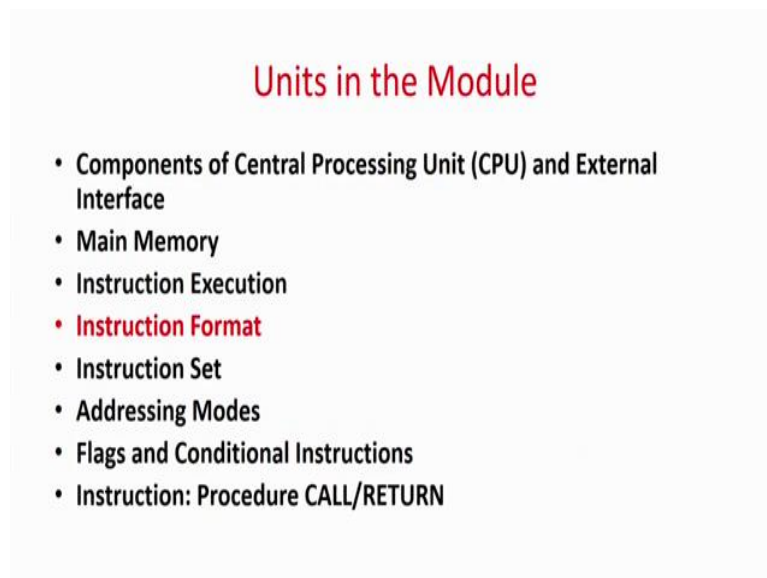
Indian Institute of Technology, Guwahati

Lecture - 10

Instruction Format

Welcome to the 4th lecture or that is the 4th unit of the module on addressing mode, instruction set and instruction execution flow.

(Refer Slide Time: 00:35)



So, in the last three units we have basically discussed what is the basically a CPU? What it consists of; and how it is interfaced with the main memory and then we have gone into the details of basically, what is the basic motive of this module is to understand how a basically an instruction executes in a CPU. So, in that in that direction, first we had seen in the last unit that how our instruction is basically executed.

That in a Von Neumann architecture we know that it is already the data and the code is in the main memory and then slowly one after another the instructions are fetched, decoded and executed. So, that is basically the instruction execution. So, up to that we have seen in the last unit.

So, now in today's unit we are going to focus on instruction format, because as I told you this module will mainly deliver or you will be able to understand, basically how to design an instruction given a set of requirements or a set of specification that is the main goal actually. So, for that we first now look in a more generic fashion that what is an instruction and what is the basic format? So, this is the unit four of this module.

(Refer Slide Time: 01:39)

The slide is titled "Unit Summary" in red. It contains a bulleted list of instruction elements and a paragraph about instruction format. Handwritten red annotations include a large circle around the first two bullet points, checkmarks next to the last two, and a bracket on the right side of the list.

Unit Summary

- The operation of a CPU is determined by the instruction it executes, referred to as machine instructions or computer instructions. The elements of an instruction are as follows:
- **Operation Code:** Specifies the operation to be performed (e.g., add, move etc.). The operation is specified by a binary code, known as the operation code or opcode.
- **Source operand reference:** The operation may involve one or more source operands; that is, operands that are inputs for the operation.
- **Result operand reference:** The operation may produce a result, which needs to be stored.
- **Next instruction reference:** This tells the CPU where to fetch the next instruction after the execution of this instruction is complete.

The instruction format is highly machine specific and it mainly dependent on the machine architecture. For example, if it is assumed that there is a 16-bit CPU. Let 4 bits be used to provide the operation code. So, we may have to 16 ($2^4 = 16$) different instructions. With each instruction, there are two operands. To specify each operand, 6 bits are used. It is possible to provide 64 ($2^6 = 64$) different operands for each operand reference.

And then as we told that the course is being delivered in a pedagogical perspective. So, first let us see what is the unit summary. So, the in the in this unit basically you will be studying the generic format of an instruction. So, as an instruction as we have discussed in the last unit that basically a instruction executes or does the general operations in a computer. So, if there is an operation to be done. So, we require basically two things one is, what operation I have to do? And, basically on what operands you have to do the operation.

So, basically opcode and source and result operands these are the two very important things. So, the most fundamental thing even if it's a non-computer perspective. So, even if I ask you that you have to operate add two numbers. So, this is one of the very basic instruction we do in the low level school days.

So, this is basically an instruction. So, what is the format of an instruction? So, if I order you something. So, I have to tell you, what to do and also I have to tell you on what objects you have to do the operation and where you have to store the result. So, that is actually called the opcode; that is, what operation you have to do?

Like for example, you have to move an operand, you have to add two numbers etcetera, etcetera. And then you have to tell that what are the operands? That is the in case of a computer the operands are actually some immediate operations some immediate values which are specified in the instruction or in the more broad terms for the time being you can think that the values of the operands, that is the variables and the values of the variables are stored in some memory if it's a Von Neumann architecture.

So, a source operand reference; that is, where the value of the operand is stored in the memory, that is; the second part which is basically has to be there in the instruction. Then of course, I do some operation, now what I do with the result in that has to be stored in somewhere stored in some memory whether it may be a register it can be a memory location etcetera.

So, that is the result operand reference that is; where you have to store the result. So, these things are basically of from a layman language as for as well as our computer prospective these three stuff should be there in an instruction.

But, when you are thinking over computer prospective or a code prospective, then after one instruction you have to execute another instruction. So, of course, you have to also tell in that instruction that which is the next instruction to be fetched. So, the reference of the next instruction that whether, it is the next immediate instruction or whether it is a jump instruction whether it will go to some other forward referencing or it may loop back.

So, that reference also should be there so. In fact, if you talk of an instruction you have basically, what to do? On, what to do where you have to store the result and what to do next whether this is last instruction or whether the next immediate instruction has to be taken or then to some condition you have to go some other places to jump in the memory.

So, that is the basic format of an instruction again as I told you one very important as I, now we will see little bit like how many? What How do you decide the length of an instruction? So, of course, you have opcode. So, it is represented in binary. So, if I said that the opcode is 3 bits. So, how many operations are possible 2^3 , 8 operations are possible.

So, if your specification says that 8 operations are fine like you can have load, store, add, subtract, multiply then more or less I am very happy with 3 bit opcode, but if I have lot more to do like I want to add immediate I want to add two numbers from a register, then, if I can have multiply I can have subtract I can have divide and it is what not.

So, in that case the numbers of instructions are much more. So, in that case the number of bits in the opcode will be larger. So, as I given in the example in the summary that depends on the bit size of the opcode of the reference you can decide how many instructions or how many different type of operations are supported.

(Refer Slide Time: 05:17)

Unit Summary

- Most of the arithmetic and logic operations are either unary or binary.
 - Need a maximum of two addresses to reference operands.
 - The result of an operation must be stored, suggesting a third address.
 - Finally after completion of an instruction, the next instruction must be fetched, and its address is needed.

An instruction may require to contain four address references: two operands, one result, and the address of the next instruction. Most instructions have one, two or three operands addresses, with the address of the next instruction being implicit (obtained from the program counter).

- It is difficult to deal with binary representation of machine instructions. Thus, it has become common practice to use a symbolic representation of machine instructions. Opcodes are represented by abbreviations, called mnemonics that indicate the operations.

– ADD	Add
– SUB	Subtract

Handwritten diagram illustrating binary representation and mnemonic: 110 0011 0011, ADD R3, 3

Then, basically as I told you many times in the last 2, 3 units that basically an instruction is divided basically into three types like: mathematical, arithmetic operation. Then you can have some load store operation and there is read write and there is some logical operation that is jump on 0 jump not on 0 etcetera; and one more thing.

So, there are basically that therefore, actually the next part means of basically; if these things are more or less of basic prerogative of an instruction that these are the basic stuff required like opcode, source, destination and what next instruction and basically three categories of instruction like arithmetic, logic etcetera.

So, if you take a logical memory operation. So, sorry in arithmetic operation, we generally have two operands it can be add, multiply, subtract. And generally we take two sometimes unary operations unary operands also can be there like for example, this is the number you want to negate it.

So, one operand is also possible, but there cannot be any 0 operand instruction, that is very obvious and again before you go to the main stuff. Actually, as I told you that all these

instructions basically are represented in binary like for add there should be an opcode and there may be the representation of can be 101, sub the opcode representation may be 111, but if you write a instruction like say 110.

Then may be 0011 and then 0011 something like that. Then, what it means? It will mean add and this may correspond to the third register and this may correspond to 3 memory location number 3. So, it will say add whatever value of the variable stored in third memory location to the register number three very difficult to understand.

So, we generally write in a mnemonic fashion add R3 to 03 sorry 3 hex. So, it was it will say that ADD 33 hex. So, what does it mean you take the value of memory location 3 in hex and then add it to register number 3 and give to it will be addition and write back.

So, this way instructions are represented will which we can read very nicely and easily. So, therefore, with all these mnemonics these are actually these abbreviations are called basically mnemonic way of representation like instead of add instead of 101, we will write add sub we will not write the binary version. So, from now onwards throughout this course whenever we will talk about the instructions we have to understand inherently that they are all represented in binary in a physical computer, but in this case for ease of representation we do it in a abbreviation form which are mnemonics. So, that is what is the summary of the chapter of this unit.

(Refer Slide Time: 07:52)

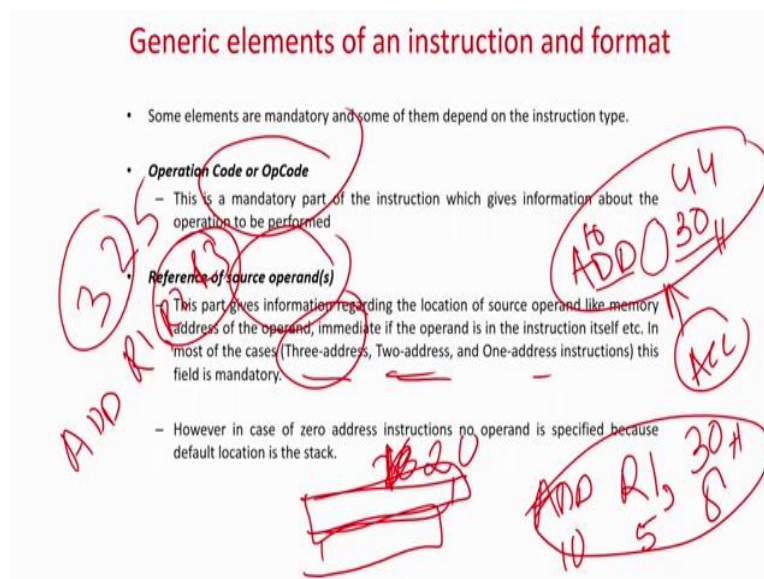
Unit Objectives

- **Knowledge: Describe:**--Describe the different elements of a Machine Instruction and some possible formats for instruction decoding.
- **Application: Illustrate:**--Illustrate the use of different instruction formats: Three-address instructions, Two-address instructions, One-address instructions and zero-address instructions.
- **Knowledge: Identify:**--Identify the different components involved, like operations, data, etc., while designing an instruction.

So, what are the objectives? That is going to we are going to fulfil after doing this unit basically we will be available to describe you will develop knowledge. So, recall type of an section in objective which will say the describe the different elements of a machine instruction and some possible formats; that is, how basically an instruction looks, then you will be able to illustrate very important instruction formats which were developed in the pedagogy of computer science or in the history of computers; that is three address instruction, two address, one address and even 0 address instruction very interesting that we will have many operand there, but how it will operate. So, 0, 1, 2, 3 basically these are the formats opcode is single.

Because, opcode will tell you what to do and the operands can be 0, 1, 2, 3 the 0 is very interesting and it will not have 7, 8, 9 instruction I means 9 operands, why? Because, otherwise the instruction will be very long it will be very difficult to store in the memory and then as a knowledge you will be tell the different you will able to identify the different type of component involved like given an instruction with which section represents the operation, which section represents the data, where the data is located etcetera.

(Refer Slide Time: 08:51)



So, let us go to in details of the unit. So, basically the generic elements of an instruction and its format so, there are some of the very mandatory features or the mandatory part of an instruction; that is, the opcode as we already discussed unless you tell what to do nobody can say that this is an instruction.

So, every instruction will minimum have an opcode; that is minimum and this is generally represented in binary. So, if you have 1000 instructions. So, if you have 1000 different tasks to do to. So, you should have 10 bits for the opcode. So, that is very simple. So, based on the number of instructions required you pick a log and that will be size of the opcode, then very important is the source of operands like where.

So, it can be 3, 2, 1 ; that means, if I say add say 30 hex ; that means, whatever is the value of the instruction is says that add, take the memory location 30 hex take the value, but add where. So, in this case if nothing is mentioned is a de facto standard then it's a accumulator. So, whatever value in the accumulator as I already this as accumulator is a special type of register the value of the whatever value we will be having in memory location 30 hex will be added to the value which is already stored in accumulator and it will be added back to this. So, if I do not write anything in this place. So, it will de facto means that it is accumulator.

So, this is a single word sorry single one address or a single operand type of an instruction. Now, why this is very good? Because, the size of the instructions are small because you can say ADD 30 and that maybe if you have some 1000 instructions. So, this one will take 10 bits and this is 4 + 4, 16 bits. So, your memory word length will be 16 bits, but say for example, two address.

So, in this case I can have something like add may be say a register one and then can be 30 hex now you see as defined. So, I will have 10 bits for that, because you require 1000 plus instruction types, register may be if you have 32 registers. So, all registers will have different binary values.

So, it will be taking 5 bits 32 registers should be 2^5 bits 32, 5 bits will be taken and should be again 32 bits. Now we can see $18 + 5$ so, it is 23. Now, 23 bits are required. So, if you assume that your memory word length is say 16 bit. So, in this case what will happen?

This instruction if there is a two address instruction will not fit into a single memory word. So, it will become a double word memory the double word instruction, that is; one part will be here and then will be other part will be here. So, you will format it in such a way. So, that it becomes 32 bits that we can alignment can be taken care of, but the main goal is to say that now it will become a double word instruction.

So, first when we fetch the instruction you have to take two memory locations at a time at a time you cannot do. So, you first fetch a part of the memory, that is; first location then you fetch the second part join them in the instruction register may be there can be can be two instruction registers the width of the instruction registers has to be increased two instruction registers you put in parallel jointly and then you decode it and do it.

So, you we will understand that it is more cumbersome to do it rather than if you have add and I said that de facto standard is accumulator in this case that is $8 + 10, 18$. So, in this case also it's a bit difficult. So, it may go ahead, but let us assume that for the time being if I have a 20 size memory, then you can easily see that add 30 hex will easily fit in one word. So, here we are taking all hypothetical examples, but to just illustrate the concept that if you have a large long instruction then basically the problem is that you will require multiple words to store in the memory, and when you fetch and decode there are more number of steps involved and the hardware is also complex, because if I say that the width of the memory is 20 or the word size is the 20 bits; then this instruction will fit $10 + 4 + 4, 18$.

So, just a single instruction will be fetched single instruction will be fetched, decoded and executed, but if you take a double two address instruction. So, in this case it is 23. So, it will be one memory location plus another. So, generally these instructions are formatted in such a way that either it takes one word or it takes two word not generally one and half this is just to give an example.

That, in now it will take more than one word. So, instruction and decoding and all those things will be much more cumbersome similarly with the three address of course, three address means you can have say something like that I can say `ADD R1, R2, R3`. So, in this case the value of *R1* will be added to *R2* and will be stored at *R1*. So in fact, what happens that I can add three numbers together.

So, it may have the value of 3 it may have the value of 2 it may have the value of value 5. So, I require $3 + 2 + 5$, I want to do you can write `add R1, R2, R3`; that is, the value of *R3* will be added we will be added to *R2* will be added to *R1* and everything will be stored back in *R1*, basically that is the instruction format or how the instruction happens. In this case if it is the single if it is two address, then basically you can take only *R1, R2*, that is; first we will be adding $3 + 2$, then we will be storing in some temporary register like *R1* and then again the new value we have to add to the existing.

So, there will be multiple numbers of steps so let us look at it in a more nice fashion. So, that the so, disadvantages already we have seen that if you take very long long instructions, then the memory words will be more and then the problem arise will be you have to have two memory location has to be fetched for an instruction, then decoder it will be slower and more hardware complex, but what the advantage is larger the instruction you can do same operation with less number of instructions like three address say that I have to add 3, 2 and 5. So, they are in three different registers for the time being let us assume.

(Refer Slide Time: 14:39)

Generic elements of an instruction and format

- Some elements are mandatory and some of them depend on the instruction type.
- **Operation Code or OpCode**
 - This is a mandatory part of the instruction which gives information about the operation to be performed
- **Reference of source operand(s)**
 - This part gives information regarding the location of source operand like memory address of the operand, immediate if the operand is in the instruction itself etc. In most of the cases (Three-address, Two-address, and One-address instructions) this field is mandatory.
 - However in case of zero address instructions no operand is specified because default location is the stack.

3 2 5
ADD R1, R2
ADD R1, R3

So, we can write `ADD R1, R2, R3`. So, very simple value of `R3` will be added to `R2` will be added to `R1` and everything will be stored back to `R1`. So, one instruction will do the purpose, but if it is a two word instruction; then you are gone. So, in fact, what will happen it will it will not be able to do it in 2.

So, this is the case. So, `R1` and `R2` so, first `3 + 2` will be added and it will be stored in `R1`. So, next instruction you have to write `ADD R2; sorry R1, R3`. So, first stage what will happen `3 + 2` will be added, because in `R1` there is `3` `R2` there is `2` they will be added and the value will be stored in `R1`. Next instruction will be `ADD R1, R3`. `R3` it is `5`. So, now, it will `5, 5, 10` and it is stored.

So, you require two instructions to solve the problem. So, your code will become larger. So, therefore, basically this is a trade off, but if you look at the current trend people have all gone

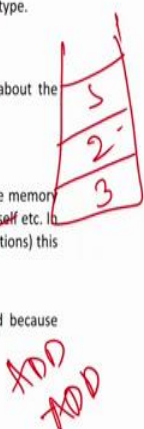
for shorter instruction length because our computers have nowadays become more and faster than the number of executing one after another instruction is quite faster.

So, people have gone in this direction that if the instruction smaller less width execute more number of instructions per cycle, because your CPU processors are much faster rather than making these instructions very complex and taking multiple words in the memory that is what is the trend and very interestingly we study about what is the 0 address instruction, there is no operand specified.

(Refer Slide Time: 16:17)

Generic elements of an instruction and format

- Some elements are mandatory and some of them depend on the instruction type.
- **Operation Code or OpCode**
 - This is a mandatory part of the instruction which gives information about the operation to be performed
- **Reference of source operand(s)**
 - This part gives information regarding the location of source operand like memory address of the operand, immediate if the operand is in the instruction itself etc. In most of the cases (Three-address, Two-address, and One-address instructions) this field is mandatory.
 - However in case of zero address instructions no operand is specified because default location is the stack.



Like if I say add then where are the operands. So, whenever you say that I am doing with a zero word instruction or zero address instruction so. In fact, there is a stack involved with it. So, in that stack there will be different elements like for example, there are three there are these two. So, and then you place the 5.

So, in case what will happen you have to write ADD and ADD. So, what is going to happen in this case. So, whenever you say add or any instruction you give it will take the first two elements depending if is the two address or three address.

Basically, but in fact, in case of zero address instruction basically why it happens is that we default there is a stack attached to it and whatever operation you attach like add if you say add it will take the first two elements on the top the stack. It will be popped up added and the value

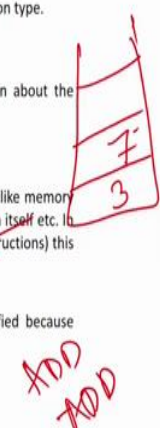
will be pushed to this stack may be if you say negate is a single bit instruction is single operand instruction.

So, the first value of the stack will be popped up made 5 and push it back. So, if I write add. So, what will happen 5 and 2 will be popped up and then what is it will be added up. So, it will be first it will be popped up.

(Refer Slide Time: 17:14)

Generic elements of an instruction and format

- Some elements are mandatory and some of them depend on the instruction type.
- **Operation Code or OpCode**
 - This is a mandatory part of the instruction which gives information about the operation to be performed
- **Reference of source operand(s)**
 - This part gives information regarding the location of source operand like memory address of the operand, immediate if the operand is in the instruction itself etc. In most of the cases (Three-address, Two-address, and One-address instructions) this field is mandatory.
 - However in case of zero address instructions no operand is specified because default location is the stack.



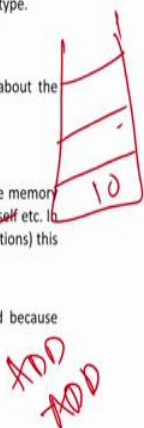
So, 5 and 2 will be popped up 7, the value of 7 will be written over here.

Next if I say another add the first two will be taken and they will be popped up and the value of 10 will push back.

(Refer Slide Time: 17:20)

Generic elements of an instruction and format

- Some elements are mandatory and some of them depend on the instruction type.
- Operation Code or OpCode**
 - This is a mandatory part of the instruction which gives information about the operation to be performed
- Reference of source operand(s)**
 - This part gives information regarding the location of source operand like memory address of the operand, immediate if the operand is in the instruction itself etc. In most of the cases (Three-address, Two-address, and One-address instructions) this field is mandatory.
 - However in case of zero address instructions no operand is specified because default location is the stack.



So, whenever I told that zero address means is nothing to be surprised; that means, in the instruction itself you are not saying where are the operands, but there is a different stack attached to it and it will start operating on the elements of this stack my first pushing means popping them up and doing the operation and push back.

So, if it's the unary operation like negate all those things single will be popped add means two will be popped. Generally, we have never heard that three values are popped and the operations has been done that architecture was never popular ok.


(Refer Slide Time: 17:48)

Generic elements of an instruction and format

- Example of a three address instruction: ADD R1, 3030, 3031**
This instruction adds values stored in memory locations 3030 and 3031 and stores the sum in R1.

OpCode	=	ADD
Destination	=	R1 (Register)
Source 1	=	3030 memory location of first operand
Source 2	=	3031 memory location of second operand
- Example of one address instruction: ADD 3030**
This instruction adds value stored in memory location 3030 to the value present in Accumulator and stores back the sum in the Accumulator.

OpCode	=	ADD
Destination	=	Accumulator (default and field absent in the instruction)
Source 1	=	3030 memory location of first operand



So, now we have told so, many theory. So, let us try to give some proper examples. So, it's an example of an instruction of a three instruction; three address instruction add *R1* 3030 hex and 3031 hex. So, what did it say the opcode is ADD. So, as I told you we will be always using mnemonics will be never writing the binary value. Destination is the register *R1* if there are 32 registers all the registers themselves will also have some binary representation it can be all 00001, because if there are 32 number of registers.

So, the number of bits required will be 5 and register *R1* means it will be 00001, but for ease of representation we will always use a mnemonic format. So, three words three address. So, this is the location of the first operand this is the location of the second operand and as I told you slowly we will see that when we will be going more advanced into the instruction types. So, sometimes add it may tell that I have to add what is the location value here what is the value here and store it in *R1*.

Sometimes it may also mean that whatever value is present in 31, 30 you add them along with the whatever the value in *R1* and the result you store in *R1*. So, that we will tell on the time of instruction as I told you that many times you have 1000 plus instructions. So, how many instructions can be possible right very difficult to think of 1000 instruction add multiply subtract load jump etcetera, but you have to think that the number is not impractical.

Because add can be of several types in this case I can say that it is add 2. So, I can say that it is add 2, what does it mean? It means it will take the variables or the operands from the last two or the last two operands and it will add to *R1* and I can also say there can be another format *R3* add 3 what it will do?.

It will take the value present in 31 30 as well as an *R1* and together the value will be stored in *R1* just like if I say that add single address instruction add 3030 hex. So, what does it mean it means whatever the value is present in 30 add to the accumulator and write it back so that means, you can have the formats or the architecture possible in which case also the register will be involved in the operation rather than just load and store. So, this one will make your instruction more effective and short. So, what I wanted to say is that add can be of several types even something can be like add immediate, likewise; I can say that add immediate what does it mean add immediate 30 hex. So, what does it mean? Add immediate means this 30 is now not a memory location this is basically the immediate value of 30 in hex.

So, whatever in the accumulator will be added to 30, and it will be written back to this written back to the accumulator, immediate means the value of the operand is specified in the instruction itself. So, in other words add can be of you know different types 20 types multiplication can be of so much variation, store can be of so much variation, jumps can also have so many variations.

That is why the number of instruction even if the basic operands operations like add multiply subtract store can be very few, but the variations are huge in number. So, therefore, 1000 different instructions or 500 different instructions or 200, different instruction is not a impractical number they are very much practical. So, anyway that is not the concentration of the or consideration or the not the method of I mean not the point of focus here. Here we are just trying to see what are the different instruction format and basically how they are represented.

So, in this case we tell you that it's a 3 address instruction. So, these are the 3 address and this is the opcode means where it is stored where the values etcetera are we will be coming later when we will be going into more depth of instruction, how to design an instruction? What are the different types of instructions etcetera?

Now, the instructions set design etcetera we will be taking up in the next three units, then all those things will come that what is the add what is add immediate etcetera, then this is actually example of a single address instruction as I told you. So, one address instruction means basically there is nothing mentioned over here this is basically the accumulator that mean already the accumulator is de facto given over here.

So, whatever the value will be represent in 3030 memory location will be added to the value already stored in the accumulator and you have to store back in this one. So, these are the basic instruction I have shown you as an example ok.

In fact, if you if you consider two or instruction. So, you can have add R1 3030. So, in that case it's a 2 address instruction. So, one will be the memory location on will be in the R1 and you can store it back in the R1.