

(Refer Slide Time: 34:30)

Addressing mode and what are the variations

ADD 101 001 0000001 (index addressing)

- Here R2 is the index register. This instruction can be used in a loop to add the contents of an array.
- Initially, let R1 (001) and R5 (101) have the value 0. Let the elements of the array be in the memory locations starting from 1 (i.e., 0000001).
- This instruction takes the address (i.e., 1) and adds to the content of R1 (i.e., 0) to get the effective address of the operand (i.e., 1); the contents of location 1 is added to the content of R5 (101).
- Next the content of R1 is incremented and the process repeats till all the elements of the array are considered.

$s = s + a[i]$ $R5 \leftarrow R5 + M(R1) + 1$

So, what is this I am effectively trying to do? So, in this case register R5 will be R5 plus what is the content of the memory location how can you find out if whatever is R1 will be added to the content of the memory location how the memory location is calculated it is content of R1 + 1.

So, I add value of 1 and I add what is the content of R1 that will be the effective memory location I get the operands from there add to R5 content and store it in the R5 itself. So, initially they are assuming that R1 is having the value of 1 and R5 has the value of 0 that is R5 is reset.

So, it will be something like $s = s + i$. So, s is reset and the i is going to be implemented the and in fact what is i? So in fact, it's something like $s = s + a[i]$. So, a is the array and i is your instruction. So, R is basically nothing but in this case your R1 and s is nothing but in this case your R5 ok.

So, now see they are saying that the initial content of R1 is 1 and R5 sorry R1 is 1, R1 is 1 sorry this R1 is 1 and R5 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 R1 has the value of 1.

So, the so what will happen the instruction takes the address 1 and adds to the content of R1. So, the content of R1 and R2 both have 0. So initially both of them has 0 value so that is this 1

will be added to the content of $R1$. So, $R1$ is having a value 0. $0 + 1$ 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)

Addressing mode and what are the variations

ADD 101 001 0000001 (index addressing)

- Here $R1$ is the index register. This instruction can be used in a loop to add the contents of an array.
- Initially, let $R1$ (001) and $R5$ (101) have the value 0. Let the elements of the array be in the memory locations starting from 1 (i.e., 0000001).
- This instruction takes the address (i.e., 1) and adds to the content of $R1$ (i.e., 0) to get the effective address of the operand (i.e., 1); the contents of location 1 is added to the content of $R5$ (101).
- Next the content of $R1$ is incremented and the process repeats till all the elements of the array are considered.

Handwritten formula: $R5 \leftarrow R5 + M(R1) + 1$

So, 0 there is garbage, 1 there is some data, 2 there is some data and so forth. So, initially your both $R5$ and $R1$ are reset. So, generally it has the value of 0 this also has the value of 0, so $0 + 1$ 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 $R5$ and it will be stored over here. Next what you will do you will increment the value of the register number $R1$. So, $R1$ will be contents of one will be added to the content of $R5$ that 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 $R5$ 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 $R5$. Next the content of $R1$ is incremented so this one will now have the value of 1. So, $1 + 1$ will be 2. Now it will be pointing to this next memory location that will be again loaded added with the content of 5 and stored back.

So, there is something happening like $s = s + a[i]$ 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 it's a very simple example of an indirect sorry

index addressing mode, it's again a displacement addressing mode, but this in this index register is our own our own defined or user available register which is $R1$ in this case, in the other way it can be any user register which can be used by a programmer ok.

(Refer Slide Time: 37:58)

Addressing mode: Examples

Consider a CPU with 8-bit data bus and 16-bit address bus.

The memory is byte organized.

The instruction format is one operand.

Length of instruction is either two bytes or three bytes.

- First byte indicates the op-code.
- For two bytes length instruction second byte indicates the data.
- For three bytes length instruction, second byte indicates the lower eight bits address and third byte indicates the higher eight bits address.

The instructions are stored in consecutive memory location starting from memory location 0770_{16} .

So, now basically now we are going to see some more examples which will give you a more in depth idea of how basically addressing mode happens where the address, where the values are located etcetera. Because in most of the cases as I was saying that the instruction is basically opcode and some operand or some addresses, but sometimes the size of the instruction cannot be such nice or such of the length of the or the width of the memory cannot be so good that it will hold the it will hold the whole instruction in 1 word there can be in multiple words; that means, the opcode will be one place and some part of the operand will be there in the word and the and the space 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's 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 opcode. So, what do you mean by that? So, in this case as I told you the instruction has some specific characteristics that

is the opcode 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 an opcode. So, when I say that basically this is your opcode. 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 an 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 an 8 bit word.

(Refer Slide Time: 40:05)

Addressing mode: Examples

Consider a CPU with 8-bit data bus and 16-bit address bus.

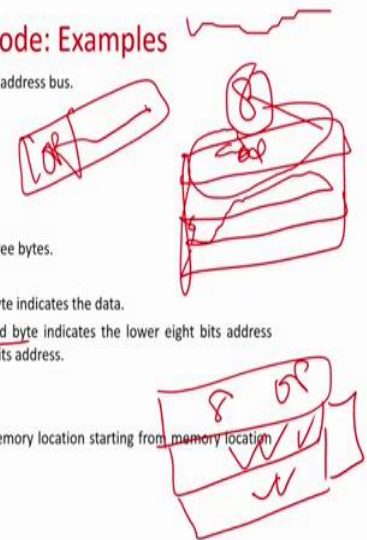
The memory is byte organized.

The instruction format is one operand.

Length of instruction is either two bytes or three bytes.

- First byte indicates the op-code
- For two bytes length instruction second byte indicates the data.
- For three bytes length instruction, second byte indicates the lower eight bits address and third byte indicates the higher eight bits address.

The instructions are stored in consecutive memory location starting from memory location 0770₁₆.



So, I am saying that the opcode takes the whole part of it that is because I want because we 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 opcode 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 because in this case there cannot be any 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 opcode. So, obviously, this 8 bits is the opcode 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 be if it's a direct addressing mode so, it will be reference of a memory register.

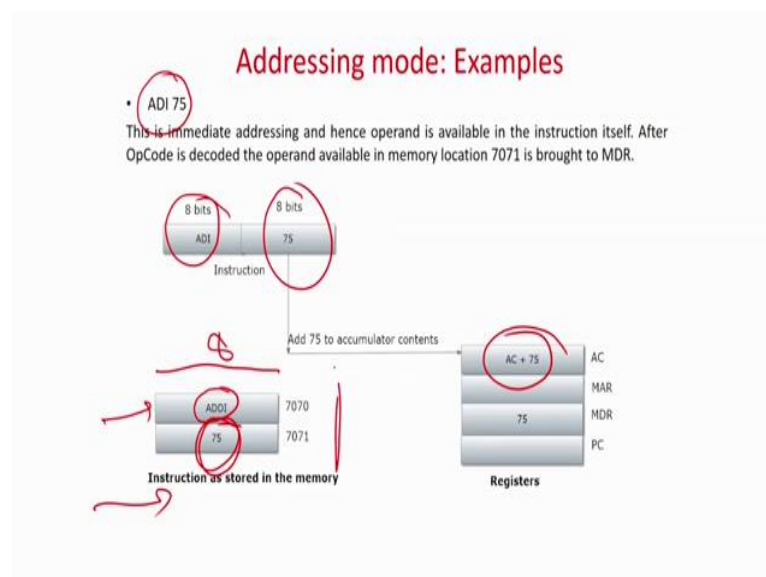
For 3 byte instruction the last two basically these two will be your data or it will be pointing out in some memory location and this one will be your opcode. So in fact, what are we showing that if the in this case the memory is 8; 8 in length and the op means instruction set is larger because the opcode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

So, you the whole instruction has to be spread out in to 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, one instruction after another, but if there is multiple word instruction then first one, if it's a 2 word instruction then we jump to 3, then if it is a single word instruction then again we 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 it for this present example we are assuming that we the instructions are stored in a memory location which is continuous and it starts from 0770 hex. So, this is this represents pictorially.

(Refer Slide Time: 42:03)



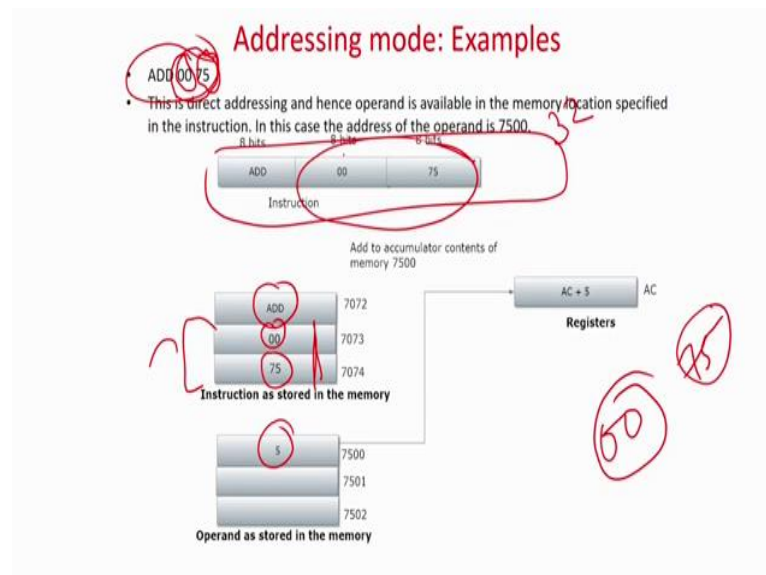
So, this is the first instruction is add immediate 75 so; that means, I told you it's a 8 bit as well as your data is also assumed to be 8 bits. Now in this case I told you the whole memory size is only 8 bits. So, the instruction this whole instruction together cannot fit into a single word add

immediate ADI that will fit in the first word that is 7070 hex and a next that is the your data as it's 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 +1 it will be +2 because in this case it is a 2 word instruction it is a 3 word it will be +3.

So, what happens? So this is the case the data is 75 add immediate means load the value of 75 sorry add the value of accumulator with 75 and store back in the accumulator. So, this 75 is taken, it is added to the accumulator and it will be in 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)

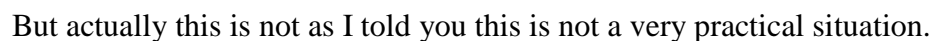


Next is add it's a direct that is it is saying that the this is your memory location. So, it is saying that add will be first gone that is 8 bit is taken then now in this case it is not an immediate addressing mode it's a direct addressing mode.

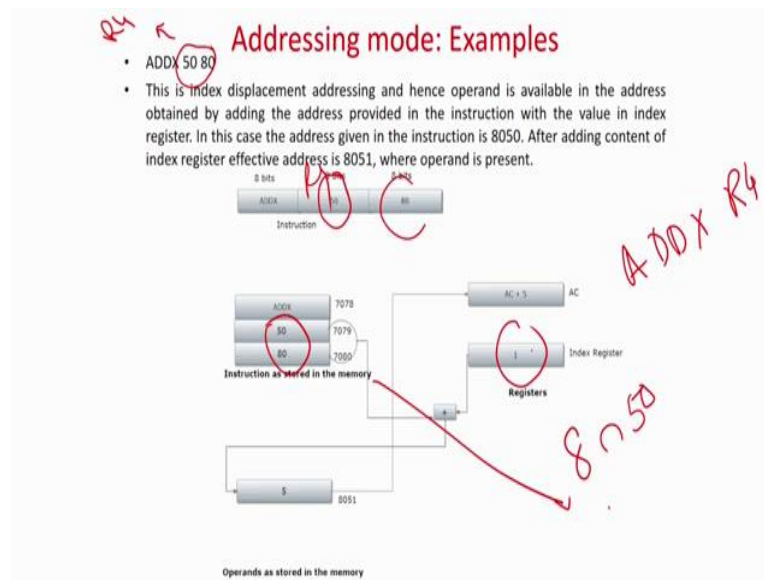
So, it is referring to a memory location. Memory location in this case is requiring 16 bits that is first hex value, second hex value, third and fourth. So, now obviously, this 2 is 4 bits and 8 bits and these 2 is 8 bits 00 is 8 bits that is 00000000, 7 means 0111 and 5 is 0101. So, 00 and 75 so this is taking 8 bits and this is taking 8 bits. So, this will again occupy one word space in the memory. So, this is the opcode then 00 is the LSB, and this one is the MSB of the address. So, it is stored over here.

So, the hardware is becoming more complex because 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.

(Refer Slide Time: 44:59)



(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 scaled 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. So in this case it is saying it is add immediate 0800 ok and then it is add indirect and in this case add and the bit is 8800. So, the content of the main memory value will be present in 8088. So, it will be it's a indirect register.

So, what happened actually? So, in this case it not 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 opcode, opcode is saying add something to the accumulator, but where I will get the operand this is an indirect one it is saying 0800; that means, this two will be referring to a memory location that is 8000; that means, you add it with the content of memory location 8000.

But what is there it is 8000 so, the content here is 70, but as it's an 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 8000 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 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 FF70 which is actually 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 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 means larger memory space has to be located then your instruction starts becoming multi word instruction.

So, in this case it's a 16 bit memory so you are requiring two words other than the opcode to get the address if the address space is memory space is slightly higher say this is 32 bits then you require 1234, 4 memory locations to address the one 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 8050.

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 an index register. So, basically in this case it's 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 ADDX; I am just making a slight mistake. So, again basically maybe I can call it as *R4* which is my basically the index register here I have forgot to mention it should be basically *R4* or something like that ADDX *R4* and then 5080. So, 5080 the instructions starts from 70 the next two bits are this one so in this case 8050. So, this one will generate the address location of 8050 which is in the instruction and the instruction is ADDX *R4* 8050 and *R4* assuming that the content is one which is shown over here.

So, you are adding it to so it will become 8051 the content of 8051 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 *R4* which is my index register assuming the value of 1.


So, the memory location in the instruction is basically ADDX *R4* and in that the *R4* is added with the content that is 8050 that is 8050 with 1 that is the content of 8051 is 5 and you get the value.

(Refer Slide Time: 51:16)

Addressing mode: Examples

Assume a stack-oriented processor that includes the stack operation PUSH and POP. Arithmetic operations automatically involve the top one or two stack elements. Stack is stored from memory location FF00 and it grows to memory location FFFF. TOP is a register which keeps the address of the location where a new element can be pushed. Begin with an empty stack.

The contents of the stack and the register TOP after execution of each of the following instructions:
PUSH 15; PUSH 12; PUSH 15; ADD; SUB; PUSH 20; PUSH 12; MULT; SUB



So, in this case the displacement has happened by one because the value of the index register is 1. Next if I increment the value of index register by 2 so, it will be accessing memory location number 8052 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 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 kind 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.