

Microprocessors and Microcontrollers
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 27
8051 Microcontroller (Contd.)

(Refer Slide Time: 00:20)

Addressing Modes

Register Addressing – either source or destination is one of CPU register

```
MOV R0, A
MOV A, R7
ADD A, R4
ADD A, R7
MOV DPTR, #25F5H
MOV R5, DPL
MOV R, DPH
```

Note that MOV R4, R7 is incorrect
MOV 4, 7 ⇒ M[4] ← M[7]

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Other addressing mode is the register addressing, so where the source or destination is one of the CPU registers. Like typical example like MOV R0 comma A, where R0 gets the content of A, they MOV A comma R7; A gets the content of R7 MOV A comma R4. So, like that we can make one of the operands one of the source or destination as the CPU register.

Like say this instruction MOV DPTR hash 2 to 5F5H. So, this DPT; so, DPTR pair will get this value. So, DPH will get 2 5 and DPL will get F 5 or we can have this individual registers accessed DPL DPH. So, R5 comma DPL; so, DPL's content will come to R5. So, there should be a number here say R5 R4 or R3 something like that the register number is missing that comma DPH. So, that register will get the value of the DPH register. So, as I told previously that I cannot have an instruction like MOV R4 comma R7. So, this is not allowed ok; so, this is not allowed.

So, if you really want to MOV the content of current register 7 to register 4. So, you have to give the corresponding memory address. So, you can write something like this that

MOV 4 comma 7. So, essentially what will happen is that the memory location 4 will get the content of memory location 7. And assuming that your memory locations are current register bank is register is the bank 0; then this will be transferring the content of register 7 to register 4. If you are using some other banks, appropriately we have to calculate these two memory location addresses and then we have to put those values.

So, this way we can use this addressing; we can use this register addressing for transferring control between the registers transferring values between the registers.

(Refer Slide Time: 02:15)

Addressing Modes
Direct Mode – specify data by its 8-bit address
 Usually for 30h-7Fh of RAM

```

Mov a, 70h      ; copy contents of RAM at 70h to a
Mov R0,40h     ; copy contents of RAM at 40h to R0
Mov 56h,a      ; put contents of a at 56h
Mov 0D0h,a     ; put contents of a into PSW
  
```

DATA MEMORY (RAM)	
INTERNAL DATA ADDRESS SPACE	ADDRESS SPACE
0xFF	Upper 128 RAM (Indirect Addressing Only)
0x80	Special Function Register's (Direct Addressing Only)
0x7F	
0x30	Lower 128 RAM (Direct and Indirect Addressing)
0x2F	
0x20	
0x1F	
0x00	

The diagram also highlights '8-bit Addressable' and 'General Purpose Registers' in the lower 128 RAM range.

Then we can have direct mode. So, here in the direct mode; so, we specify data by a by 8 bit address. So, usually the addresses are in the range of 3; 30h to 7 Fh. So, we can write like MOV a comma 7 0 h; so, content of memory location 70h they will be coming to the A register. So, this is that 30 to 7F memory range that I was talking about. So, this is in the direct mode we can use them then R0 comma 4 0 h copy content of memory location 40h to R0.

So, this way we can do this thing like. So, this one say MOV was 0D0h comma a. So, 0D0; so, this happens to be the PSW register; this is the special function address h 0; D0 is the special function address for this register PSW. So, there the content of accumulator will the will go there; so, if we put the content of a into the PSW register.

(Refer Slide Time: 03:22)

Addressing Modes

Direct Mode – play with R0-R7 by direct address

MOV A,4 ≡ MOV A,R4

MOV A,7 ≡ ~~MOV A,R7~~

MOV 7,2 ≡ MOV R7,R6

MOV R2,#5 ;Put 5 in R2

MOV R2,5 ;Put content of RAM at 5 in R2

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then we can play with R0-7. So, then we have got the direct mode; so, in the direct mode. So, we can have these instructions like MOV A,4; so, instead of writing register. So, when you are writing like MOV A,R4. So, it is you can get this it is equivalent to MOV A,4.

Then we can have this MOV 7,2 which is equivalent to MOV R7,R6; then MOV A,7. So, this is correct, but this one we cannot write; MOV A,R7. So, this is not allowed like MOV R2,#5; so, put 5 in R2 and MOV R2,5. So, it content puts the content of RAM at 5 in R2; so, this we have already discussed. Only thing to note is that this MOV A,R7; so, this is not allowed in. So, R7 cannot be used in the direct addressing mode.

(Refer Slide Time: 04:18)

Addressing Modes

Register Indirect – the address of the source or destination is specified in registers

Uses registers R0 or R1 for **8-bit** address:

```
mov psw, #0           ; use register bank 0
mov r0, #0x3C
mov @r0, #3           ; memory at 3C gets #3
                       ; M[3C] ← 3
```

Uses DPTR register for **16-bit** addresses:

```
mov dptr, #0x9000    ; dptr ← 9000h
movx a, @dptr        ; a ← M[9000]
```

Note that 9000 is an address in external memory

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

You can use this register indirect mode the address of the source or destination is specified in register. So, you can say like say PSW you can say like mov psw has 0. So, so, that sets that usage of register bank 0 and then we are telling that mov r0 comma 0 x three C. So, r0 gets the value of this 3 C. And then at the rate r0; we are putting the value 3. So, what will happen is that since r0 is having this value 3 C.

So, this 3 C will be used as the memory address and then this memory address will be going to this memory address 3 C will be getting the content that we mentioned here. So, when we execute this instruction MOV at the rate r0 comma hash 3; so, it will be getting the memory location 3 C will get the value 3.

So, only the registers R0 and R1 can be used for this purpose. So, you cannot use any other register like you cannot have any instruction like mov at the rate r2 comma hash 3. So, that is not possible only R0 and R1 and we have to be careful that R0 and R1 they have been initialized with proper values.

So, in this case when we do this so, we are setting R0 first of all we set the bank then in that bank in the R0 register; we are putting this value 3 C and then doing this movement or you can fall.

So, that is for internal memory operation. So, internal memory when you are trying to access in indirect mode; so, you can do it like this. So, if you are trying to use external

memory in indirect mode then we have to use this DPTR register ok. So this instruction this pair of instructions; so, they will be putting the content they will putting get the content of memory location 9000 into the A register and 9000 is an external memory; external memory location.

So, we first of all MOV the value 9000 to that dptr register ok. So, DPH gets 9 0; DPL gets 0 0. So, DPTR pair gets 9000 h and then we say the mov x a comma at the rate dptr. So, it will be accessing the external memory and memory location 9000 from there the content will come to accumulated at a. So, mov x is not allowed for internal memory access; similarly mov is not allowed for external memory access.

So, this x; so, this will mark that a processor should is trying to access the external memory.

(Refer Slide Time: 06:56)

Addressing Modes

Register Indexed Mode – source or destination address is the sum of the base address and the accumulator(Index)

- Base address can be DPTR or PC

```

mov dptr, #4000h
mov a, #5
movc a, @a + dptr ; a ← M[4005]
    
```

Lookup Table

$f(x)$	DPTR
0	4000
1	4001
2	4002
3	4003
4	4004
5	4005

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then we have got this register indirect mode; the source and destination addresses, the sum of the base address and the accumulator some accumulator can be used as an index. Like say this one; so, dptr hash 4000 h mov dptr comma has 4000 h dptr gets 4000; then we mov a comma 5 and then we say mov c a comma at the rate a plus dptr here. So, dp a is current content is 5. So, this DPTR value will be added to that; so, that way we get we will get the value 4005. And then so there from this external program memory; so, we are getting the value into the a register.

So, this is useful when you are having some sort of look up tables ok. So, what happens is that this whenever we are having some lookup table. So, maybe we have got some function $f(x)$ and for different values of x ; we have got a table. So, this is the x and this is the $f(x)$. So, this is 0, 1, 2, 3; so, up to some value say up to 100; we have computed the values and those values are stored in a ; so, these are the different values. So, while writing the program; so, instead of writing the code separately what we do is that we take a portion of memory.

(Refer Slide Time: 08:21)

Addressing Modes

Register Indexed Mode – source or destination address is the sum of the base address and the accumulator(Index)

- Base address can be DPTR or PC

```

mov dptr, #4000h
mov a, #5
movc a, @a + dptr

```

$a \leftarrow M[4005]$

The diagram shows a memory stack with addresses 5200, 740, 760, and 780. A bracket indicates a range from 5200 to 4000. A handwritten note shows a pointer 'a' pointing to the value 5, and another note shows 'a' pointing to the memory location M[4005].

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And then there we write down the values of this function f at various locations. The first location has got the value of function $f(0)$, then the next one is at $f(1)$, next one is that $f(2)$; like that. This is typically useful when you are say for example, doing some filtering operation where these signal samples are multiplied by filter coefficients. So, filter coefficients are fixed ok; so, that are not going to change. So, they can be the filter coefficients can be stored in this type of lookup table.

Now, since these values are not going to change. So, in some sense we can say that they are part of code as the values are not going to change. So, what we do they can be; so in your program that you are writing. So, this can be a part of the programs after that we have got the main program, but this entire thing can be treated as the program. Now you need to access the contents of these locations for doing some operation.

Now this particular example that we have here; so, what it is doing. So, somehow suppose this is stored from the external memory location 4000 onwards. So, 4000 4001 like that; so, we are trying to access the fifth value. So, if 5 we are trying to access the location with the value of say if 5, then with this 4000 we add 5 so, that it comes to in this particular address, comes to this particular address and then `movc a, @a + PC` at the rate a plus `dptr`; so, what this instruction what it.

So, with the `dptr` the value of A register will be added and at the rate it is on this whole thing ok; at the rate is on a plus `dptr`. So, that is the semantics; so, that value becomes 4005. So, memory location 4005's content comes to the accumulator. So, that way you get the content of this location. So, this is very much useful when you have got this type of look up tables stored as arrays in the external memory and then you can use it for in your code ok.

So, there for that purpose this has been thought about and since these microcontrollers are used for embedded application. So, many times we have got this type of situation. So, maybe we have to have some complex function. So though it is difficult to compute it every time; so, what is done is that they are computed once only at the design time and the values are kept as lookup table and that that becomes a part of the code. So, that can be done.

(Refer Slide Time: 11:03)

Addressing Modes

Register Indexed Mode continue

- Base address can be `DPTR` or `PC`
`ORG 1000h`
`1000 mov a, #5`
`1002 movc a, @a + PC ; a ← M[1008]`
PC → `1003 Nop`
- Table Lookup
- `MOVC` only can read internal code memory

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, next see another one. So, where it is; so, we can also have this PC; this base address can be DPTR or the program counter. So, we have seen an example where the base address is DPTR; so, this other one is. So, whenever other possibilities that we can we can we can have this thing from this with respect to PC like mov a comma hash 5; then 1002 movs at 1000 to we have got mov c a comma at the rate a plus PC; so, a plus PC.

So, that will be having this memory location 1008. So, that so; this requires 1000 and this is after this instruction has been faced ok. So, after this instruction has been faced the PC value is 1003. So, with that 1003; this 5 will be added; so, it will become 1008. So, that way we can have this 1008 remote location content available in the a register. So, this mov c the instruction it can read internal code memory. So, that way so, it can it can get the value from there.

(Refer Slide Time: 12:15)

Acc Register

- A register can be accessed by **direct** and **register** mode
- This 3 instruction has **same** function with **different** code

0703	E500	mov a,00h
0705	8500E0	mov acc,00h
0708	8500E0	mov 0e0h,00h
- Also this 3 instruction

070B	E9	mov a,r1
070C	89E0	mov acc,r1
070E	89E0	mov 0e0h,r1

IIT KHARAGPUR
 NPTEL ONLINE CERTIFICATION COURSES

Then the A register it can be accessed by direct and register mode. So, both direct and register mode can be used; so, these three instruction has same function with different type of code like. So, the address of this A register is E 5; so, we can have we can have this mov a comma 0 0. So, that is coded as E 5 0 0; then mov a cc comma 0 0 h. So, that is also that is coded in a different format 8500E0.

And similarly so, this is mov 0e0 comma 0 h. So, this is also actually that accumulator also so e 0 is the address of the A register. So, e 0 being the address of a register; so, the all these are same. So, you can. So, this is this instruction and this instruction they are

same and this is semantically same with this instruction though the size of the instruction here is 2 byte and here there are three bytes, but semantically they are same.

Similarly these three instructions they; mov c comma r1; then mov accumulator comma r1 and mov 0e0h comma r1; so, they are all same ok. So, e 0 as we said that e 0 is the address of accumulator register. So, there is that way it remains; it is same as that one. So, all these instructions they have got the same meaning.

(Refer Slide Time: 13:40)

The slide is titled "SFRs Address" in red text. It contains three bullet points. The first bullet point is "B – always direct mode - except in MUL & DIV". Below it are four lines of assembly code: "0703 8500F0 mov b,00h", "0706 8500F0 mov 0F0h,00h", "0709 8CF0 mov b,r4", and "070B 8CF0 mov 0f0h,r4". The second bullet point is "P0~P3 – are direct address". Below it are three lines of assembly code: "0704 F580 mov p0,a", "0706 F580 mov 80h,a", and "0708 859080 mov p0,p1". The third bullet point is "Also other SFRs (pcon, tmod, psw,...)". At the bottom of the slide, there are logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES. A small video inset of a man is visible in the bottom right corner of the slide.

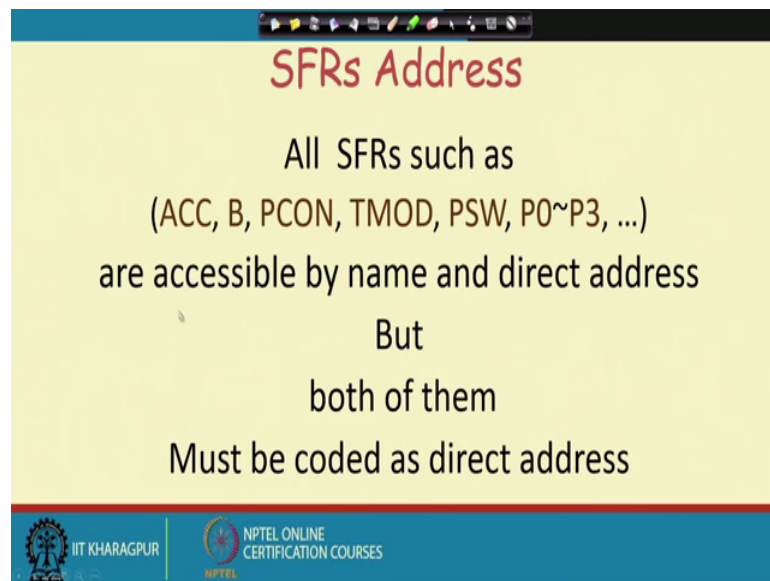
Some other special function registered like the B register. So, it is always it can be used in direct mode accepting in the multiplication and division operation. These two instruction it can be used separately; so, this mov b comma 0 0 h. So, this is the code 8500F0; then mov 0f0h 00. So, this is also the same is the same instruction same meaning, but they can have different formats ok. So, all these will be same b is coded as f 0.

So, that f 0 comes here; so, here also the same thing so, two instructions having the same meaning then P0 to P3. So, they can be used as direct address like mov p 0 comma a. So, this is ah; so, we can have this p 0 register is this 8 0; p 0 registers the address is 80 h. So, this is F580; then we can have this instruction which will be also F580, but here h 0 is mentioned explicitly.

Then we can also have `mov p 0 comma p 1`; like say, so this is p 0 is 80 and p 1 is 9 0. So, `mov p 0 comma p 1`; so, the this is this becomes a 3 byte instruction, we can have something like this. So, these are this P0 to P3. So, we can use them as direct addresses.

Then other SFRs like PCON, TMOD, PSW that they can also be used as used for this movements.

(Refer Slide Time: 15:08)



SFRs Address

All SFRs such as
(ACC, B, PCON, TMOD, PSW, P0~P3, ...)
are accessible by name and direct address
But
both of them
Must be coded as direct address

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, all special function registers like accumulator B, PCON etcetera; they are accessible by name and direct axis, but both of them must be coded in as direct address. So, while writing in the assembly language program; so, we are we are writing them as name or direct address, but while the machine code is generated. So, they are coded as direct addresses only; that is obvious.

(Refer Slide Time: 15:38)

The slide is titled "8051 Instruction Format" and is divided into two sections. The first section, "immediate addressing", shows the instruction "add a,#3dh" with a box labeled "Op code" containing "add a" and a box labeled "Immediate data" containing "#3dh". Below this, it states ";machine code=243d". The second section, "Direct addressing", shows the instruction "mov r3,0E8h" with a box labeled "Op code" containing "mov r3" and a box labeled "Direct address" containing "0E8h". Below this, it states ";machine code=ABE8". At the bottom of the slide, there are logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with a small video feed of a presenter.

Next we look into the immediate addressing like say mov say add a comma 3 d hash 3 d h; that means, with A register; we want to add the we want to add the value 3 d h. So, this instruction is a 2 byte instruction where the first byte is the Op code and Op code for add a; is the 24. So, that is the 24 then they add a add a immediate add a immediate is 24 and the immediate value will be for our preceding you will be following the Op code.

So, that is the 3 d. So, this 3 d is the immediate data then in the direct addressing. So, you can have like MOV R 3 comma 0E8h so; that means, the content of memory location 0E8h will come to the register R3. So, this is the; this Op code is A B and the address is E A and the address the if the value the address value is 8. So, that will come to this second byte. So, we can have this way the direct addressing.

(Refer Slide Time: 16:44)

8051 Instruction Format

- Register addressing

Op code	n	n	n
070D E8	mov a, r0		;E8 = 1110 1000
070E E9	mov a, r1		;E9 = 1110 1001
070F EA	mov a, r2		;EA = 1110 1010
0710 ED	mov a, r5		;ED = 1110 1101
0711 EF	mov a, r7		;Ef = 1110 1111
0712 2F	add a, r7		
0713 F8	mov r0, a		
0714 F9	mov r1, a		
0715 FA	mov r2, a		
0716 FD	mov r5, a		
0717 FD	mov r5, a		

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this is the instruction format like register addressing. So, this Op code will be there and there will be three bits which will identify the register. So, this mov a; so, all these instructions. So, these all these mov a type of instructions they have got the code like say E and then this. So, these 5 first 5 bits are fixed ok; so, this is for the mov mov a; the first 5 bits are fixed.

And then the next three bits will identify the register r0, r1, r2 up to r7. So, that way this op code coding is done similarly add a has got the code as again the 5 bits will come and this will turn out to be 2. So, this last configuration will come where these bits are actually 0010; this 4 bits will be 0010 and after that we will have 1111 because this is R R7. So, that way these 3 bits will correspond to 7 and this 00101. So, that will be the op code for add a.

So, this way we can have different coding for different instructions and they have got; so, that is, so, they have got the uniformity. So, this will be in this format.

(Refer Slide Time: 18:07)



8051 Instruction Format

- Register indirect addressing

Op code	i
---------	---

mov a, @Ri ; i = 0 or 1

070D E7	mov a, @r1
070D 93	movc a, @a+dptr
070E 83	movc a, @a+pc
070F E0	movx a, @dptr
0710 F0	movx @dptr, a
0711 F2	movx @r0, a
0712 E3	movx a, @r1

On the other end we if we have talking about this register indirect addressing then we know that only two registers are allowed like r0 and r1 ok. So, we can have i equal to 0 or i equal to 1. So, the 1 bit is sufficient at the end to identify that this is r0 or r1. So, when you say that mov a comma at the rate R1. So, that is coded as E 7 then similarly mov c a comma at the rate a plus dptr. So, this will be coded as 93, then this mov c a comma at the rate a plus PC. So, that will be coded as 83; so, this way this will be done mov x at the rate DPTR comma a. So, this will be accessing this thing.

(Refer Slide Time: 18:56)

8051 Instruction Format

relative addressing

Op code	Relative address
---------	------------------

here: sjmp here ; machine code = 80FE (FE = -2)
Range = (-128 ~ 127)

1000 1001

OP (80) REL (FE)

Absolute addressing (limited in 2k current mem block)

A10-A8	Op code	A7-A0
0700	1	org 0700h
0700 E106	2	ajmp next ; next=706h
0702 00	3	nop
0703 00	4	nop
0704 00	5	nop
0705 00	6	nop
	7	next:
	8	end



07FEh

1000 1001

1001 1002

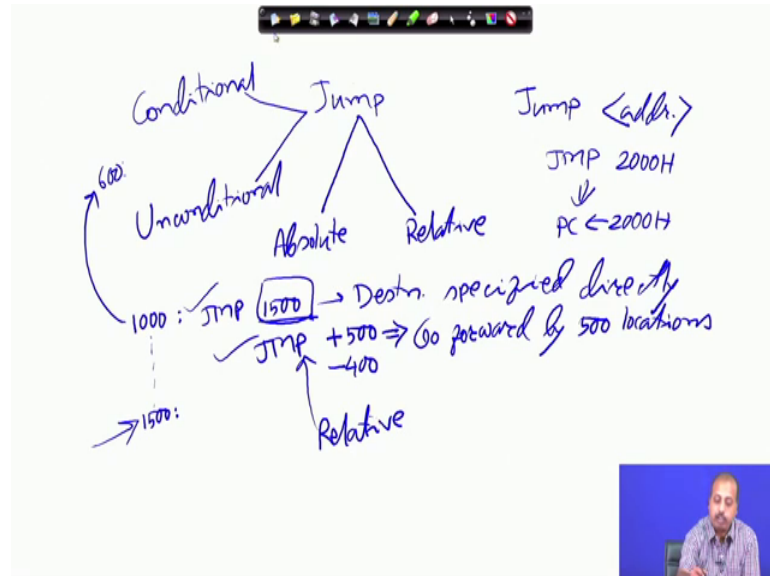
1002

+2

Then we have got the relative addressing. So, the relative addressing is one category of this comes in the context of jump instruction.

(Refer Slide Time: 19:17)



So, so if you if we see this jump or this JMP instruction; in general so, with all these jump operations that we have. So, in; if you look into any processor the instructions are like some op code for jump and then the address to where you want to jump. So, typical example say in a 0 8 5 we have sense jump 2000 H. So, this is what it does it loads the program counter with the value 2000.

So, that the next instruction executed is the instruction stored at location 2000 so, this jump we can have different type of classification. One classification is from the functionality which is conditional versus unconditional; conditional jump versus unconditional jump. So, this we have seen like we have got jump on (Refer Time: 20:18) jump on 0, jump on not 0. So, they are conditional jumps and we have got unconditional jumps like the JMP instruction.

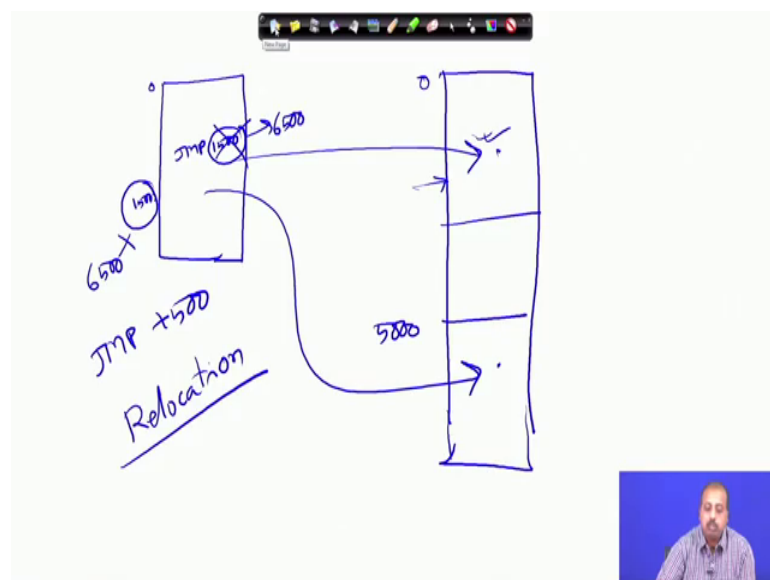
Another classification of this jump can be like this absolute jump and relative jump ; absolute jump and relative jump. So, what is it? Like say suppose at present we are executing; we are at location memory location 1000 and there I am putting; from here I want to go to the location 1500; put a jump instruction there. So, that the next instruction executed is the instruction at location 1500; now this is fine.

So, execution will be fine. So, here what is happening is I am telling specifically that what is the destination address; destination address is specified directly. So, you can say that destination specified directly. Another way of doing the same thing is to tell that it is jump by plus 500 locations. So, something like that; so, at present I am at 1000. So, if I say plus 500; so, with respect to current location, you go forward by 500 locations. So, this means that go forward by 500 locations ok. So, when I go forward by 500 location; so, after executing this one or this one the effect is same.

So, it comes to this address only, but the way we are telling it is different. Similarly you can say jump minus 400 so, that you go back to location 600 in the execution. So, that way we can do both forward and backward jumps and. So, this type of jumps so, they are known as relative jumps. So, these; so, these are actually the relative jumps that we talk about. Because these jumps the target address is specified relative to the current address; relative to the current address we are telling where do you want to jump.

Now, why all these things? Why this relative jump is; so, coming into picture why is it so, may be so, important or at all useful.

(Refer Slide Time: 23:01)



So, it is like this suppose I have written a program I have written a program and this program has got at it is it is ah; there is an instruction like jump 1500 here or 1500 is somewhere here. And this program is starting at location 0; now if this is my ultimately

the computers physical memory and if I assume that the program is loaded from memory location 0.

If I assume that the program will be loaded from memory location 0; then this jump is fine. So, this is loaded here; so, when it comes to this particular instruction. So, it executes; so, jump 1500; so, it goes to location 1500. But next time instead of the next time you run the program, instead of being loaded from location 0. Suppose the program is loaded from location 5000 onwards; the program is loaded from location 5000 onwards then what happens is that this jump or.

So, this 1500 address; so, this is no more 1500; this is basically 6500. So, again I have to make this correction in this program. So, this should be jump 6500; so, if you are using absolute jump then every time the program is loaded from a different address, this jump addresses they are to be corrected ok.

So, but if it is relative jump; so, that is not required because here is I am not telling jump 1500 what I am telling is jumped by plus 500; that means, with respect to the current location. So, you jump forward by 500 locations and that is true here as well as it is true here. So, there is no difference will be there even if the program is loaded from a different address compared to the 0 address.

So, this helps in something called program relocation this relative jump. So, this helps in the process called program relocation; that is the program may be loaded from a different address; next time it is loaded into the memory. So, that is how this relative jumps are going to be useful.

So, these processors they have got this relative jumps introduced so, that we can we can do this relocation jobs easily. Like when you are thinking about any complex system, then we have to have these many user programs loaded into memory. And it is not possible to ensure that always the program will be loaded from the same address.

So, this relative jump will be like this; so, this is say it is like it is an 11 bit sorry; this is an the sjmp instruction; sjmp and then the address where you want to jump. So, this sjmp is structure is op code followed by this relative address and this relative address is in the range of minus 128 to plus 127. So, you can jump by within 256 locations from the current location.

So, you can go back maximum 128 bytes and you can come forward at most 127 bytes. So, if you have the machine code is 80 for this sjmp and then this upward this relative address may be minus 2. So, minus 2 is coded as FE; so, this if here is. So, actually if you look into this instruction the coding of this sjmp here; so, this sjmp here instruction. So, this is a 2 byte instruction so, if the address if this instruction is at location 1000, then this location 1000 will have the code for is the op code part. So, this is the op code part and the location 1001 will have the this relative address part; the second part.

So, first one is this one, second byte is this one. So, the next instruction; so, if you look into the memory at location 1000 you have got the first byte of op; at the location 1001 you have got the second byte this value. Now after that the PC value becomes 1002; now from; so, after when this instruction has been faced by the processor; so, the PC value is equal to 1002. So, from there we want to go back to sjmp here and here is 1000. So, from 1002 we want to go back to 1000; so, we need to go back by two locations.

So, this relative address becomes equal to minus 2. So, while coding it; so, the first byte will be the code for this sjmp 80 and the second byte will be minus 2. So, that when this instruction is executed with the current PC value is 1002 with that this minus 2 will be added and it will be going to location 1000. So, there that is why; so, this is the coding of this sjmp here this instruction will be 80 FE and the FE value is equal to minus 2.

Similarly, we can have some other absolute addressing. So, we will see that absolute addressing there sjmp; there the ajmp and ljmp type of instructions that we will see later.