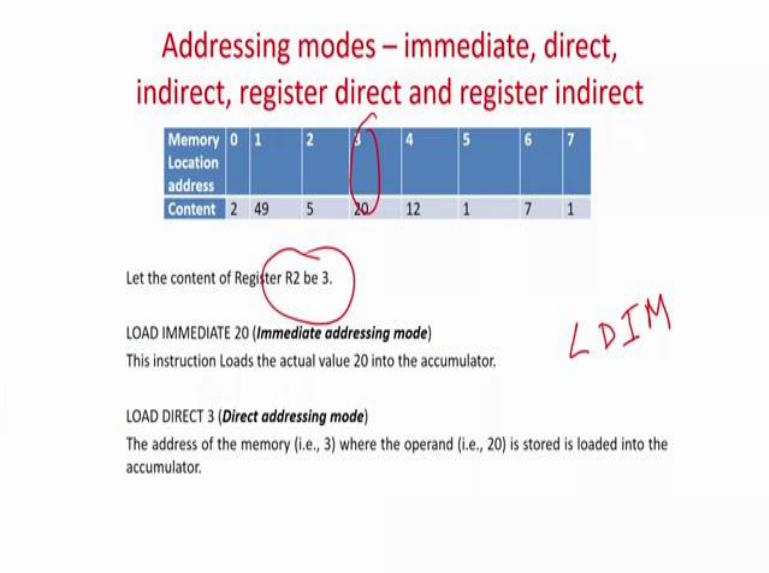
(Refer Slide Time: 16:55)



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 like this way. And let us assume that you have a register whose content is 3.

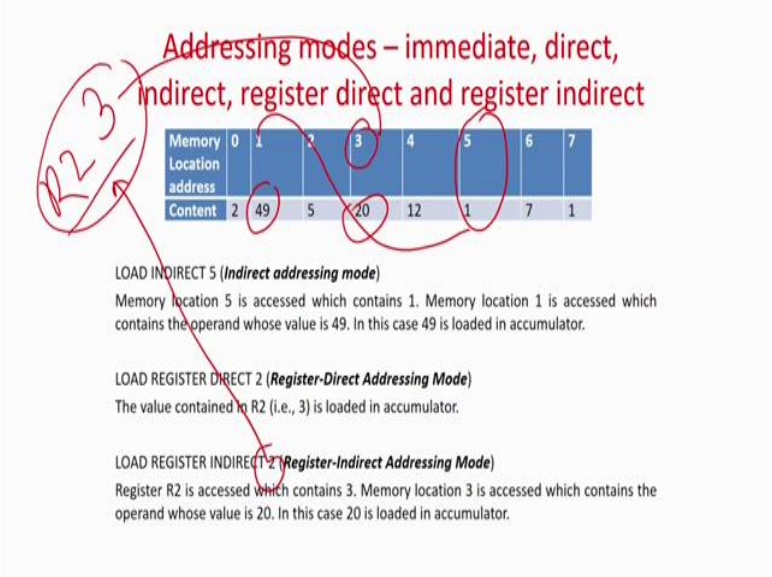
So, let us write an instruction LOAD IMMEDIATE 20 that is may be the opcode can be something like LDI. Generally here I have discussed in a more linguistic manner, but we always have a mnemonic for that so it may be LDI that is load immediate or you can put some times a person may call it load indirect also. So, I you can also have the word load immediate it's 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 3 is a direct addressing mode. So, what is a direct addressing mode? Direct addressing modes means this 3 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 $R2$ has the value of 3; $R2$ 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 it's an indirect. So, 1 is not the data the data is actually present in memory location one.

So, what is memory location 1 present; that is 49. So, the 49 will be stored in your 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 or the data is present that is 49.

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

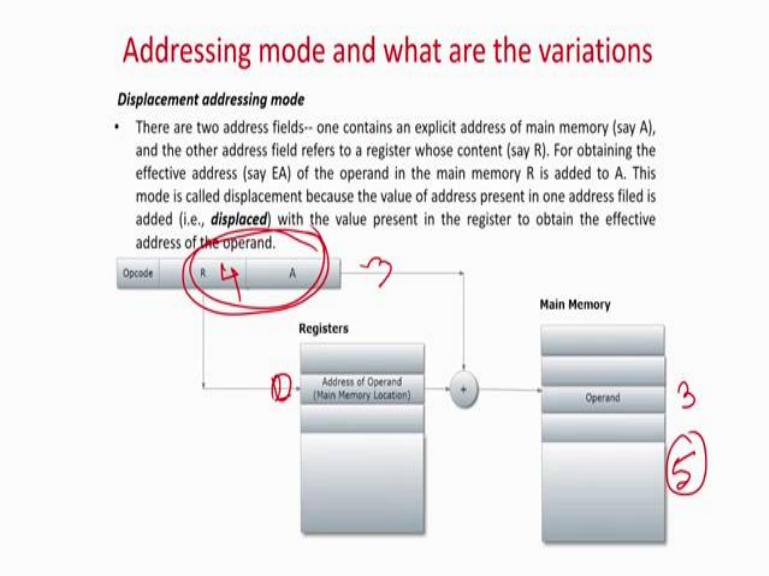
So, as I told you our assumption was register 2 has the value 3. So, it is loaded into the register next is as this normally indirect so this is called register indirect 2.

So, register indirect 2 means so register 2 has a value of 3. So, now it's 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 2, register 2 has 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 were somehow what I told 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 that there are two components of addresses which will be used 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 it has some explicit value and the other address field refers 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, 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 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 another.

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 will 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.

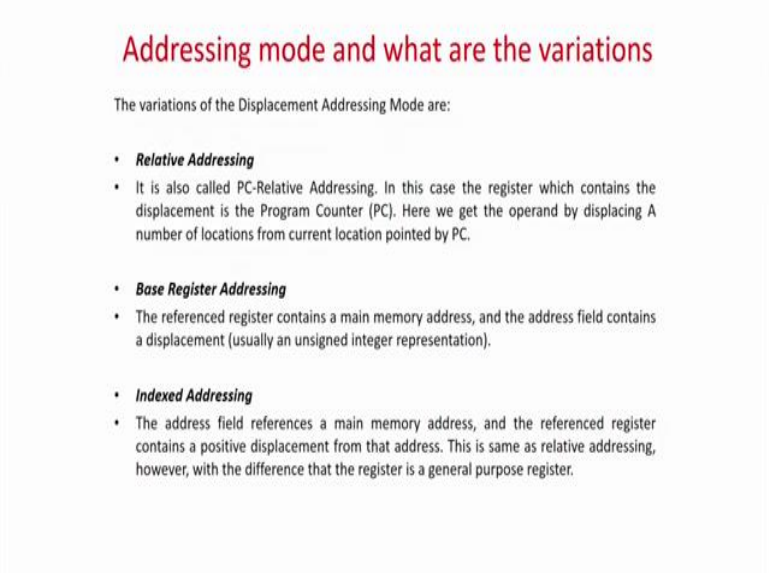
Similarly for array if the data is present in the contiguous location in the memory. So, you can access different elements of an array by such kind of a displacement mode. There is something called relocatable loader in that case what happens say you load the whole code say for example, for program counter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

So, in this case as we 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 + 10.

So, such type of lot of cases 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 addressing is required.

But in this case you have to know that difference from all others are 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 they will be added and the effective address will be calculated. So 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 that is constant in the sense that it 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 like 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 is the register addressing it's the relative addressing. So, is actually the where the displacement is present is the program counter. So, in generally in this case 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 effective location is obtained by the location of the PC, content of the PC, that is 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 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 as the base register instead of the program counter if it is a base register we call it as 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 for the displacement we 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 instead of program counter if you are using $R1$, $R2$, $R3$, $R4$ which is a general purpose register we will call it as a indexed addressing. If indexed register addressing is generally used for loops like for example, $i = j, i = 1$ to 10 then do something.

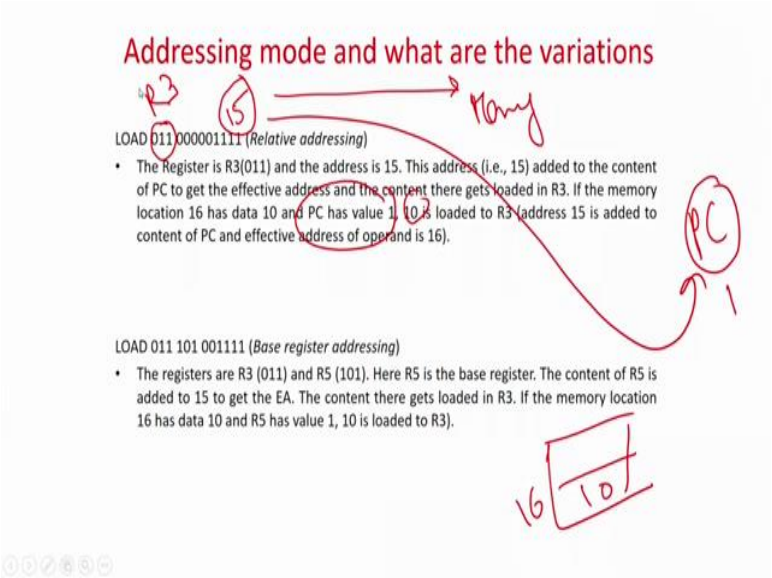
So, that if I refer i to a register then it will actually I called as the indexed register addressing. Because I will map i to a general purpose register like an array if there is an array and I index it with j, so zeroth element, 1 element, 2 element, third element and so forth. So, I call it as the 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 mapped 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 part 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 will 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 functionality 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 + 5. So, this type of addressing is displacement addressing is used. But based on the address or based on the register which is used for that purpose there are different types.

(Refer Slide Time: 26:41)



So, again it is better lot of theory we have discussed. So, it is better to take some concrete examples for 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 $R3$ because this is the register $R3$, and the address is 15 so that means, whatever is the content in $R3$ the address 15 that is this is your address in to the main memory this is location to the memory then it's added to the content of the program counter So, base register.

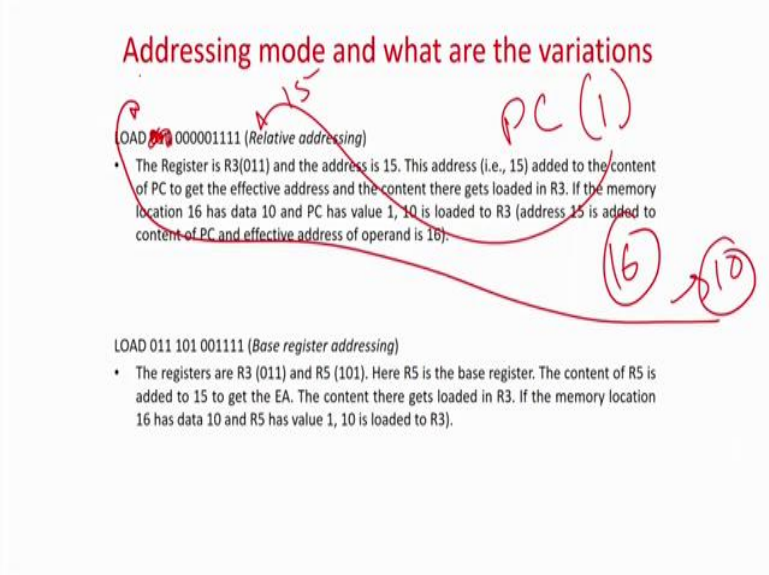
So, the whatever will be the is an immediate addressing relative addressing mode. So, PC is default one of the register and in this case what it is saying the register $R3$ that is register $R3$ is the register the content of this is 15 will be added this 15 will be added to the content of the PC whatever the PC will be have the value and that contain that is 15th memory location has 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 they are assuming that the program counter is 1.

So, if the program counter is 1 then 15 + 1 will be the 16 and whatever is present in the 16 memory location will be added will be loaded to register $R3$. So, what happens 15 is the memory location main memory location program counter is 1, 15 + 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 to register 3 because they are assuming at the memory location 16 address 16 has the value of 10. So, program counter was 1 this explicit addressing was pointing at 5. So, add it and load the value of 10 that is the operand to register $R3$. So, if it is the accumulator type of a addressing then what we can do let us make it a very simple 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)

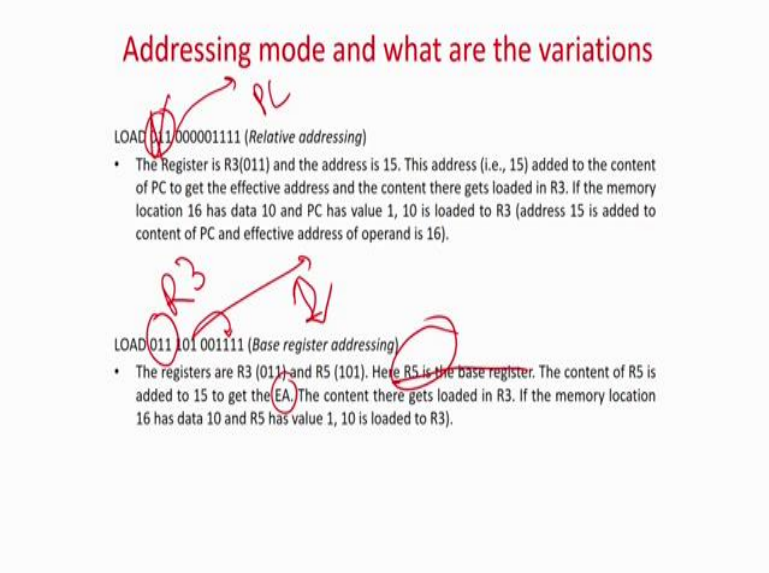


So, what it will mean that whatever is in the program counter content in this case is assuming to be 1 you add it with 15 then memory location is becoming 16, 16 memory location is having the value of 10 this 10 you load it to the accumulator, but in this case that will go to accumulator, but in this example they are not taking the example of an accumulator direct machine we are taking the value of $R3$. So, the content will be loaded to the accumulator sorry content will be loaded to register $R3$.

So, whether it's a register $R3$ or whether it's 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 mode take the value of value of PC add to 15 and load at some place in this case it is register 3 if I drop this and assume 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 register $R3$.

(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 $R3$ sorry instead of using a PC if I am using a very special register in this case they are assuming that the R5 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 R5 is added to 15 to get the affective address and the content is loading into $R3$. So, in this case if we assume that this just like program counter if these are register R5 that is the base register is the value of 1.

So, it will be again 1 + 15 into 16, 10 is there in the 16th memory location it will go and load into $R3$ say for example, but if I have the base register value of 2. So, it will be 15 + 2, 17. So, whatever will be available in the memory location 17 will be loaded to register $R3$. 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 we are say assume when we are say generating a code in the offline of just after compilation then we assume that the code starts from memory location 0.

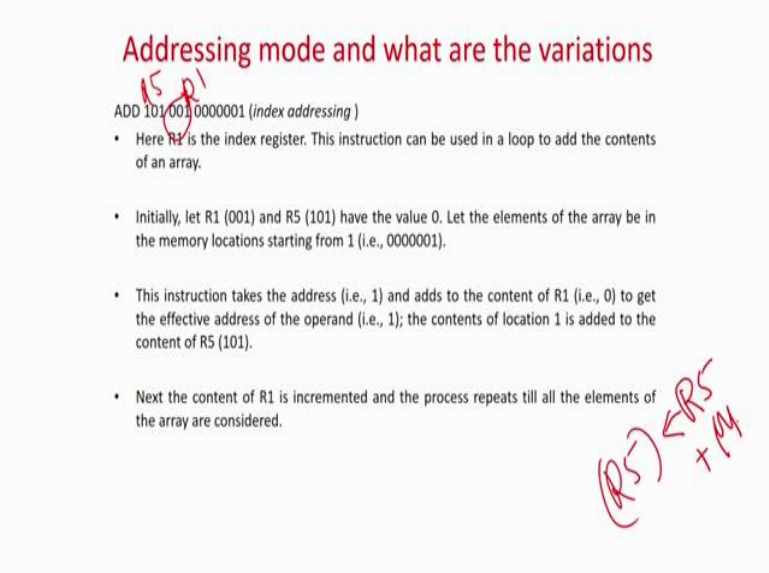
But due to non-availability of the zeroth memory location to be loaded and executed it sometimes get loaded at starting from 100 memory location. So, the program counter will be starting from 100.

So, in this case program counter value 0 will be added to 100 which is actually this value we 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 execution starts from 0 all the addresses of the assembly language code are assumed from 0. But then when it 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. Say starting from 100 so you put 100 over there. So program counter will start from 0 so it will be 0 + 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 for that we have a base register in which 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 $R1$, $R5$, $R10$, $R12$ 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 $R1$ is the index register it can be any general purpose register in this case what they are doing they are using it as a loop. So, as I told you the one of the most important utility of index, index addressing is loops basically.

So, here $R1$ is the index of some loop. So, it is saying that whatever available content of $R1$ you add to 1 so that will be the effective memory location. In this case if the $R1$ is 0 so the effective memory location is 0 + 1. So, the first memory location the constant the contents of the first memory location will be added to whatever was the content at R5 and will be stored back over here. So, R5 is equal to R5 plus content of $R1$ that is the memory location. So, what is the memory location over here? So, let me just write it.