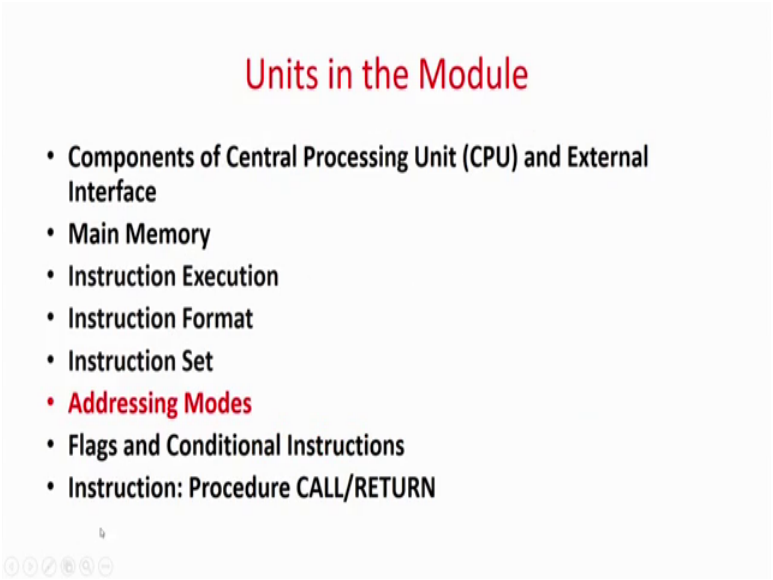
**Computer Organization and Architecture: A Pedagogical Aspect**

**Prof. Jatindra Kr. Deka**
**Dr. Santosh Biswas**
**Dr. Arnab Sarkar**
**Department of Computer Science & Engineering**
**Indian Institute of Technology, Guwahati**

**Lecture – 12**
**Addressing Modes**

Welcome to the next unit on the module on Addressing Mode instruction Set, and Execution Flow.

(Refer Slide Time: 00:34)



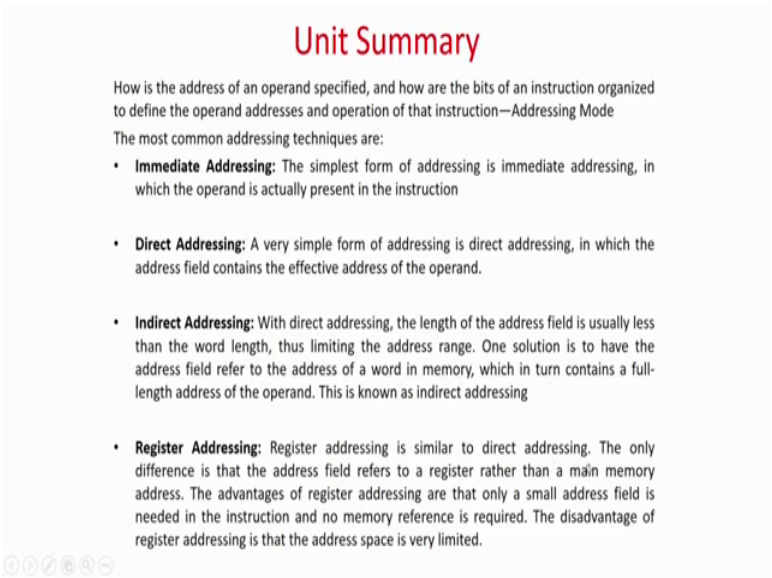

So, till now we have covered 5 units of the module in which we are discussing basically the basic architecture of the central processing units, how it is interface with a memory? Then how what are the basic instructions are and how a instructions is executed. Then we have also seen different format of instructions and then we have seen how instructions basically can be clustered together based on their formats.

Now, today we are going to focus on the next unit which is actually very important in this part which is called the addressing modes. That is, in other words or in an Indian language what happens that all instructions as we have discussed in the previous units they require and op-code that which tells what it has to do and of course, we require the operands.

So, how the operands are present in the instructions will be actually refer to as the addressing modes. Like for example, some instruction may have the data to be operated on in the instruction itself like add 5. Then if you can call it as a immediate instruction mode, but sometimes you may have the instructions with the data is deeply embedded in a very indirect manner. Like for example, the address of the instruct of the data may be available in the instruction which is something called a memory indirect that is there are different spectrum in which case in which the data is present in an instruction.

So, in this unit we are going to discuss in detail what are the different addressing modes?

(Refer Slide Time: 01:57)




So, as we are a dealing the whole course in a very pedagogical manner. So we will use it is summary we will talk about define a instruction in modes. So, the first one which will have we are going to see is the immediate addressing.

So, immediate addressing means the instruction itself will have the data. Then, there is something called direct addressing means the instruction will not have the operand itself, but it will be pointing to a memory address where the data will be present. Already we have seen in the fewer previous units that if you want to keep very accurate or a data whose a accuracy is high it requires a large number of bits to be represented.

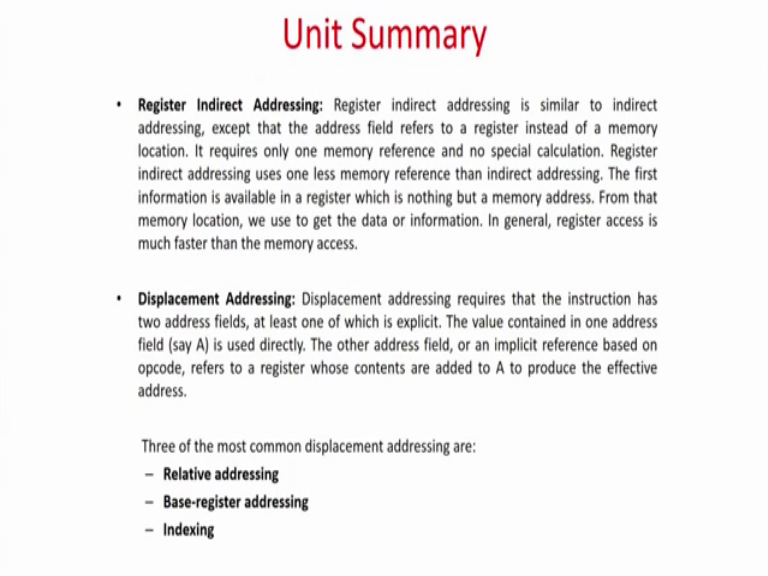
So, if you want to do it is as a immediate format then the instruction size will become larger, to avoid that we go for direct addressing that we give the address of the memory where the data is present and in fact, the data can be accessed from there.

Then there is something called indirect addressing. The details or advantages and disadvantage we will see as will go along the units. So, what is an indirect addressing? In this case the data is not present in the address which is these mentioned in the instruction is a indirect way of referring that is the one how reference. The instruction will have been reference to a memory and indirect memory location there will be another address which will be pointing to the exact data in the memory. We will let us see that if allows you to expand the memory space or this size of the data to be represented.

Then there is something called we will see there is something called register addressing. So, register addressing is very similar to direct addressing, but in that case the only difference is the address refers to a register than a main memory. In this case there are as I told you there are different variations in where the data is present.

So, in register addressing what we do instead of saying that the memory is or the data is locating in so and so memory location we will tell the data is present in such and such register. So, as the number of registers are small in number. So, the data or the space in the instruction required for such type of addressing will be lower as well as the access to data will be faster because you need not go to the memory you can directly get it from some other register. But again, the trade off is the number of registers are small so you cannot have a large number of data present in this addressing mode.

(Refer Slide Time: 04:11)




We have seen a direct indirect then you are seen register similarly there can be register indirect. So, register indirect is very similar to normally in direct, but in this case the address refers to a register in a than a memory location.

So, what happens in indirect in indirect addressing? In the instruction you tell where this is the memory location where the data is present, but indirect means you tell the memory location in that memory location again there will be a redirection to the exact data. But in case of register indirect you are not going to effort to any memory location whether you are going to use a register to do the indirection.

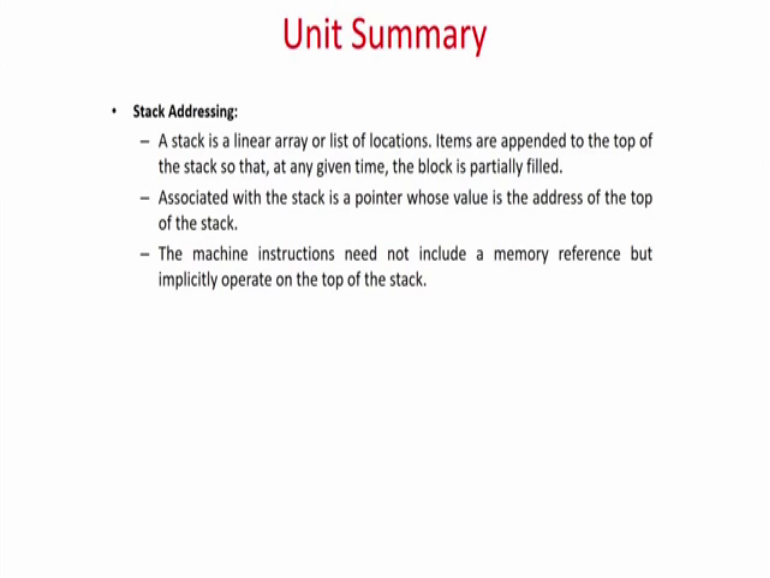
Another next very important is something called a displacement addressing. This is slightly different from all other three we have studied. In displacement addressing basically there are two parts of the address: one part is exactly fixed which is mentioned in the register instruction and there is another part actually with says to a whose contains are basically can be modified is some similarly something like that say may be we have a op-code and two parts of the address one parts is fixed and the other parts actually can be modified or it can be in fact they are added and you can play on with this two variable addressing part so that you can go for a wide spectrum of addressing like looping or you can go for a displacement then you starts from 0 then you put a buffer or you put a displacement to that. Then you can move some 10 x starts from 10 x plus the kind addressing.

So in fact, you can understand in a we will see in details when you are going into the module the displacement means the two components of an address there added to find out the effective address and one of them is modified so that or you can go as for example, you want to increment a loops. So, one part of the address can be changed or incremented is added to the other. So, you can get lot of displacement you can go in a loop any in that manner.

So, displacement addressing is slightly different from all the others we have discussed till now. So, all others are basically kind of a static way and here actually it is something like which is the updated so you can go for a shift you can go for a loop etcetera.

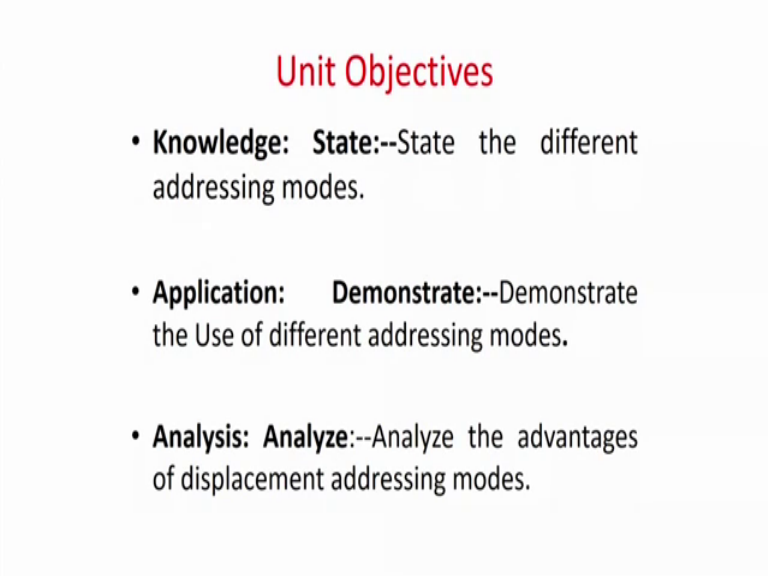
And the most common displacement addressing are relative addressing base register addressing and index register addressing, so that is we will see. Means based on the register in which you will be trying to do all this displacement the names have been given.

(Refer Slide Time: 06:25)




And then finally, we will see stack addressing. So, it is very simple we have already discussed in the previous unit in case of stack addressing the op-code or the instruction will not have anything basically rather than the only the op-code and all the data will be presenting in a stack the top two elements of the stack one elements of the stack will be used to do the operation.

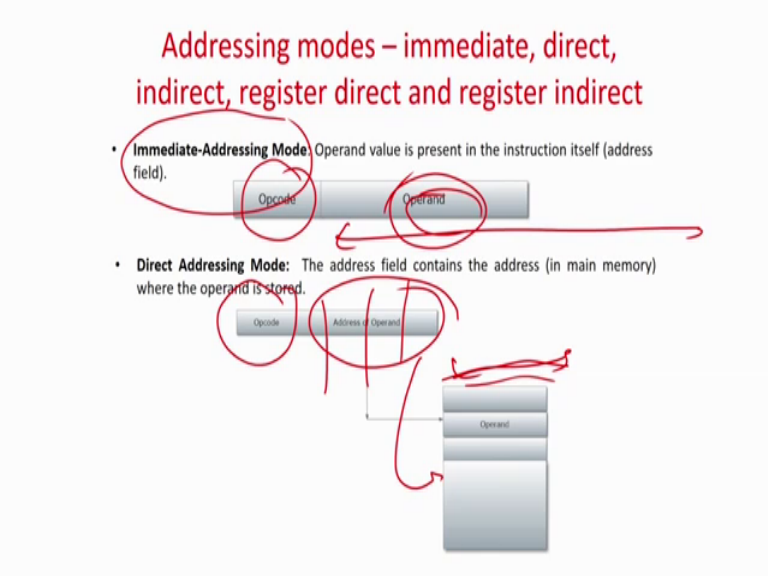
(Refer Slide Time: 06:41)




And then what are the objective of this unit where the objective of this unit is first knowledge you will be able to state different type of addressing modes. Then next is the application objective we will be able to demonstrate the use of different addressing modes when it is better which one as the trade off which is faster which is slower which takes a larger space in the memory and so forth. And then finally, analyze then you will be analyze the advantages of different addressing mode in particular the displacement addressing mode.

Because all other addressing mode as I was telling is some point of a static either the data is in the instruction or the instruction has the memory reference where the data is there. But in case of the displacement you add you increment and all those some kind of dynamism is present. So, this one I will be you will be able to analyze that what are the advantages of such a kind of a addressing mode.

(Refer Slide Time: 07:36)




Now, this was about the basic objective of the modules, the summary of the modules. Now we are going to look start the unit and we are going to look at real concrete examples and more in a more elaborate a pictorial representation what are these type of addressing modes are?

So, the first one as I was telling you was the immediate addressing mode, immediate addressing mode means in the same instruction you will have the op-code as well as the data will present over here.

So, whenever this instruction will be fetched from the memory you need not have to fetch any others staff or any other memory location or any other register to get the operand the operand is present in the memory location itself. Then the next most, but the problem here is that as I told you if your data is to be more precise that you require more than width to representation and this is not going to a very handy kind of a addressing mode because the size of the instruction is has to be then a 2 words, 3 words, which is not actually very fast to be implemented.

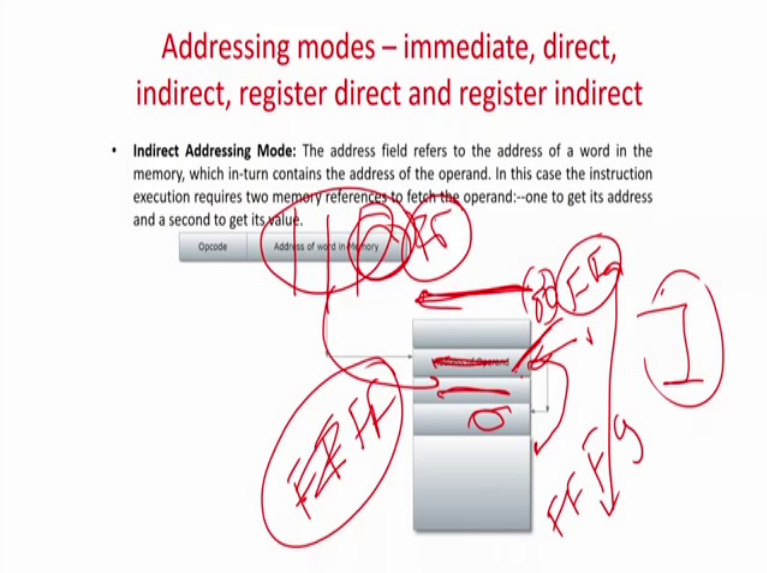
But what is another way of doing it? Then what you can do then you can say this is called the direct mode on addressing in which is the op-code and here you will refer to a memory location and we know that this can be quite this is this why this is the width of the memory location. So, you have to just tell that which memory location the data will be present. So, this is direct way of addressing. So, if you say op-code something add 32

it means the data is available in the 32th memory location. But in this case there is a memory operation required.

But in fact, as you can see; obviously, you can have a more wider space to do your operation. Because in a general instruction format there will be an op-code there is not only a 1 operand, there can be multiple operands. So, I mean you have to divide this space into multiple places.

So, you will have a very less space for each operand, but if you refer to a memory location then it will be just the address of the memory location and the data can be quite wide quite wide or you can have a more precise data in that manner. So, this is called the direct mode of addressing.

(Refer Slide Time: 09:24)




Then there is something called indirect mode of addressing. So, indirect mode of addressing is very similar to direct mode in direct modes what we have seen that this one will refer to a memory location where the data will be present, but in this case what happens the data will not be present over here either it will refer to another memory location where the data will be present.

So, in this case to get a data you will have to have two memory addressing. So, first you have to address the first location which you tell you the location of the data which is available in the; that means, the data is available actually in this memory location.

So, first one you will have first the instruction will tell you which memory location to address which has the address the memory the memory address of the data is not directly present in the instruction itself which was the case in the direct addressing.

In this case it will tell the address which will refer to a memory address which will exactly which will absolutely have the address of the location of the data in this and not here. So, this is actually the address and not here.

So, there are 2 memory addresses required to get a data, but what is the advantage as I told you generally in op-code there is a number of operand present in any instruction. So, this is not quite large.

So, what happens actually so in fact, I mean if the memory size is quite large. So, you may not be able to put the full a full bandwidths of the, or I should not put full range of the memory cannot be encoded in this small space. For example, if you require FF, so if the memory has 0 0 to FFFF number of locations. So, 4 into 4 so it is FFFF then actually you require 16 bits to represent the memory location.

But in fact, if you understand that it is divided into 4 number of operands. So, you may not have the space to represent a 16 bit operand over here that is the memory address of this. So, in this case we can restrict that my data path will be a smaller part of the memory maybe I can say that F 0 0, FF will be limited to the range of the data.

So, in this case the 0 0 will be expletively gave is it implicit and the only FF has to be kept over here. So, I am actually making the size of the op-code size of the size of the operand means address of the operand in the memory smaller then you will apply FF 0 0 will be explicit. So, you can go over there, but that is one way of doing it. The other way of doing is the in that case now you have you we will refer to this one.

Now, now in, but you need to actually address the full memory for that because memory the data can be anywhere in the memory. So, it may refer to this one, but the exact, but to access the full memory you required to access the 16 bit that is 4 4 4 4 the 16 bit address space of the memory. Now where it will be present? The exact location of the data will be I mean that may data may be available in say FFF 9. So, data may be present over here.

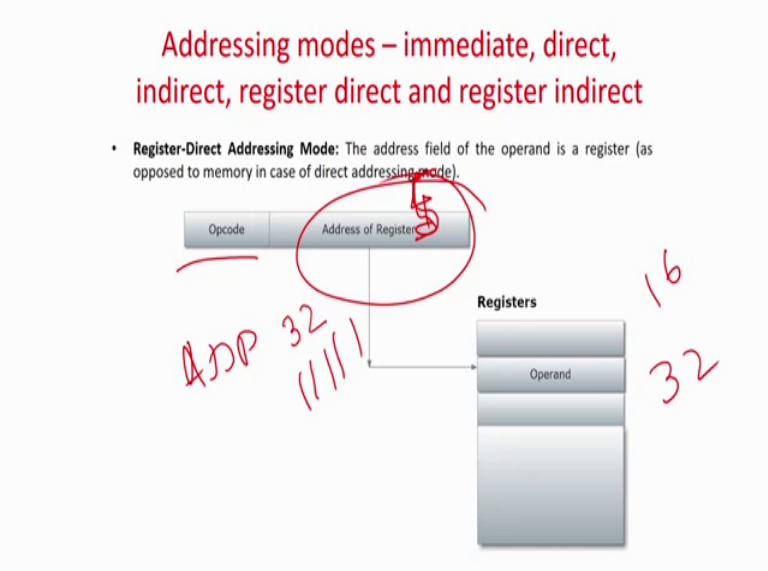
So, what it will do? So you are referring only FF because the exact location that is FFF 9 is actually present over here so this one is having the value of FF f 9 with exactly the data is present.

And in fact, that FFFF 9 we will actually take 16 bits to be represent that much space I do not want to keep in the instruction, but in the instruction I give the only say value FF that means, it refers to the first I mean the 0 0 is explicit. So, you just refer to this memory location and in this memory location I have the because this is much much is quite wide compare to this smaller space in the memory.

So, now can you can access the hold range of the memory itself so that is one advantage of the indirect addressing mode. There are several others like we will see then actually we require a indirect addressing mode for implementing or accessing arrays and all those things.

But as we are I have not gone into that much of details in the course as of now so for time being these understanding is enough that to get a more wides more full range of the memory access such an addressing mode is useful.

(Refer Slide Time: 13:23)




Then next we are going to this was all about the direct indirect and immediate addressing mode. But there is another way of doing is that is the instead of referring to the memory locations the data can also be present in the registers because register access is more

faster it is inside the CPU itself and then other advantages are there regarding speed etcetera. But again this is what is the disadvantage the disadvantage is the number of registers are small are in number.

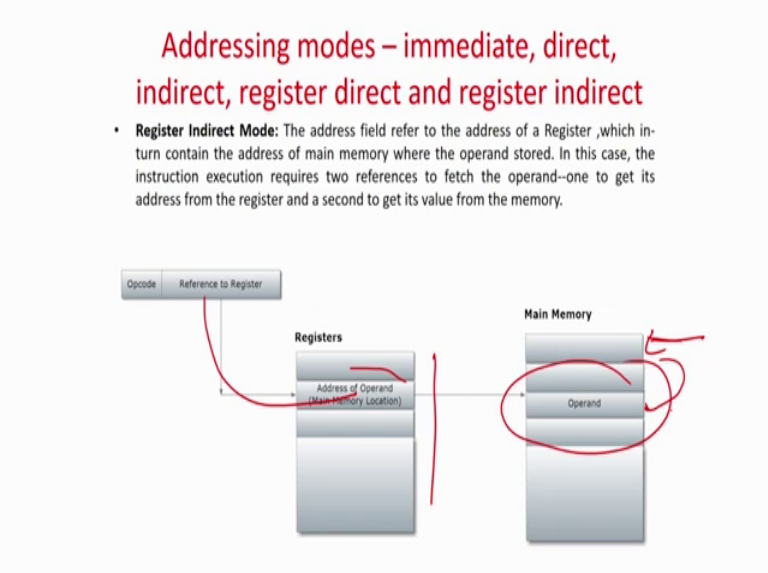
So, if you if you have you have more number of data that cannot be accommodate in a register, but it can be accommodate in a memory, but for certain access like you want to store some intermediate results where we want what the data which has to be fetched more frequently it can be temporarily stored in registers. So, what is the register a direct addressing mode a op-code will be there and there will be address of the register.

In this case actually instead of referring to the memory you are going to (Refer Time: 14:10) register. In fact, registers are the if there is the 32 numbers of registers or in fact if there is 16 number of registers then you require just a 4 bit representation to be present any of the 4 register, 16 registers.

If there are 32 registers you just require a 5 bits for the address of the register. Say for example, if you write some add and then you write say a 32 that means what that is 2 to the power 5 that is 1 1 1 1 ok. Then in that case you will be referring to the 32th register that is last register if you are having a thirty two number of registers, so you can see that the size of the address size of the instruction will be smaller and because the number of registers are less in number.

So, register direct means it is a just like a direct addressing mode the op-code will have the address of the data, but the data is present in a register and not in a memory.

(Refer Slide Time: 15:02)




Then same thing also applies for register indirect just like a indirect addressing mode this is also indirect register addressing mode like for example, in the direct addressing of register. So, the op-code refers to a register and you are data is present over the register in this register, but in this case is a indirect one. So, your op-code is there you refer to a register and the register actually refers to a part in the main memory where the data is present. So, this is register indirect.

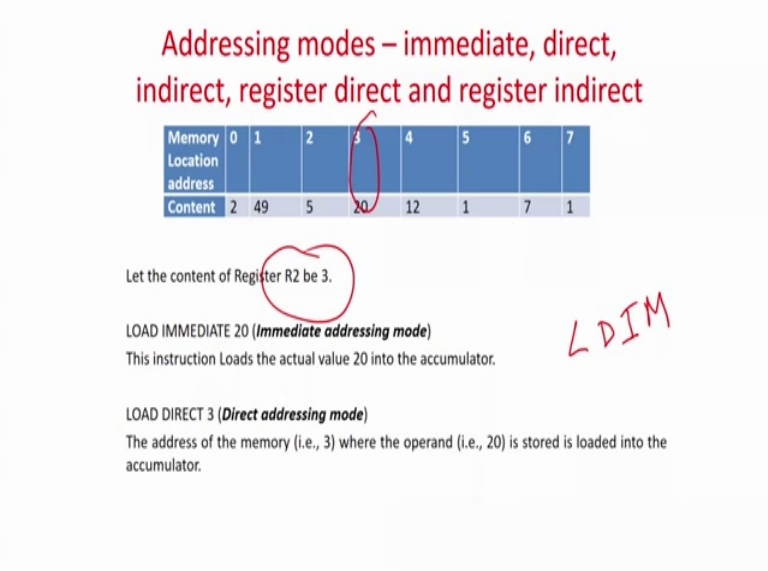
What was the normal indirect mode? In the or the generally indirect mode generally indirect mode means from the op-code you will actually point into one memory location which contains the location of another memory location which actually has the data. In case of register indirect you are giving the address of a register the data is not in the register, but the register is pointing to another memory location where the data is present and the advantages I told you that is before going for indirect addressing mode the you can access the full range of the memory because number of registers are in small, so if you compare to the general indirect general indirect addressing and register indirect addressing.

Generally indirect addressing means you will first refer to one memory location here and in from that you are going to be redirected here. So, you required two general through main memory access.

But in case of a register indirect the first you refer to a register and then you refer to a memory. So, in this case this is one register access and one main memory access. So, this is the more faster way of getting the operand because register accesses are more faster compared to a memory access. So, in this case there is a one memory access and one register access.

So, it is more faster than your normal indirect mode where there is two memory access so that way there is advantage and disadvantage. Disadvantage is may registers are less in number. So, you cannot use it very often while memory main memory is larger in larger in size. So, you can always you can have more frequent such type of indirect addressing mode ok.

(Refer Slide Time: 16:55)




So, now let us go to some concrete examples that is let us take a very simple memory which has location from 0 to 7 and the content are 2, 49, 5, 20, 12 likes this way. And let us assume that you have a register whose a contents is 3.

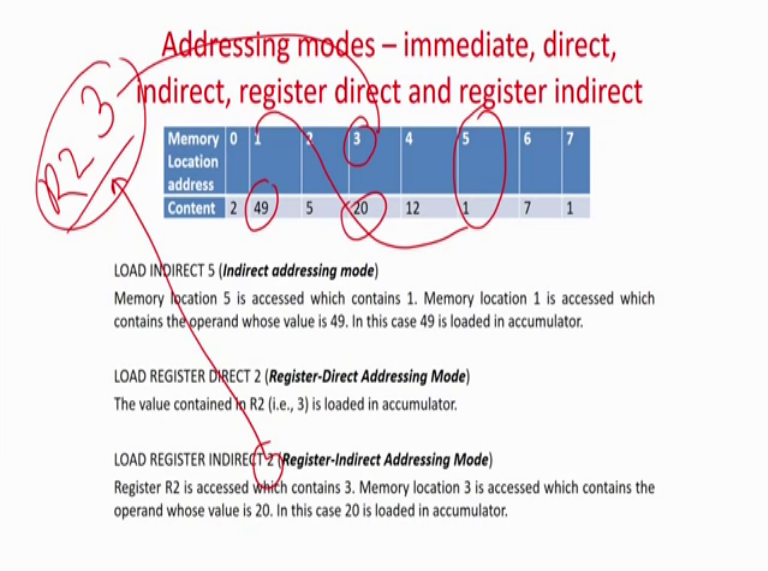
So, we will let us that instruction load immediate 20 that is may be the op-code can be something like LDI. Generally here I have discussed in the more linguistic manner, but we always have a pneumonic for that so it may be LDI that is load immediate or you can put some time person may call it load indirect also. So, I you can also have the word load immediate is up to you how you define your instruction set. So, this is load immediate 20.

So, what does it happen as an immediate? So, in that case the value of twenty binary twenty will be in the instruction itself in immediate mode of instruction you need not refer to any of the memory location neither you have to refer to any other register other than the accumulator. So, it is says load immediate 20.

So, in this case what happens the value of 20 that is 20 binary will be loaded into the accumulator load direct 30 is a direct addressing mode. So, what is a direct addressing mode? Direct addressing modes means this 30 actually refer to the memory location 3 where the data is present in the memory location 3 so, direct.

So, load accumulator 3; that means, to the accumulator you load the data which is present in memory location 3 that is 20. So, the 20 will be stored in the accumulator one thing you remember register R 2 has the value of 3; R 2 has the value of 3 that you have to remember.

(Refer Slide Time: 18:30)




Then load indirect 5. Now this is interesting load indirect 5 means accumulator will be loaded with something which is present in memory location 5 memory location 5 has value 1 now is an indirect. So, one is not the data the data is actually present in memory location one.

So, what is memory location one present; that is 49. So, the 49 will be stored in you are accumulator that is load indirect 5 5 1 is the content this is nothing this is not the data or

the operand basically it is the address of the memory where actually the operand is present of the data is present that is 49.

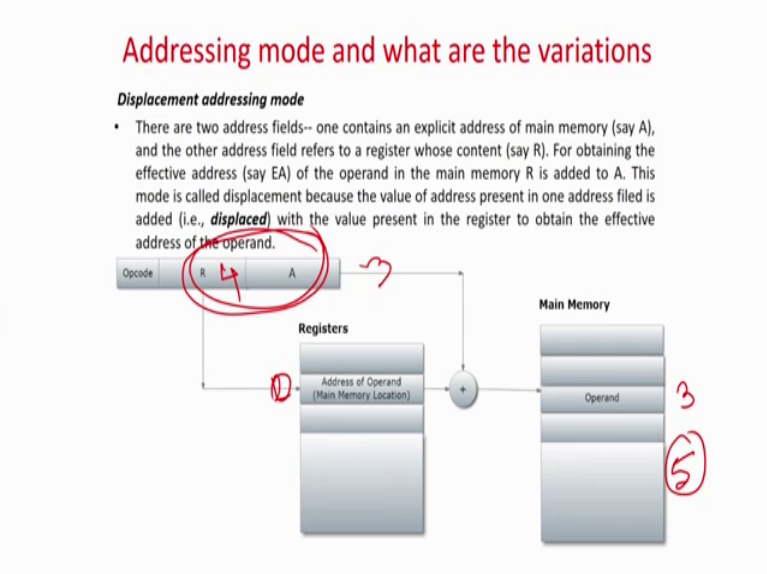
Then load register direct 2. So, in this case as I told you what is says is a direct it is a direct instruction, but instead of referring to a memory it is referring to register it is saying a load register direct 2 that is you load to the accumulator what is available in register number 2.

So, as I told you our assumption was register 2 has the value 3. So, it is loaded into the register next is as this normally indirect so this is called register indirect 2.

So, register indirect 2 means so register 2 has the value of 3. So, now is the indirect mode of addressing. So, 3 is not the data basically you have to access memory location 3 because 3 is present over register 2 and the content is 20 so 20 will be loaded over here same thing like load indirect.

But in case of register indirect we first refer to register to register 2 was the value 3; 3 is not the data over here we have to look at the memory location 3 is there where the content of 20 is there the 20 will go to the accumulator. So, this is an example of register indirect addressing.

(Refer Slide Time: 20:08)




Then as I told you so all these where somehow would I called you is something like a static kind of an addressing. So, you refer there then you go for indirection and you get

the value. But wherever you talk about displacing displacement basically you will find the there are two components of addresses which is the use together to get the effective one.

So, what is the displacement mode it says it has 2 address field one contains an explicit address of main memory say that is a either some explicit value and the other address field refer to a register whose content something say R it refers to some memory R that is one of this component refers to the memory another R correspondence to a register though just two components basically. One is for the memory and one is for the register for obtaining the exact value that where the operand will be present the these two are actually added.

So, for example, if it says the memory location 3 and the value of this may be 2 for example, in the register then what is actually happening say it is referring to register 4 and register 4 has the value 2. Then this 2 and 3 will be added together you are going to get the value 5 and 5 is the place where you actually the data will be present.

So, there are lot of advantages of this means compared to the other one there are lot of applicability of this one first is if you think of loops then actually it will go say I want to access 10 memory locations one after the other than.

So, it will first start with say I can make this content as 0 then it will start with 3 this memory location is 3. So, it was start with 3 then I increase one then it will go for 4, 5, 6, 7, 8, 9, 10. So, I can just change the value of the register by 1 1 1 1 and you can continuously access first location second location third location and so forth.

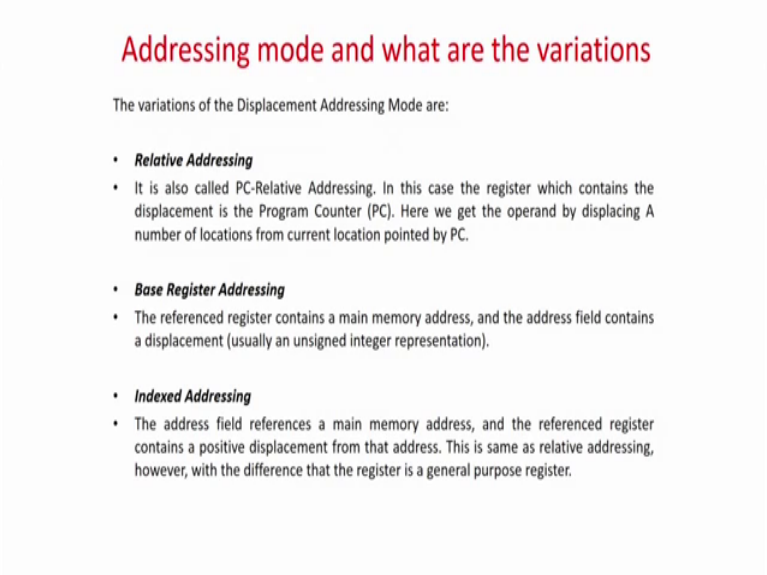
Similarly for array if the datas are present in the contrivers location in the memory. So, you can access different elements of an array by such kind of a displacement mode. There is something called relocatable loader in that case what happens say you load the whole code say for example, for program counter starting from 100 then the program counter jump because of a jump instruction.

So, in this case as will see later you can change the value of the register and you can give the value of say initially my loop program count was 10, then you jump to 20. So, you make this location counter as say 10 or something like that so there will be a jump of plastic.

So, such type of lot of case is where some dynamism is required where you require the displacement of the addressing like jump increment by 1 in case of loop such kind of addressings are required.

But in this case you have to know that differs from all other said that you add the two values of the component that one is the memory location content another is some register content and together you will get the value and you can modify with the register content by 1 5 whatever one you want as replacement displacement there will be added and the effective address will be calculated. Say there are different variations of displacement addressing.

(Refer Slide Time: 22:54)




In fact, the variations all starts based on what register you are accessing for this. The second part is more or less constant but these constants in this says that you refers to a memory location in the main memory, but what register you are using it for and what is the purpose we will define the different types of displacement addressing right first is the relative addressing. So, it is also called program counter relative addressing.

In this case the register actually contains the PC say if this one if the register addressing is the relative addressing. So, is actually the where the displacement is present is the program counter. So, in generally in this place so if you see here we get the operand by displacing a number of locations from the current point location pointed by the PC that means, in this case the affective location is obtain by the location of the PC, content of

the PC, that is if where the PC is pointing you added to A and then you get the displacement where you have to start.

Generally jump means instructions and several other similar kinds of things are addressed or when you want to a go for such type of that means, such mode of operations you require like there is a jump instruction etcetera then you can use this type of addressing mode.

Because in this case the a program counter actually has the displacement value then you can know different values and the jump will actually happen based on that. There is something called base register addressing. So, instead of the program counter if the referenced register contains a main memory address and the address field contains a displacement.

So, if the name of the register if I call it has the base register instead of the program counter if is a base register we call it is a base register addressing mode and if the address field as if the you have 3 type of registers basically.

So, one is actually called the relative addressing if the register is a program counter if it is a base register then actually on the displacement will call a base register addressing mode and index register addressing mode basically is now a general purpose register it is same as relative addressing but with the different register that is a general purpose register.

So, instead of base register or instance of program counter if you are using R 1, R 2, R 3, R 4 which is a general purpose register we will call it as a indexed addressing. If indexed register addressing is generally use for loops like for example, I equal to j I equal to 1 to 10 then do something.

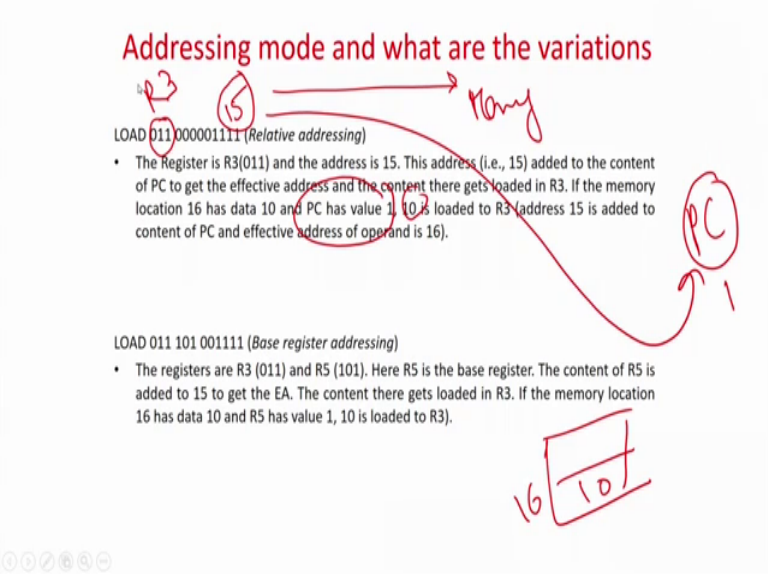
So, that if I refer I to a register then if you will actually I called as the indexed register addressing. Because I will map I to a general purpose register like an array if there is a array and I index if is j, so zeroth element, 1 element, 2 element, third element and so forth. So, I call it as the index of the array.

So, that index of the array means where which will be incremented and I want to address first memory location second memory location, third memory location, and so forth corresponding to that array.

So, that index of the array is basically map to a general purpose register whose value I will be incrementing by 1 and then from which memory location the array starts that will be the explicit dot of the addressing like this one. So, A means it will be the starting point of the array and this will be your index register which wills have the initial value 0 then you go for 1, 2, 3, 4, 5, 6 then you access the entire array.

So, if this register is general purpose then we call it index, if is the base register then this base register displacement addressing, if is the program counter we call it is a relative addressing the function may be it is more or less similar that is there is a memory location and you have to add address 1, 2, 3, 4, 5, 6, or you have to jump from memory location x to x plus 5. So, this type of addressing is displacement addressing is used. But based on the address or based on the register which is used further purpose there different types.

(Refer Slide Time: 26:41)




So, again it is better lot of thing we have discuss. So, it is better to take some concrete examples or this addressing mode that is displacement addressing mode to make the things clear. So, first I am taking a relative addressing mode relative addressing mode is

means that the program counter is directly involved. So, you are writing load 011 and then this is the value.

So, as you are all assuming is that these an accumulator default instruction. So, it says that the register R 3 because this is the register R 3, and the address is 15 so; that means, whatever is the content in R 3 the address 15 that is this is your address in to the main memory this is location to the memory then is added to the content of the program counter So, base register.

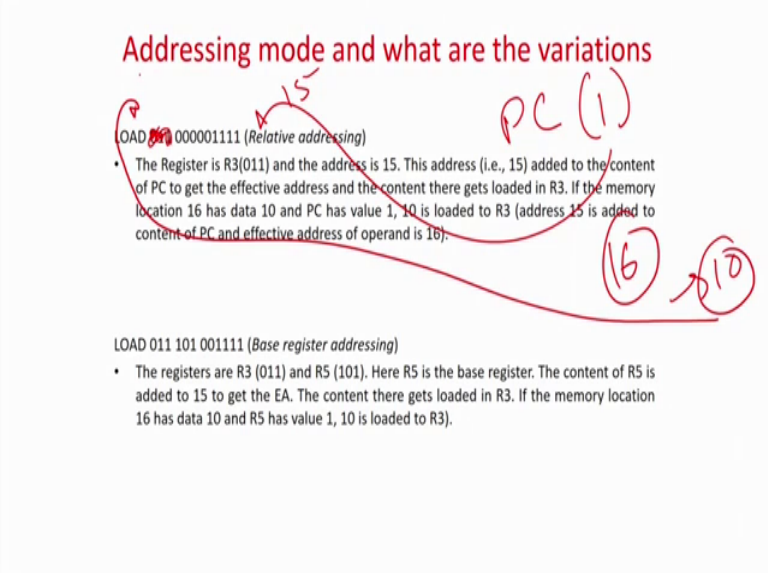
So, the whatever will be the is an immediate addressing relative addressing mode. So, PC is default one of the register and in this case what it is saying the register R 3 that is register R 3 is the register the content of this is 15 will be added this 15 will be added to the content of the PC whatever the PC will be have the value and that contain that is 15th memory location does the contents of PC will that data will be taken and loaded into the register number 3 if the memory location has. So, in this example there assuming that the program counter is 1.

So, if the program counter is then 1 then 15 plus 1 will be the 16 and whatever is present in the 16 memory location will be added will be loaded to register R 3. So, what happens 15 is the memory location main memory location program counter is 1 15 plus 1 is 16; 16 memory location whatever will be the data.

So, if we says PC has the value 1, 10 is loaded in the accumulator sorry 10 is accumulated register 3 because there assuming at the memory location 16 address 16 has the value of 10. So, program counter was one this explicit addressing was pointing at five. So, add it and load the value of 10 that is the operand to register R 3. So, if it is the accumulator type of a addressing then what we can do let us make it the where is it will be more simpler. So, it can be something like we can say write, load we can remove this.

So, in accumulator based addressing. So, in this case it will say load that is load relative. So, relative addressing load 15.

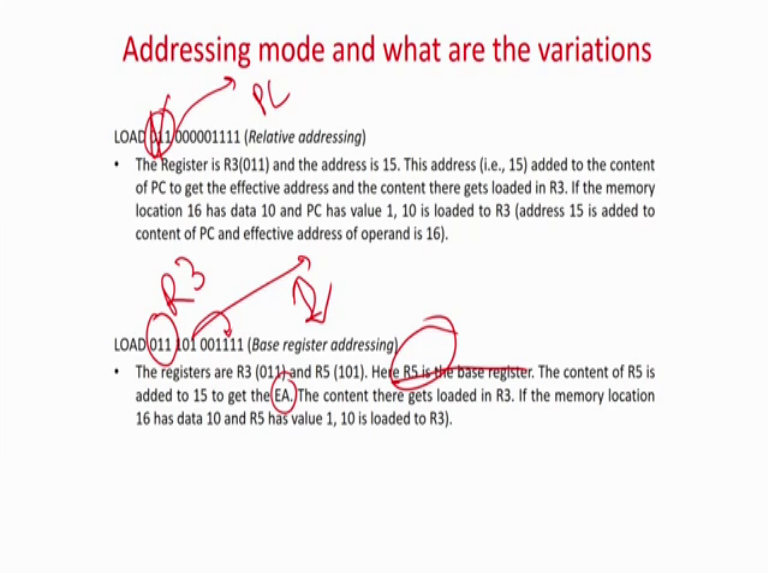
(Refer Slide Time: 29:12)




So, what it will mean that whatever is in the program counter content in this case is assuming to be one you add it with fifteen then memory location is becoming 16 16 memory location is having the value of 10 this 10 you load it to the accumulator, but in this case that will go to accumulator, but in this example they are not taking the example of an accumulator direct machine we are taking the value of R 3. So, the content will be loaded to the accumulator sorry content will be loaded to register R 3.

So, whenever is a register R 3 or whether it is an we loaded to the accumulator that does not depend on the that does not have anything to do with the addressing mode the addressing mode basically in this case is the displacement addressing modes take the value of value of PC add to 15 and load at some place in this case it is register through if I drop this an assuming it to be an accumulator based accumulator based instruction. So, it will load it to the accumulator. Now let us in this example we are not considering that it is a directly loading to the accumulator we are all considering the destination is resister R 3.

(Refer Slide Time: 30:08)




So, if you look at it so base register addressing. So, in this case as I told you that here in the last case we do I have not mentioned anything here it was default was the PC, but in this case as I told you instead of using a R 3 sorry instead of using a PC if I using a very special register in this case they are assuming that the R 5 is a base register.

So, that content will be added to 15 and you will get the affective value of the memory location where the operand is present. So, the content of R 5 is added to 15 to get the affective address and the content is loading into R 3. So, in this case if you assume that this just like program counter if these are register R 5 that is the base register is the value of 1.

So, it will be again 1 plus 15 into 16 10 is there in the 16th memory location it will go and load into R 3 say for example, but if I have the base register value of 2. So, it will be 15 plus 270. So, whatever will be available in the memory location 17 will be loaded to register R 3. So, only difference in between these two that one is a program counter and in this case there is a very special register which is a base register and you are doing the same functionality. But as I told you program counter is a very special group it always points to the present memory location to be executed after that it goes to the next instruction.

If there is a jump instruction it goes to the jump location. So, in that case this these type of instructions are extremely useful because there is something called means relative

loading. So, we always when you are say assume when we are say generating a code in the offline of this after compilation then you will assume that the codes starts from memory location 0.

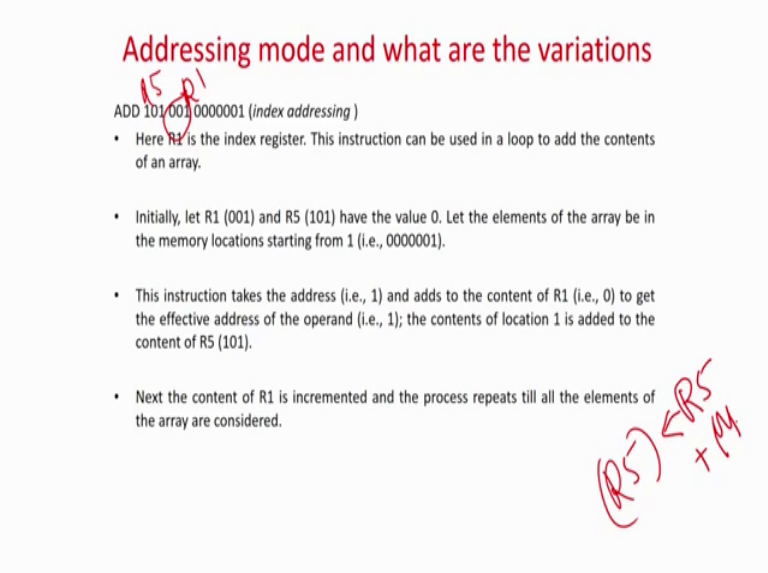
But due to non availability of the zeroth memory location to be loaded and executed it is sometimes get loaded at starting from 100 memory location. So, the program counter will be starting from 100.

So, in this case program counter value 0 will be added to 100 which is actually this value you can make it 100 because your code could load from memory location 100 and it could not load from memory location 0 because it was not available. So, you can all find the details of such type of concrete examples in the course on assembler linker and loader which is actually called relative addressing.

So, in this case we are always assume that the location of the means executions starts from 0 all the addresses of the assembly language code are assumed from 0. But then when has to be executed it may not start from zeroth location it may start we have a displacement. So, in this case this value we can give the displacement. So starting from 100 so you put 100 over there. So program counter will start from 0 so it will be 0 plus 100.

So, effectively whole thing is displaced below by 100 memory location and the execution happens. So, that is a very that is the used for this type of addressing when we are going for some kind of a relative addressing or relative loading kind of a thing. Base register means instead of the program counter the value of this can be or this program counter is not used further you have a base register in which can you can do some kind of a displacement.

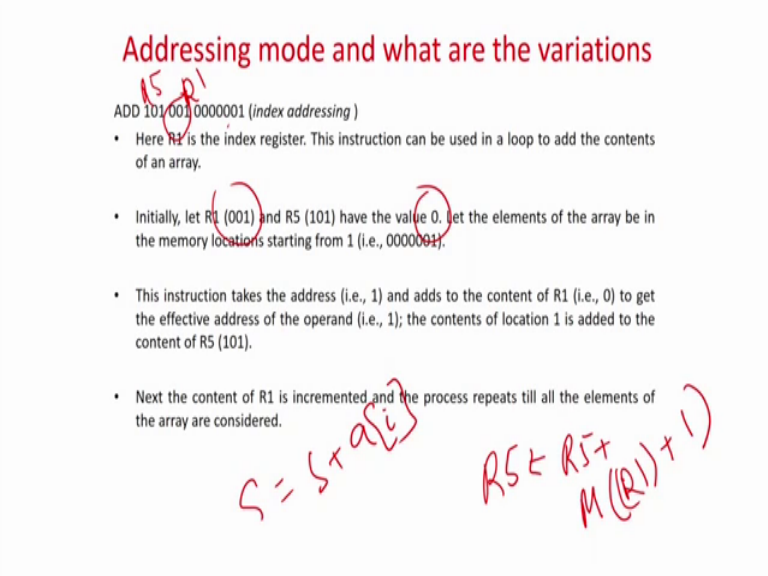
(Refer Slide Time: 33:07)




Then the last one is the index register as I told you index register that is this one sorry the index register this one is a general purpose register this is not the base register this is not a program counter you can have it can be R 1, R 5, R 10, R 12 which is a general purpose register. So, in this case the instruction is add 101 that is register 5 this is register 1 and some memory location is 1 that is this term. So, here R 1 is the index register it can be any general purpose register in this case what they are doing that using it as the loop. So, as I told you the one of the most important utility of index, index addressing is loops basically.

So, here R 1 is the index of some loop. So, it is saying that whatever available content of R 1 you add to 1 so that will be the effective memory location. In this case if the R 1 is 0 so the effective memory location is 0 plus 1. So, the first memory location the constant the contents of the first memory location will be added to whatever was the content at R 5 and will be stored back over here. So, R 5 is equal to R 5 plus content of R 1 that is the memory location. So, what is the memory location over here? So, let me just write it.

(Refer Slide Time: 34:30)




So, what is this and effectively trying to do? So, in this case a register R 5 will be R 5 plus what is the content of the memory location how can you find out is whatever is R 1 will be added to the content of the memory location how the memory location is calculated this content of R 1 plus 1.

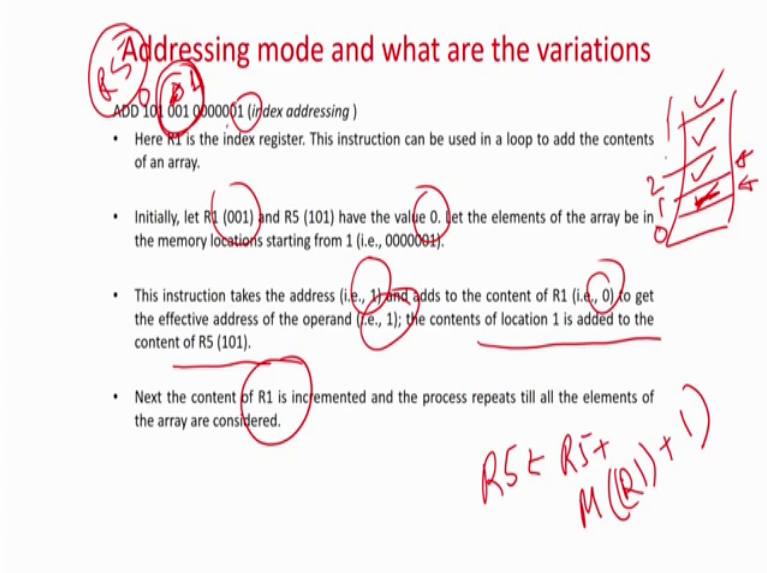
So, I I add value of 1 and I add what is the content of R 1 that will be then effective memory location I get the operands from there add to R 5 content and storing the R 5 itself. So, initially they are assuming that R 1 is having the value of 1 and R 5 as the value of 0 that is R 5 is reset.

So, if you something like s equal to s plus i. So, s is reset and the i is going to be implemented the and in fact what is I? So in fact, is something like s equal to s plus ai. So, a is the array and i is your instruction. So, R is basically nothing, but in this case your R 1 and s is nothing but in this case your R 5 ok.

So, now see they are saying that the initial content of R 1 is 1 and R 5 sorry R 1 is 1 R R 1 is 1 R sorry this R 1 is 1 and R 5 is 0 that is reset and this is one. So, initially the elements of the array may be starting from 1. So, array is the location of the array are starting from memory location 1, 2, 3, 4, 5, 6 and this initially has the value this initially has the value of 0 reset and R 1 has the value of 1.

So, the so what will happen the instructions takes the address 1 and adds to the content of R 1. So, the content of R 1 R 1 and R 2 both have 0. So initially both of them has 0 value so that is this one will be added to the content of R 1. So, R 1 is having a value 0 0 plus 1 is 1. So, effective address is 1 and it will address the first content of the memory location. So, the memory location is something like this.

(Refer Slide Time: 36:23)



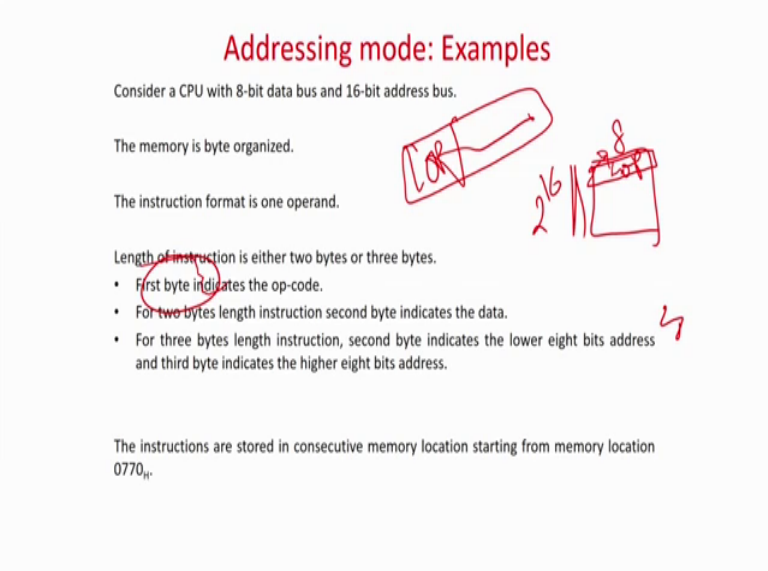

So, 0 that is garbage, 1 there is some data to there is some data and so forth. So, initially your both R 5 and R 1 are reset. So, generally it has the value of 0 these also is the value of 0, so 0 plus 1 is 1, so you are going to address this value.

So, whatever will be the content will be added with 0 that is the content of R 5 and it will be stored over here. Next next what you will do you will increment the value of the register number R 1. So, R 1 will be contents of one will be added to the content of R 5 here will be done.

Then next that is actually what I have told you whatever is present over here will be added to the contents of R 5 which is now 0, so it will be 0 plus the content of this one is nothing, but this one and it will be stored at R 5. Next the content of R 1 is incremented so this this one will now have the value of 1. So, 1 plus 1 will be 2. Now will be point will be this less memory location that will be again loaded added with the content of 5 as stored back.

So, this something happen in like s equal to s plus ai where a is an array and i is the index. So, this index is actually keeping on incrementing by 1, 2, 3, 4, 5, 6 and it is and that continuously first memory first array location, second array value, third array value, fourth array value you are getting and storing with them. So, that is why is a very simple example of an indirect an inside index addressing mode is the again a displacement addressing mode, but this in this index register is our own our own defined or user user available register which is R 1 in this case it in the other way in can be any user register which can be used by a programmer ok.

(Refer Slide Time: 37:58)




So, now basically now we are going to say some more examples which will give you a more in depth idea of how basically addressing mode happens we where the address where the values are located etcetera. Because in the most of the cases as I say at the instruction is basically op code and some operand or some addresses, but sometimes the size of the instruction cannot be such nice or such or the length of the or the width of the memory cannot be so good that it will whole the would be the whole the whole instruction in 1, while there can be in multiple points; that means, the op code will be one place and some part of the of operand will be there in the word and the and this case is exhausted.

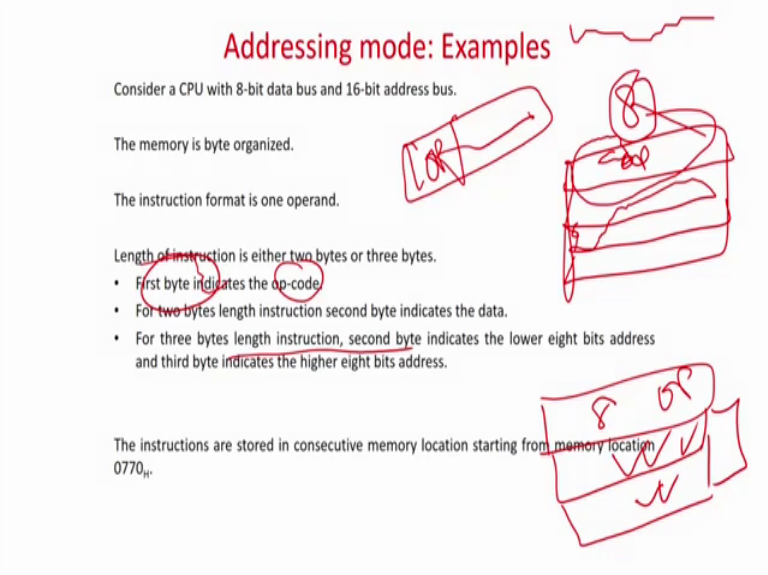
The remaining part of the instruction will go to the next memory location. So, taken the 2 memory location together you can have a instruction. So, that is called multiple word

instruction. So, now you have to go look at such complexities. So, we are considering a CPU with 8-bit data bus and 16-bit address bus. So, it is very simple it is something like this is 8 bits, and the whole space is 2 to the power 16, then then some other assumptions here we are taking.

So, what is the assumption? The first byte of the instruction is the op code. So, what do you mean by that? So, in this case as I told you the instruction has some specific characteristics the that is the op code the and then after that you can have a lot of other modes like immediate addressing you have the data value etcetera.

So, when I say there is a op code. So, when I say that basically this is your op code. So, some part of the instruction is deformed taken by this one. So, in this case I am say saying that the 8 bits. So, as I told you the instruction basically has a opcode and then some other places for operands. So, in this case I am assuming that the first byte is the instruction itself ; that means, if it is a 8 bit word.

(Refer Slide Time: 40:05)




So, I am saying that the opcode takes the whole part of it that is if because I want because you are trying to discuss in this part that is multiple word instructions that is this is your memory, this is your 8 bits, and in fact, this part op-code is taking the whole 8 bits together.

So, then what will happen then the operands will be present in the other parts because there is no space available to do that. So, for two byte instruction this taken by the instructions instruction that because in this case there cannot be any single word instruction sorry say single word instruction because one word actually in this case is 8 bits.

And if you look at it the whole 8 bits are taken by your op code. So, obviously, this 8 bits is the op code the next second bit can be some kind of an operand. So, for two byte instruction this is taken by the instruction data data or it can give if you can direct addressing mode. So, it will be reference of a memory register.

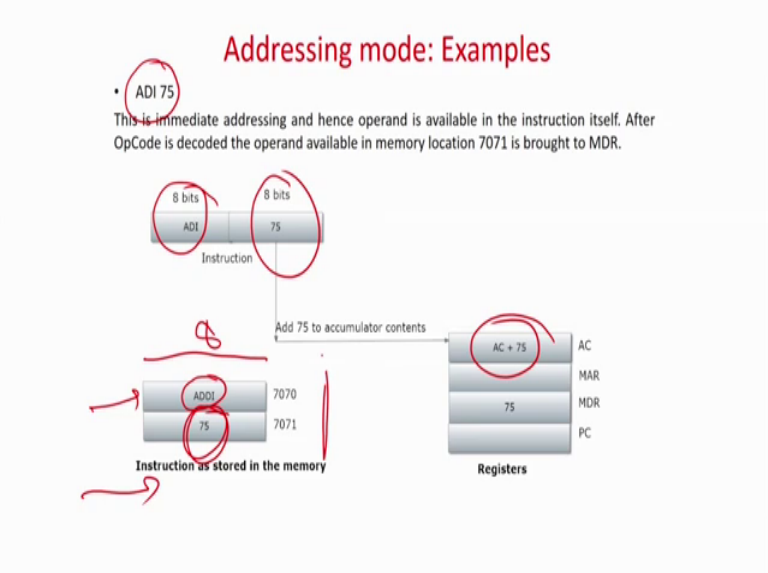
For 3 byte instruction the last 2 basically these 2 will be your data or it will be pointing out in some memory location and this one will be an op code. So in fact, what will be showing that if the in this case the memory is 8; 8 in length and the op means instruction side is larger because the op code is 8 bits and then other bits are reserved for operands or data.

So, you the whole instruction has to be spread out in the multiple memory location and in fact, this is what basically happens in all cases very very rarely we will find instructions which will fit into a single word then if it is a single word instruction then life is very easy.

First program counter 1, then program counter 2, then program counter 3, 1 instruction another another, but if there is multiple word instruction then first one if it is a 2 word instruction then you will jump to 3, then if it is a single word instruction then again be jump by 1.

So, the movement of PC is non regular and it depends on the size of the instruction. So, we are assuming that the memory instructions are in the for this present example we are assuming that we the instructions are stored in a memory location which is continuous and it starts from 0 7 7 0 hex. So, this is this is present pictorially.

(Refer Slide Time: 42:03)

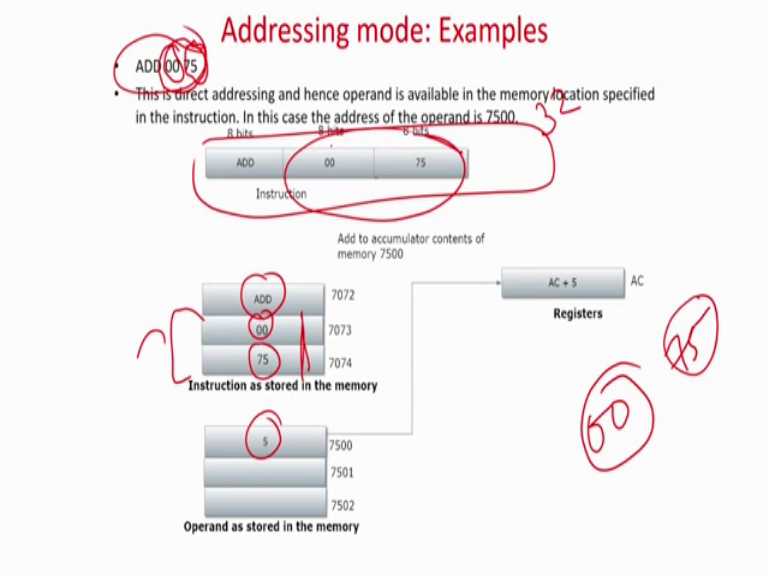



So, this is the with first instruction is add immediate seventy 5 so; that means, I told you if it is a 8 bit as well as your data is also assumed to be 8 bits. Now in this case I told you the whole memory size is only 8 bits. So, the instruction this whole instruction together cannot fit into a single word add immediate ADI that will fit in the first word that is 7070 hex and a next that is the your data as it is an immediate instruction it will be available in the just immediate address, immediate word.

So, in this case first your program counter will go over here and when the instruction is executed after that it has to go to the third memory location it cannot be the plus 1 it will be plus 2 because in this case it is a 2 word instruction it is a 3 word it will be plus 3.

So, what happens? So this is the case the data is seventy 5 add immediate list load the value of 75 sorry add the value of accumulator with 75 as stored back in the accumulator. So, this 75 is taken, it is added to the accumulator and it will be accumulator itself. So, this is the example of a immediate addressing mode and how it is stored in two words we have shown that.

(Refer Slide Time: 43:09)




Next is add is a direct that is it is saying that the this is your memory location. So, it is saying that add is the first drawn that is 8 bit this was taken then now in this case it is not an immediate addressing mode is a direct addressing mode.

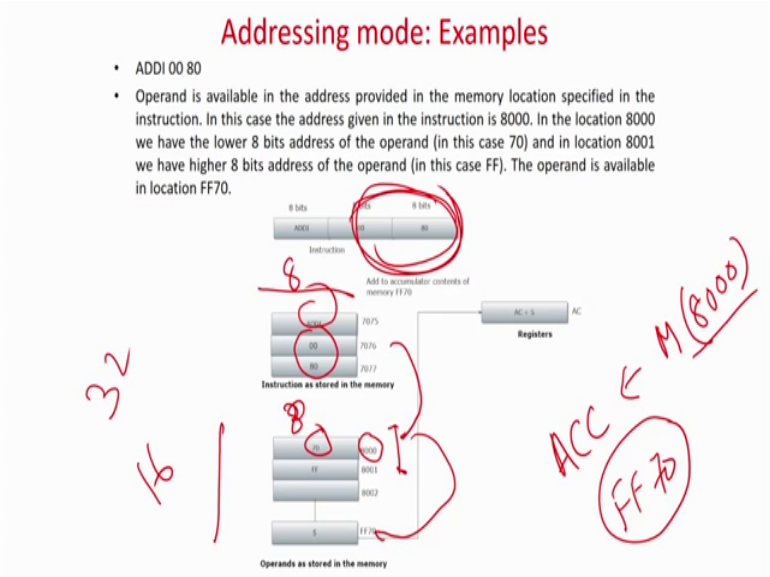
So, it is referring to a memory location memory location in this case is require a 16 bits that is hex first hex values, second hex is a third and fourth. So, now obviously, this 2 is 4 bits and 8 bits and these 2 is 8 bits 0 0 is 8 bits that is 000000007 means 01111 and 5 is 0101. So, 0 0 and 75 so this is taking 8 bits and this is taking 8 bits. So, this will again occupy one words space in the memory. So, this is the op-code then 0 0 is the lsb, and this one is the lsb of the address. So, it is stored over here.

So, now add direct so 0075 it is 7500 it will be addressing so these two will be together addressing the memory location 7500 hex, content is 5 and it will be added to the accumulator content as stored back to the accumulator. So, as I now we will easily appreciate the fact now as the data memory location everything is put over multiple words.

So, the hardware is becoming more complex because the in this case I have to add these contents of two different memory location side by side then I have to give it to the address bus then I can only give the value of 7500 and I can address that memory location I can get the content 5.

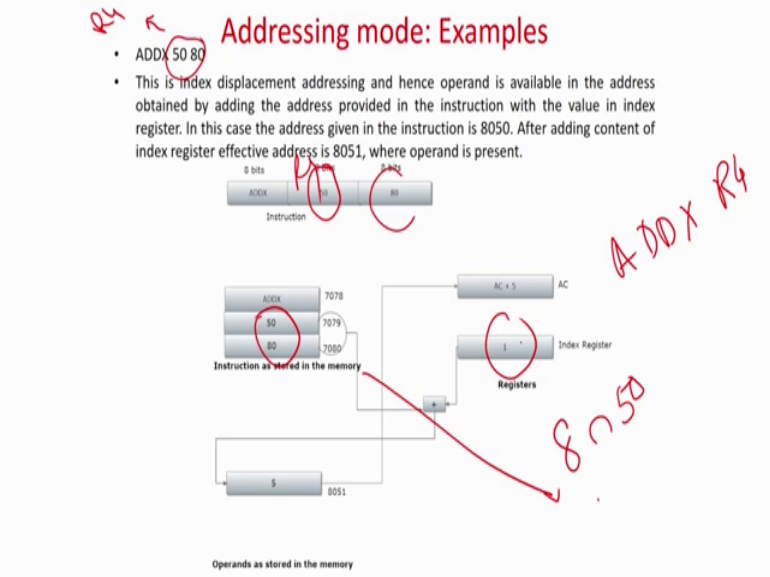
But if for instruction was quite big maybe in this case 8 plus 8 plus 8 16, 32 if this can be 32 bits then the program counter will be first here then will be plus 1 then it will be plus 1 and so forth and then this two we can just take together and you can addresses; that means, you need not address two different memory locations to get the value of the effective address.

(Refer Slide Time: 44:59)




But actually this is not as I told you this is not a very practical situation.

(Refer Slide Time: 45:02)



Because we cannot have such wide memory 32 bit memories are available. But as we are we are discussing in a much more scale down to. So, for us basically this may be a quite large size.

Because nowadays, when you are talking about 32 bit memories when the memory size is becoming so high that 16 bits are not enough to represent the whole spectrum of memory which can be in the levels of gigabytes. There are several other examples we are taking one by one. In this case it is saying it is add immediate 0800 and then it is add indirect and in this case a add and the bit is 8800. So, the content of the main memory value will be present in 8088. So, it will be is a indirect register.

So, what happen actually? So, in this case it as all direct, but this is an indirect one. So, in this case it is saying add indirect the content of memory at the operand is 0800 and it has to be stored with the it will be means accumulator has to be added. So, in is an indirect addressing mode. So, whatever is the op code op code is saying add something to the accumulator, but where I will get the operand there is an indirect one it is saying 0800; that means, this two will be referring to a memory location that is 08000; that means, you add it with the content of memory location 80000.

But what is there it is 8000 so, the content here is 70, but as in indirect addressing mode I cannot do that indirect addressing mode means here again I will get the I have to again refer to this memory location to get the exact memory locations where the data will be available. So, it is pointing to 8 0 8082 8 0 00 so this is the location.

Now in this case it is mentioning as 70 because this location is again 8 bits so, but the whole memory size their length means memory address space is 16 bits, but the width is 8 bit. So, here the content is 70 so, it is actually these two taken together is referring this memory location 8000 so, now it is having the value of 70.

So, the real content of the memory should be available in memory location number 70, but 70 is an incomplete address something more is required. Because the memory address space is 16 bits, but as this width is 8 bit. So, the whole instruction a whole address cannot be placed over here. So, by default I will again take the next value I have to take the both the value because this is 8 bits and I can only store half of the address here so it is 70 FF.

So, basically the effective address is FF 70 which is actually a again a redirection from here to here where actually the data is present which is 5 which is added. So, let me tell you so what happens add immediate 80 8000. So, that is; what is the address of the indirect address of the data so I am going over there the content here is 70.

So, by indirect addressing the contents will be available in memory location number 70, but 70 cannot be a memory location address because the space is 16 bits and this space is only 8 bits. So, I require two words to store the again address so by default 8000 plus 1 this contiguous memory locations are taken so 70 and FF.

So, FF and 70 are taken which is this memory location which is the exact location of the data which is available as 5 so in this way I go on calculation. So, if you can understand that if more values or more number of operands or operand maybe more precise or larger area means larger memory space has to be located then your instruction start becoming multi word instruction.

So, in this case it is a 16 bit memory so you are requiring two words other than the op-code to get the address if the address space is memory space is slightly higher say this 32 bits then you require 1234 4 memory locations to address the whole memory space. So, higher memory space means in this case you require more number of memory locations to do it. So, it becomes a more number of multiple word memory.

Then just to come before just we complete and go to other examples let us take a very simple example of add x which is the displacement addressing 5080. So, in this case 50 is one part, 80 is another part. So, as we all know that basically if you look at it so 80 50.

So, in this case add x that is your displacement addressing we are assuming that and the content is 50 and 80. So, basically and there is a index register. So, basically in this case it is a displacement addressing mode the address is obtained by provided in instruction with the value in the in index register.

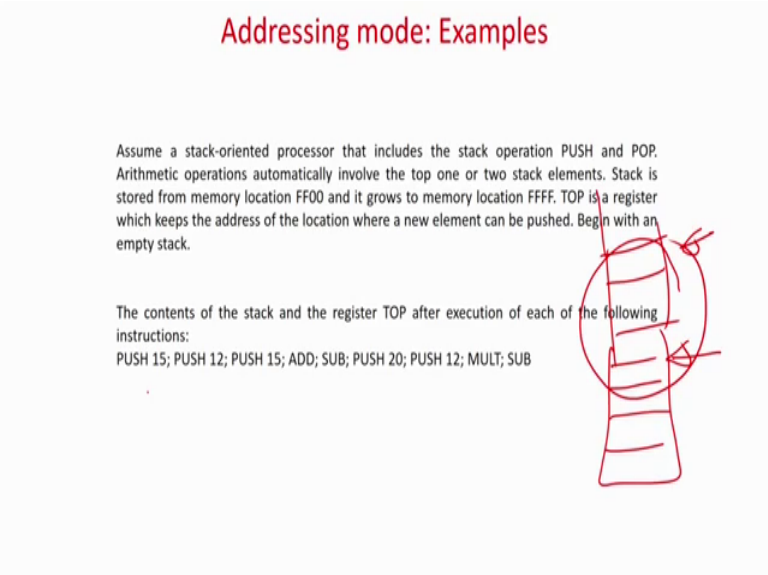
So in fact, actually it should be add X; I am just making a slight mistake. So, again basically maybe I can call it an R 4 which is my basically the index register here I have forgot to mention it should be basically R 4 or something like that that add x R 4 and then 58. So, 58 the instructions starts from 70 the next two bits are this one so in this case 80 50. So, this one will generate the address location of 80 80 50 which is in the

instruction and the instruction is add x R 4 80 50 and R 4 assuming that the content is one which is shown over here.

So, you are adding it to so it will become 80 51 the content of 80 51 is 5. So, you will add with the accumulator to get the value this is simple displacement addressing which I have told you in this case I am explicitly referring R 4 which is my index register assuming the value of 1.

So, the memory location in the instruction is basically add x R 4 and in that the R 4 is added with the content that is 80 50 that is 8 80 50 with 1 that is 80 51 the content of 80 51 is 1 sorry 80 51 is 5 and you get the value.

(Refer Slide Time: 51:16)

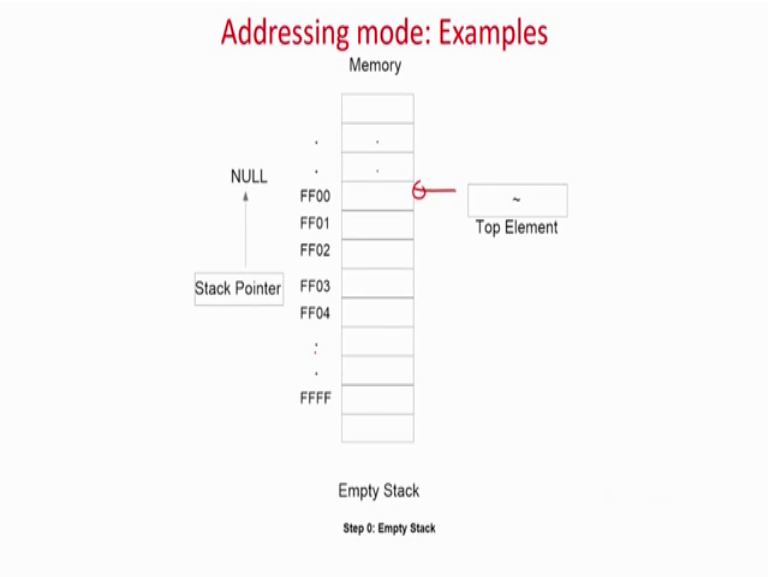



So, in this case the displacement has happened by one because the value of the index register is 1. Next if I increment with that of instruction register by 2 so, it will be accessing memory location number 80 52 and we can keep on doing it then the last addressing mode which is quite simple used mainly in a stack machine.

So, in a stack machine what happens as I told you basically this simple example we have taken. So, let us assuming that there is a stack there is a stack over there and there is a pointer so this stack can go from anywhere. Because we we generally get to access only a certain portion of this stack. So, there is something called a stack pointer and then we can do such type of instruction like push, pop, add, sub. Push means you are here the

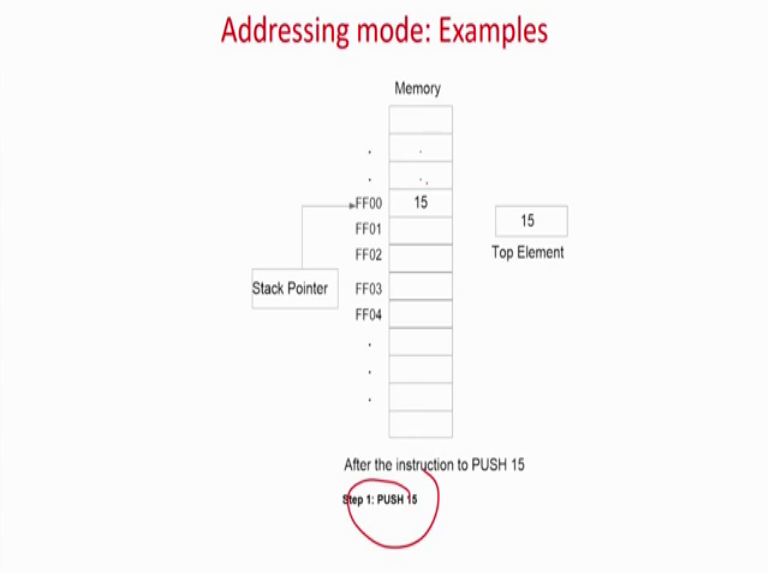
number of instructions would be very limited we have already discussed in some previous units I can push the values you can pop the values and when you want to do some kid of operation it will take the top two elements. So, some examples like push, pop, add, sub some of the examples we will try to show.

(Refer Slide Time: 52:02)



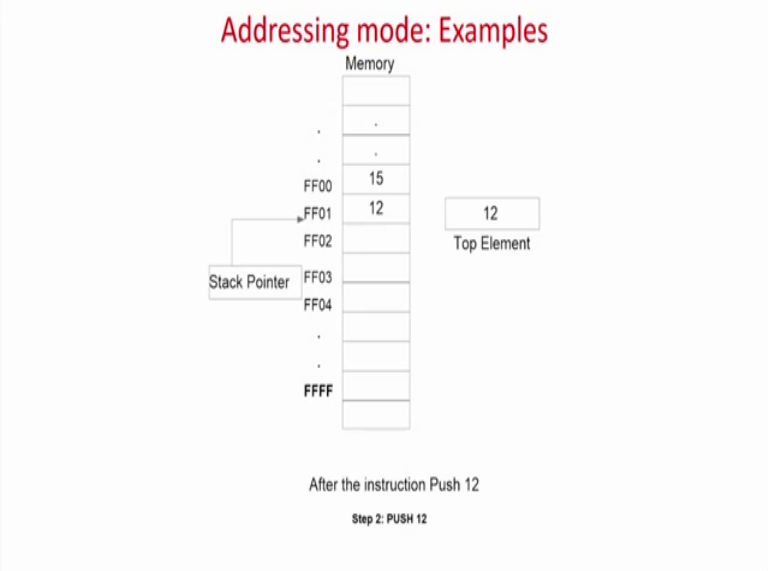

So, this is an empty stack and then this is the stack pointer maybe say that we will start pointing from here this is the top element, this is the whole stack available. And we will do some of the instruction which we have seen in the this question that we will first push 15; then push 12; then 15 then you will add and so on. So, this is the first scenario.

(Refer Slide Time: 52:26)



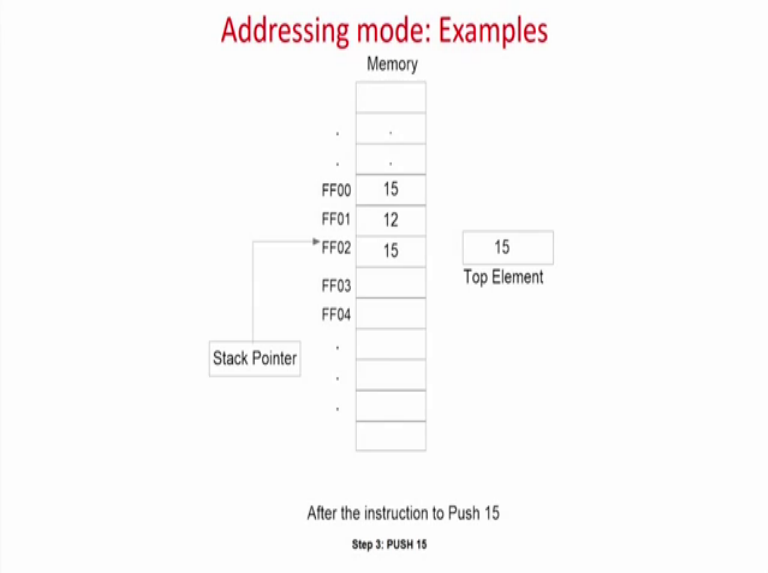

Then we say that push 15; so 15 will be pushed in the top position because if an infinite step because this may be available to program one.

(Refer Slide Time: 42:38)



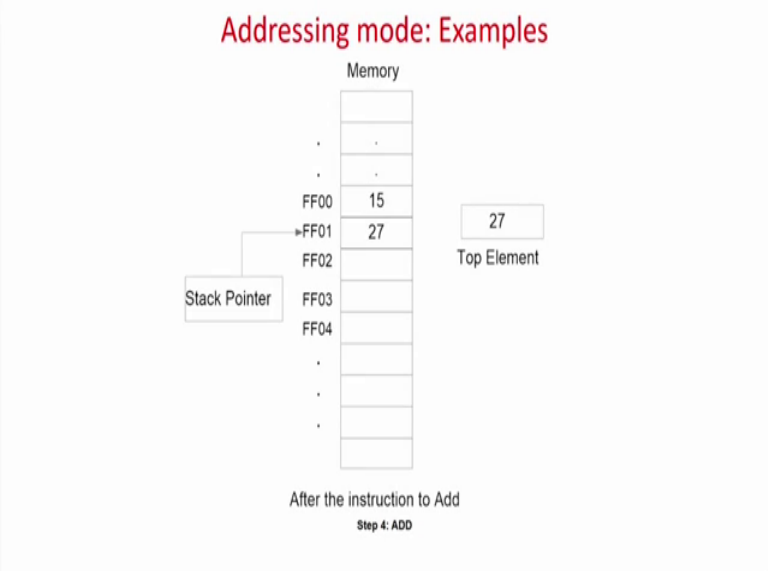

Then the other part maybe available to some other program and so forth. Then it is saying push 12. So, again 12 will be pushed then we again say push 15, so the value of 15 has come next you see I will say add.

(Refer Slide Time: 52:40)



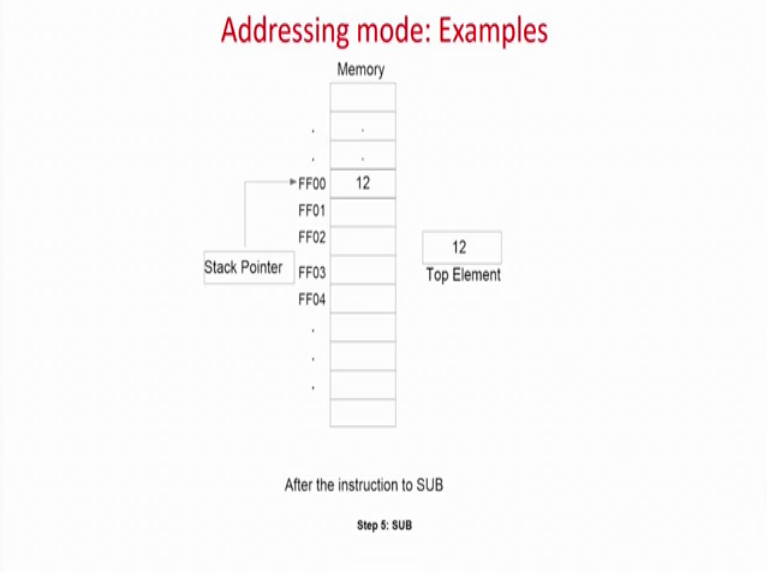

So, what if you say add what happens? If you take the top 2 elements add it and put the value there itself.

(Refer Slide Time: 52:51)



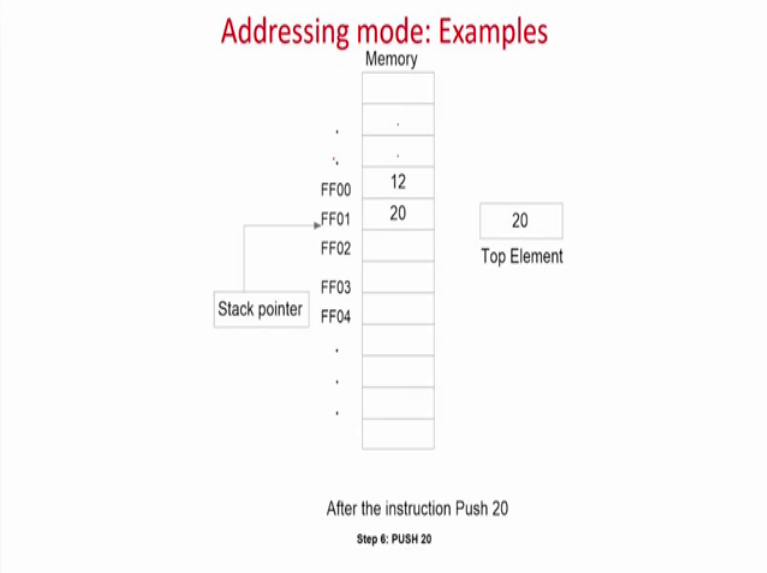

So, it is 15 plus 12 is 27. So, 27 is pushed back over there and this be stack pointer.

(Refer Slide Time: 52:59)



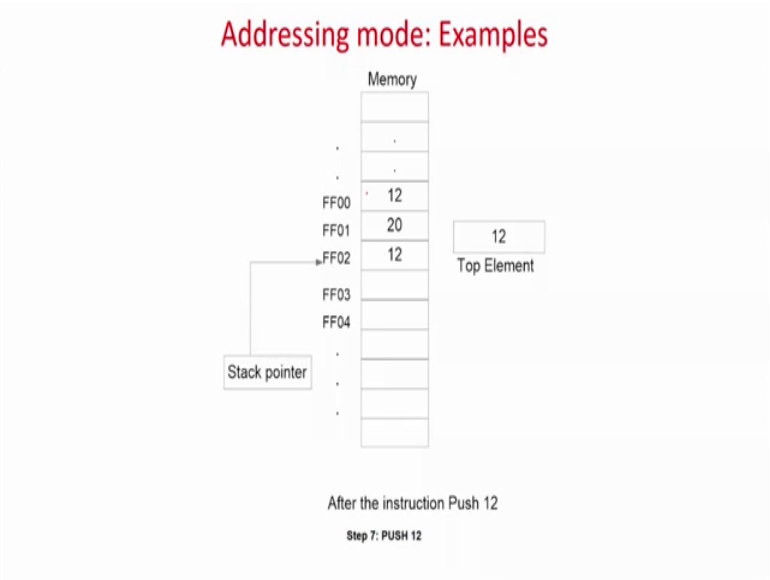

Then I say sub then what will happen if you subtract these 2 values. So, it is 15, 27 minus 15 that is equal to 12 so this subtraction will be that the value of 12 maybe present over here.

(Refer Slide Time: 53:09)



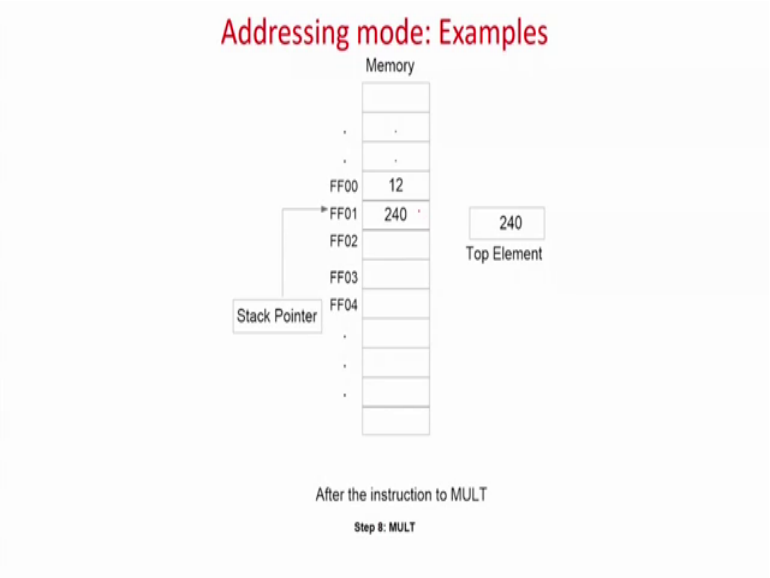

Then I say again say push 20. So, the next value will be pushed on the top.

(Refer Slide Time: 53:14)



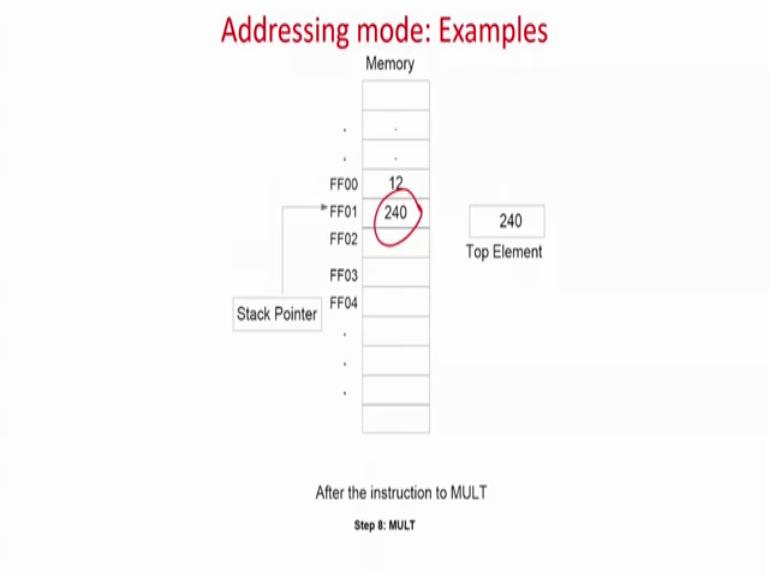

Then in the next instruction is pushed 12, again I push on 12 very very simple operation. Stack version is one of the most simplest computing that is available over here only thing is that it is more slower to it. Because in fact, you can you have to do any two operation you have to first push the values and then you have to operate on the last top two elements and the answer will be pushed over here. Slightly sure way of implementation compared to a general computing where you have lots of different modes of instructions different type of instructions etcetera.

(Refer Slide Time: 53:48)



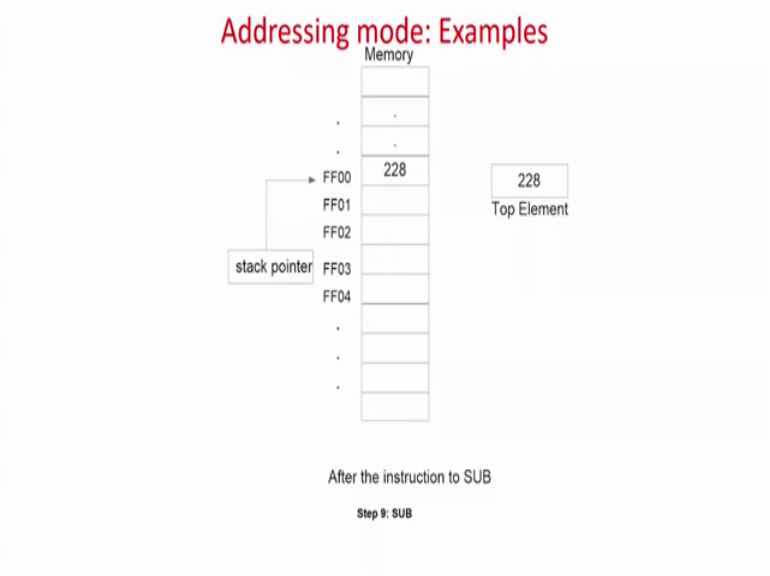
So, I have pushed 12 then this thing multiplied. So, if you multiply it will take the top two elements and multiply it will be 240.

(Refer Slide Time: 53:51)



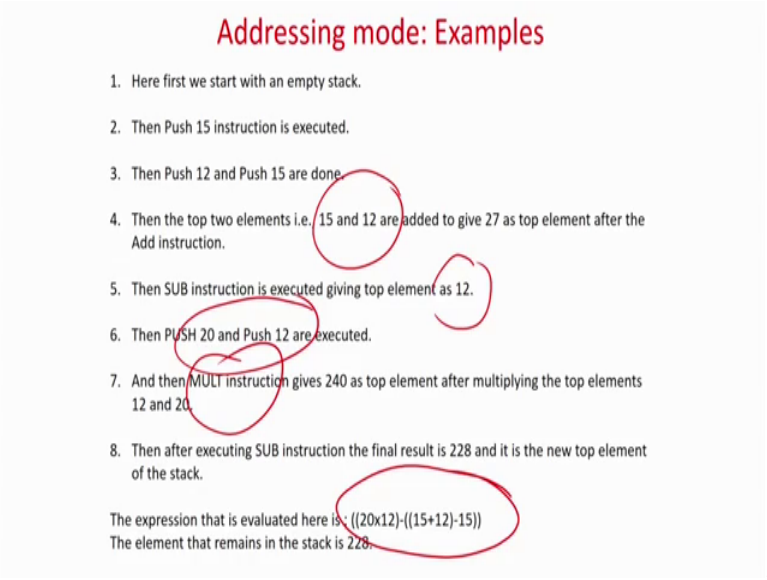

So, 240 will be there over. And then finally as I subtract some example this will subtract these two values and the value will be put over here.

(Refer Slide Time: 53:56)




There is just some examples that how basically a stack machine is implemented.

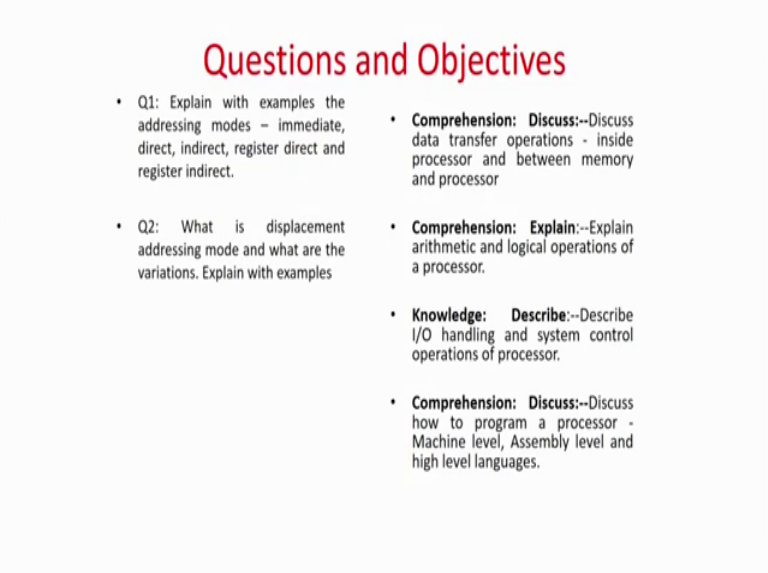
(Refer Slide Time: 54:01)




Only 3 steps push, pop, and operate. Push means some elements will be pushed, pop means the top elements will be popped out to the memory buffer and operation means you will operate the top two elements.

So, if you look at this slide that discussed it is like push 15, push 12 is done. In the top two elements are added. And then you subtract with 12 then again you pushed to two elements and then again you multiply and finally, you will see that this is; what is the expression that is actually computed. So, this stack mode of instruction execution is extremely simple compared to all others, but it is all more slower way of doing it ok.

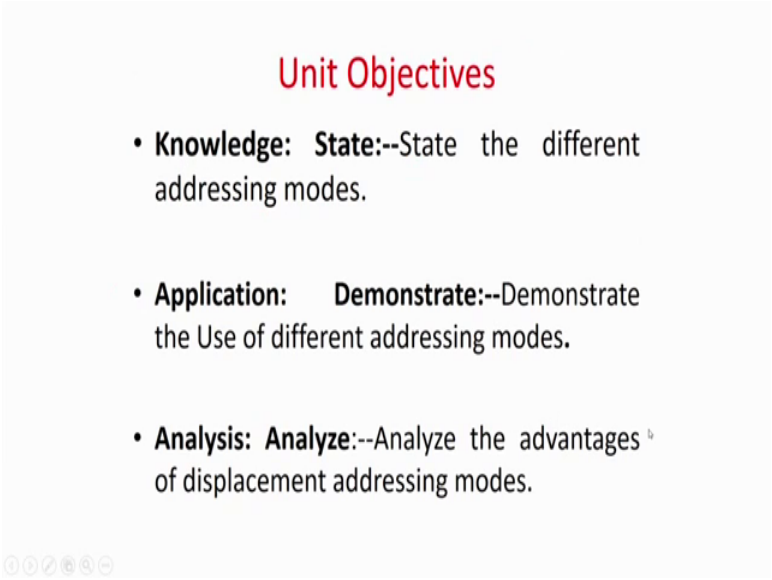
(Refer Slide Time: 54:36)




So, basically this brings us to the end of different addressing modes which we have seen in a wide spectrum from immediate to as long as displacement with index displacement, program counter displacement, based displacement.

So, there is a wide variety of instruction addressing modes available and based on the requirement we choose any one of them. So, towards the end as we have seen let us take very simple examples like explain with example different addressing modes already we have done and it will basically satisfy your objective like explain the discuss if you consider this examples that what we explain with the examples different addressing modes what are the what is displacement and what it is required. So, if you look at the objectives the questions.

(Refer Slide Time: 55:21)

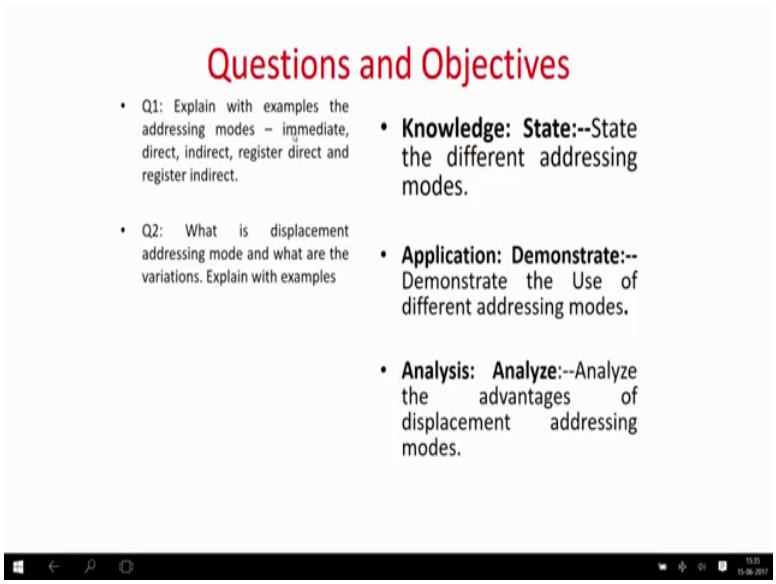



We have here like other the two questions where what are the different type of addressing modes? What are the examples?

So, basically it satisfies the objectives of state the different addressing modes, demonstrate the use of different addressing modes. The second question was if you are refer to the question asked that what are the displacement modes and what it is where it is useful at different type of examples. So, basically it will satisfy the objective of analysis of different type of different type of advantages of different addressing modes.

So just now we have listen to different type of a addressing modes like immediate addressing, then what are the other types of addressing modes which are indirect, then basically registered and so many variations we have discussed. And now let us look at some simple questions and let us see how it satisfies the objectives we have targeted before starting this unit.

(Refer Slide Time: 56:11)




So, explain with the examples the addressing different type of addressing modes like immediate, direct indirect, so many we have already discussed. So, if you are able to I think with this units of the understanding this unit and with these several examples you have discussed you will be able to solve this problem.

And one solving this problem of course, you will be able to demonstrate the use of different addressing modes and also you will be able to state the knowledge of different addressing modes that is with a whenever you are going to explain somebody the different type of addressing modes basically. And the advantages and disadvantages you will be able to meet the objectives of stating the different type of addressing mode, demonstrating the different type of addressing modes with examples and also analyzing the advantages and disadvantages of it.

And already we have seen apart from the normal instruction modes addressing modes there is something called displacement that is from the one part we can easily go to another part based on indexing the generating effective address space based on one part of the instruction by adding it with the explicit addressing and so forth.

So, if the second problem says that what are the displacement addressing modes different types like index, index, base, register etcetera and explaining with that examples and what are the advantages and disadvantages of course this will try to this will be helping me to solve the objective on analyzes the advantages of displacement addressing modes

compared to the other addressing other addressing modes like immediate direct and so forth.

So, just by completing this object unit you will be able to state demonstrator and analyze the advantages and different advantage disadvantages of different addressing modes because what you achieve and how complicated is one addressing mode than the other, but what it gains.

So, basically this brings us to the end of this unit and next unit what you are going to see is that basically what are the instructions we have seen till now are mainly sequential 1 2 3 and so forth. But in a program the basic logic comes if your based on some condition you will do either this or you will do either that without this no program is completes or in fact, no problem can be built without certain condition and logic.

So, the next unit we will be focusing on conditional statements, flags, how a conditional code executes that if it is true you do this and so forth. So, we will be next unit we will be looking at the flags and conditional instructions.

Thank you.