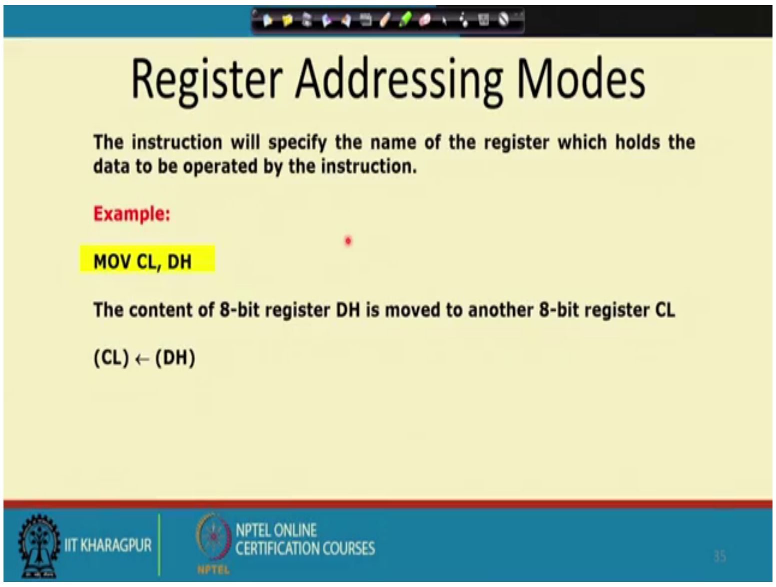
Microprocessors and Microcontrollers
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 62
8086 (Contd.)

The simplest of all these addressing modes is the register addressing mode.

(Refer Slide Time: 00:21)



Register Addressing Modes

The instruction will specify the name of the register which holds the data to be operated by the instruction.

Example:

MOV CL, DH

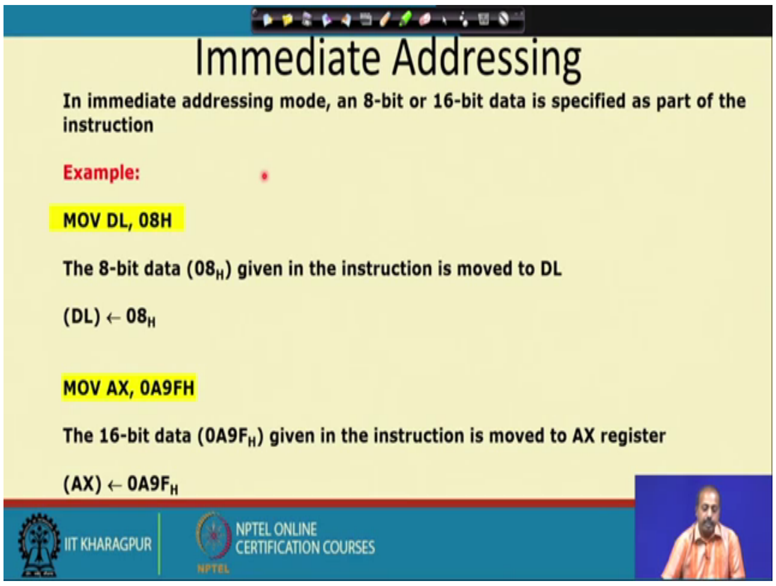
The content of 8-bit register DH is moved to another 8-bit register CL

(CL) ← (DH)

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
35

So, here the operands are registers. So, most of the operations that we have in 8086, they are two operand instruction. So, where we apart from the off code, we will have two operands on which through which the operation, we will take place. So, the instruction will specify the name of the register which holds the data to be operated on by the instruction like say MOV CL comma DH. So, content of this DH registered will be moved to CL register. So, both are 8 bit registers. So, there are no problem. So, CL gets DH. So, this is an example of this register addressing mode.

(Refer Slide Time: 00:56)



Immediate Addressing

In immediate addressing mode, an 8-bit or 16-bit data is specified as part of the instruction

Example:

MOV DL, 08H

The 8-bit data (08_H) given in the instruction is moved to DL

$(DL) \leftarrow 08_H$

MOV AX, 0A9FH

The 16-bit data ($0A9F_H$) given in the instruction is moved to AX register

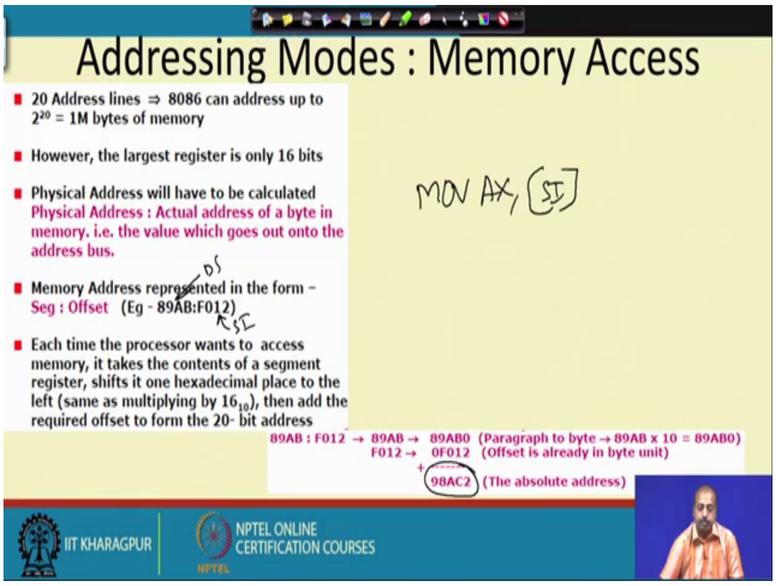
$(AX) \leftarrow 0A9F_H$

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Then we have got immediate addressing mode; so, in immediate addressing mode. So, the second operand that we have is an immediate value; the first operand have may be is a register; whereas, the second operand has to be a immediate value. So, this is immediate value can be 8 bit or it can be 16 bit that depends or so, there should be a match like if the specifying an 8 bit registered here, then this value should also be an 8 bit value, if you are mentioning a 16 bit register here and this value will be taken as a 16 bit value; so, this MOV DL comma 08 H.

So, this is 08 H will come to the DL register; that is a 8 bit value and MOV AX comma 0A9F H. So, this is a 0A9F H will come to the AX register out of that this is 0 A part. So, this will go to the register and the 9 F part will go to the AL register. So, that way, this 16 bit value will be divided into higher order byte and lower order byte and they will be going to different registers.

(Refer Slide Time: 02:02)



Addressing Modes : Memory Access

- 20 Address lines ⇒ 8086 can address up to 2^{20} = 1M bytes of memory
- However, the largest register is only 16 bits
- Physical Address will have to be calculated
  Physical Address : Actual address of a byte in memory. i.e. the value which goes out onto the address bus.
- Memory Address represented in the form – Seg : Offset (Eg - 89AB:F012)
- Each time the processor wants to access memory, it takes the contents of a segment register, shifts it one hexadecimal place to the left (same as multiplying by 16_{10}), then add the required offset to form the 20- bit address

```
89AB : F012 → 89AB →  89AB0  (Paragraph to byte → 89AB x 10 = 89AB0)
              F012 →  0F012  (Offset is already in byte unit)
                    + -----
                      98AC2  (The absolute address)
```

MOV AX, [SI]

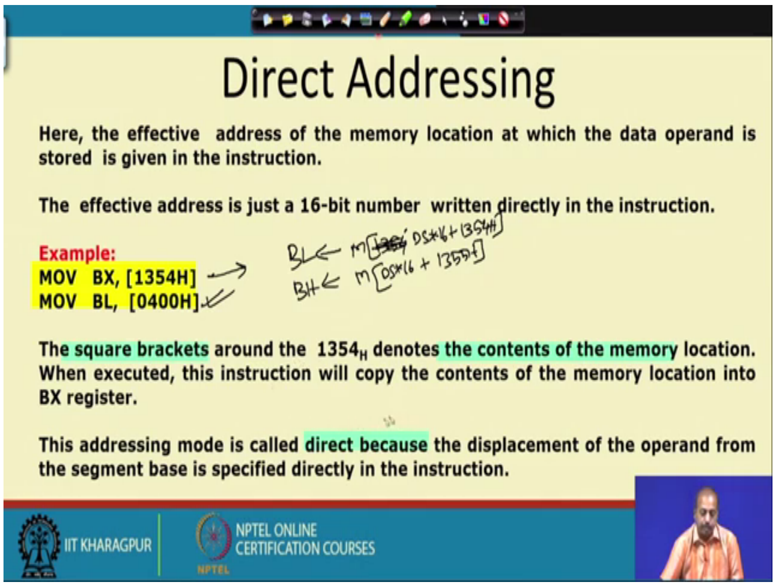
So, for memory access; so, you see that 8086 has got 20 address lines. So, these address lines are they can address up to 2 power 20 equal to 1 megabytes of memory; however, the largest register is only 16 bit; so to get this physical address. So, this actual address is calculated the actual address of a byte in memory that is the value of that goes out on the address bus that has to be calculated and that is the physical address.

So, how is this physical address represented, this is represented in the form of a segment value and then offset value; so, 89AB colon F012. So, what is what will happen is that this is 8 9 AB. So, this will be left shifted by 4 bits. So, that way, in a extra decimal forms if you write. So, as if I a 0 will appear at the end. So, it will become a twenty bit value. So, this is the 20 bit value each digit corresponding to 5 4 bits the 5digits. So, there are 20 bit and this F012. So, this is the offset part.

So, F012 is taken as this is also a 16 bit values. So, we take it as a 20 bit value. So, most significant digit in hexadecimal notation becomes 0 and then you do the addition after that you get a 20 bit result. So, that will be accessed that will be the physical address. So, physical address is the address the actual address of the bytes for the physical memory. So, it will be doing this memory access will be done by doing this operation; so this 89AB F012. So, these quantities can come from different registers like if you are writing like if you are using say the SI register.

So, SI register contains F012 and DS register contains 89AB, then when you are saying that MOV AX comma within. So, if you are writing like say MOV AX comma within bracket SI then we assume that this 89AB. So, this is in the DS register and this is in the SI register. So, this is what will happen internally and then the on the address bus the address that will be put is 98AC2 H. So, that way the operation takes place.

(Refer Slide Time: 04:24)



Direct Addressing

Here, the effective address of the memory location at which the data operand is stored is given in the instruction.

The effective address is just a 16-bit number written directly in the instruction.

Example:
MOV BX, [1354H]
MOV BL, [0400H]

$BL \leftarrow M[~~1354~~\ DS*16+1354H]$
$BH \leftarrow M[DS*16 + 1355H]$

The square brackets around the 1354_H denotes the contents of the memory location. When executed, this instruction will copy the contents of the memory location into BX register.

This addressing mode is called direct because the displacement of the operand from the segment base is specified directly in the instruction.

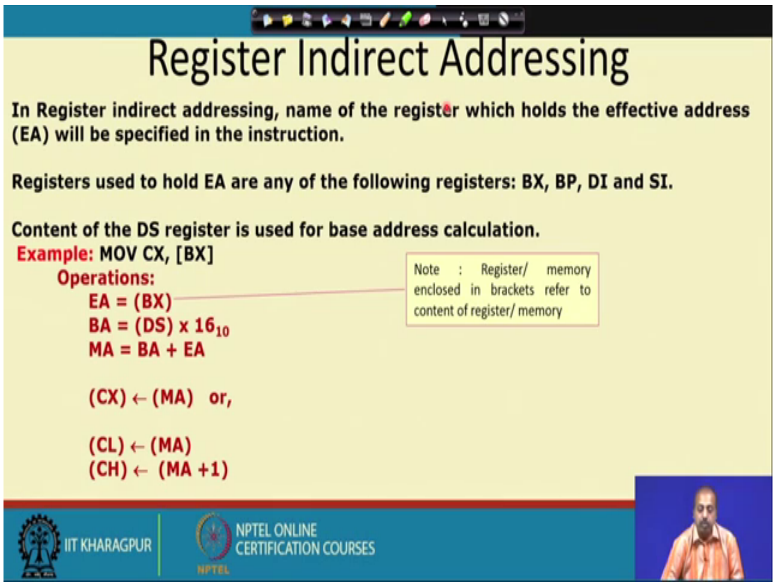
Next we will look into the different addressing modes, first of all, first one is the direct addressing. So, in case of direct addressing the address is directly provided in the instruction itself. So, the address of the memory location is given in the instruction itself the effective address is a 16 bit number.

So, this is the 16 bit number that we have. So, 1354 h ok. So, this is the 16 bit number or 0400H. So, that is the 16 bit number. So, this square bracket that we have this square bracket. So, they will identify that we are interested in the contents of those location. So, if these are not there if these are not there. That means, we will tell that that is actually the immediate addressing ok. So, that is the convention followed during a 8086 assembly language. So, if these square brackets are missed are missing, then that will mean and mean and immediate data if this square brackets are there, it will mean that it is an address.

So, it will be accessing the corresponding memory location. So, when executed this instruction will copy the content of memory location into the BX register and then. So, in this case, in the first instruction, what will happen is that this BL register will get the content of memory location 1354 h. So, that will come to so, 1354H. So, it that DS colon should be there sorry, this is not 1354, I cannot say exactly, what is the location content, but that is DS into 16 plus 1354.

So, that location content will come to BL and the BH register will get may content of memory location DS into 16 plus 1355H, the next memory location content, we will come to the BH register because this the 16 bit data movement whereas, this instruction. So, this is one bite data movement. So, BL register will get the content of memory location 16 into DS plus 0400. So, this is called direct mode because the displacement of the operand from the segment base is specified directly in the instruction itself. So, that is the direct mode of the operation.

(Refer Slide Time: 06:50)



Register Indirect Addressing

In Register indirect addressing, name of the register which holds the effective address (EA) will be specified in the instruction.

Registers used to hold EA are any of the following registers: BX, BP, DI and SI.

Content of the DS register is used for base address calculation.

Example: MOV CX, [BX]

Operations:

EA = (BX)

BA = (DS) x 16_{10}

MA = BA + EA

(CX) ← (MA) or,

(CL) ← (MA)

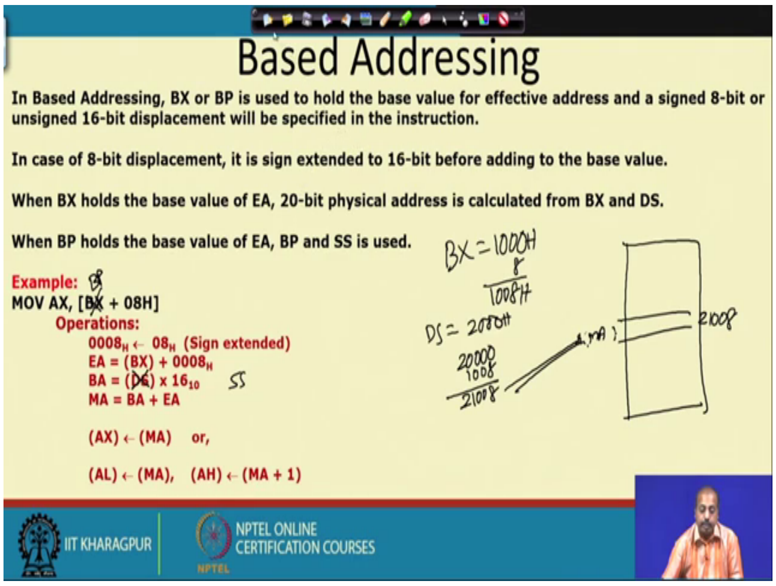
(CH) ← (MA +1)

Note : Register/ memory enclosed in brackets refer to content of register/ memory

Next, we will look into something called register indirect mode. So, register will be used to give the address whose content will be accessed by the those content will be accessed by the processor; so in register indirect mode. So, name of the register which holds the effective address is specified in the instruction itself like we are writing like MOV CX comma within square bracket bx. So, this effective address is the content of the BX register. So, this BX register whatever it is holding. So, that will be the effective address and the base address is DS multiplied by 16 and the actual memory address are the physical address that is generated is this base address plus this effective address. So, they will be summed up and we will get the actual memory address.

So, this way it will be operating and ultimately the CX register will get the content of this memory location whose address is given by this MA memory address or I can say that CL will get the content of MA and CH will get the content of MA plus 1. So, this is how this register indirect mode is working. So, this BX register is used as the indirect as the base register as the offset register that can be used for accessing the memory location.

(Refer Slide Time: 08:17)



Based Addressing

In Based Addressing, BX or BP is used to hold the base value for effective address and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction.

In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value.

When BX holds the base value of EA, 20-bit physical address is calculated from BX and DS.

When BP holds the base value of EA, BP and SS is used.

Example: BP

MOV AX, [~~BX~~ + 08H]

Operations:

$0008_H \leftarrow 08_H$ (Sign extended)

$EA = (BX) + 0008_H$

$BA = (\cancel{DS}) \times 16_{10}$ SS

$MA = BA + EA$

$(AX) \leftarrow (MA)$ or,

$(AL) \leftarrow (MA)$, $(AH) \leftarrow (MA + 1)$

Then in base addressing; so this is the this is the another type of addressing which is similar to your indirect addressing, but this DX or DP register, they can be used for holding the base value and the effective address and then sign 88 bit signed or unsigned value or 16 bit displacement that will be specified in the instruction. So, the signed 8 bit or unsigned 16 bit displacement can be specified.

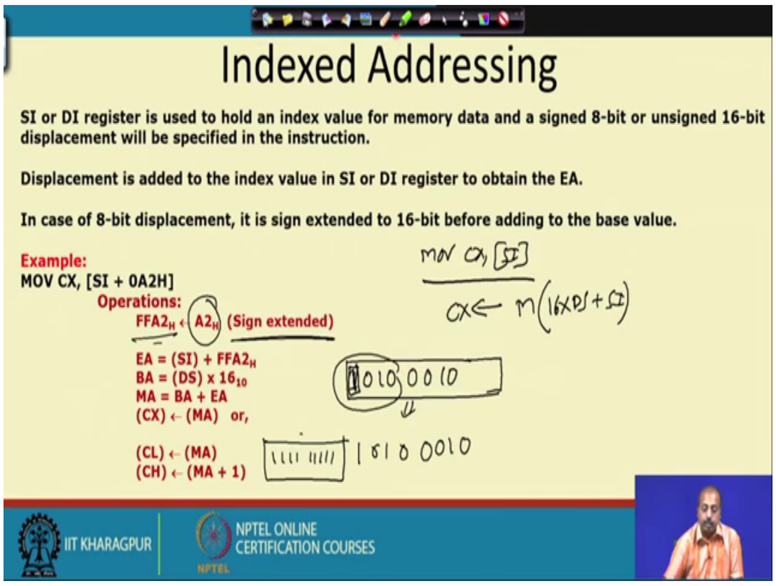
So, this is a typical example like MOV AX BX plus 08H. So, along with this base registered BX we are also specifying we can also specify an offset. So, in the offset is not there then of course, this BX will be used as the based register and then this will be accessing AX plus a, this DS into 16 as BX, but if you are specifying some offset x another offset some displacement. So, that displacement will also get added. So, first of all this 08H, this is an 8 bit number.

So, this is extended to 16 bit sign extended to 16 bit. So, get 0008H and then this ea gets BX plus 008H. So, this effective address that we get; so, that is BX plus this is value if the content of BX is a 1000. So, after this addition the EA the effective address will become 1008 and then this based address is DS in to 16 and the memory address that is generated is this base address plus this, this, this base address plus this effective address that we have. So, then this MA register sorry this AX register will get the content of memory location whose address is this memory address MA and or you can say that the AL gets the content of MA and MA h gets the content of MA plus 1.

So, so if say BX is say equal to 1000H then with this thousand this 8 is added. So, this becomes 1008 x and then it depends on the DS register. So, if DS location content is say 2000, then this 2000H. So, this 2000 is left shifted by 1 B of 4 bits. So, it becomes this one with that this 1008 will be added. So, it is 21008. So, that particular memory location will be accessed. So, this is that memory location 21008. So, this location will be accessed and this MA equal to 21008; so this MA.

That is the memory address. So, that co locations content will be accessed and that will be going to the EX register AX register. So, that locations content will come to the AX register. So, this way we can have the based addressing mode and you see instead of BX you can also use BP, but if you use BP, then this stag segment register will be used. So, here instead of this BX, if I use BP, then the modification that we will have is instead this one instead of being DS. So, it will be SS the stack segment register and otherwise it is same otherwise the execution is same then we will look into the next addressing mode which is known as the indexed addressing mode.

(Refer Slide Time: 11:54)



Indexed Addressing

SI or DI register is used to hold an index value for memory data and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction.

Displacement is added to the index value in SI or DI register to obtain the EA.

In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value.

Example:
MOV CX, [SI + 0A2H]
Operations:
$FFA2_H \leftarrow A2_H$ (Sign extended)
$EA = (SI) + FFA2_H$
$BA = (DS) \times 16_{10}$
$MA = BA + EA$
$(CX) \leftarrow (MA)$ or,
$(CL) \leftarrow (MA)$
$(CH) \leftarrow (MA + 1)$

So, indexed addressing mode it uses some index registered to ho to access the memory. So, this SI or DI registered is used to hold an indexed for memory data and signed 8 or 6 unsigned 16 bit displacement can also be specified. So, you can say something like this. So, the basic form is like so, where the displacement is not there. So, we the basic form is like say MOV CX comma within bracket SI. So, you see that all this mode. So, they are some sort of modifications of this immediate sorry the indirect addressing mode, but they have been even different names because they have got the specific usage this source index registered, they are they are thought to be working as indexed register and they are useful when we are doing this string data transfers. So, that is why they have been put in to separate mode this SI and DI.

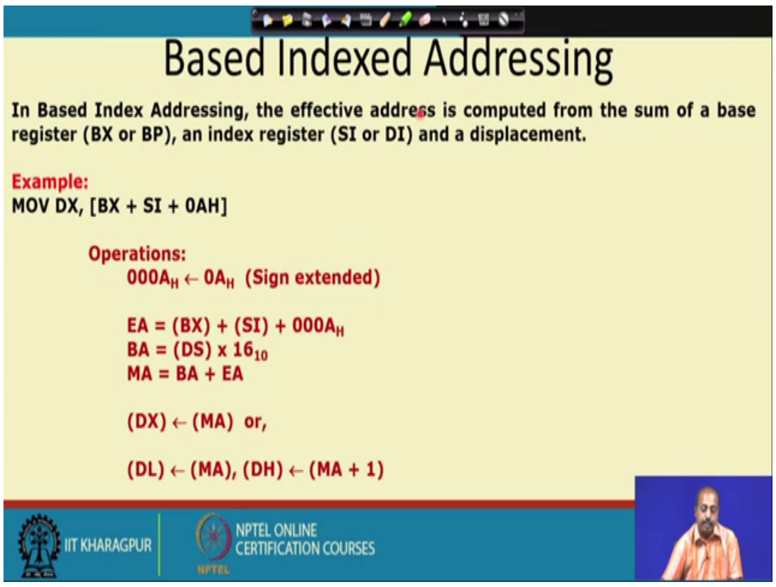
So, in this particular case what will happen you can understand that CX register will get the content of memory location, whose address is given by 16 into DS plus SI and in this example, we have got some additional, we have got some additional offset specified additional displacement specified. So, that displacement will be added. So, this so, this is the sign extended. So, you see that this particular number A 2. So, A 2 means that if you if you take that 8 bit notation then a means 1 0 1 0 10 and 2 is 0 0 1 0.

So, in that 8 bit notation this most significant bit is one most significant bit is one means this is the negative number. So, first before doing the addition the SI plus 082H; so this is extended to 16 bit. So, for 16 bit extension what is done this most? So, this bit is made

0010 the 2 part and this part is 0 1 0 0010 1 that is all right, but since this is negative; so this 8 8 bit that we have. So, that is made negative ok. So, all these this sign extend; so this or the sign extension. So, all these bits will become one. So, that it represents the same number A 2 x in 16 bit notation. So, that is the sign extended form of this.

So, these become FFA2 H and with that the FFA2 H will be added with the SI register for getting the offset and then this memory address will be calculated. Next, we have this based indexed mode so based indexed mode mean.

(Refer Slide Time: 14:54)



Based Indexed Addressing

In Based Index Addressing, the effective address is computed from the sum of a base register (BX or BP), an index register (SI or DI) and a displacement.

Example:
MOV DX, [BX + SI + 0AH]

Operations:
$000A_H \leftarrow 0A_H$ (Sign extended)

$EA = (BX) + (SI) + 000A_H$
$BA = (DS) \times 16_{10}$
$MA = BA + EA$

$(DX) \leftarrow (MA)$ or,

$(DL) \leftarrow (MA), (DH) \leftarrow (MA + 1)$

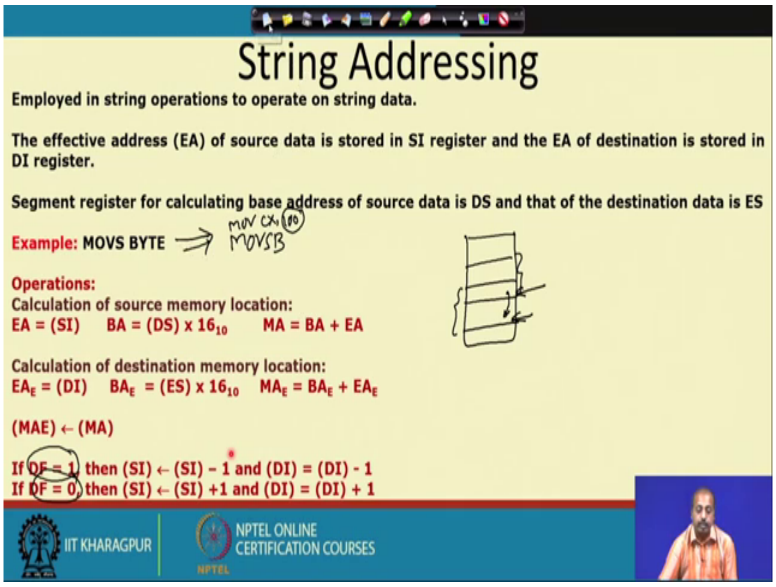
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

So, we have got this based register plus this indexed register. So, they are taken together. So, many times what is what will happen is that you may you may need to have 2 or more indices particularly if you are doing say in particular these the 2 dimensional arrays and all that then this may be useful that in maintain 2 different indices I and j for the 2 different dimensions and for that purpose.

So, we can use this type of double indexing register and double in direct addressing type of thing. So, the effective address is computed at some of the base register and indexed register. So, based register is DX plus BX or BP and index register is SI plus SI or DI; so here also the same thing that is 0AH. So, when you sign extent. So, it becomes 0 0. So, 0AH is a positive number. So, sign extended form is 16 bit sign extended is 000AH, then this effective address is BX plus SI plus this 000AH. So, it based address becomes based address is DS into 16 because these instructions since SI and BXI have been used.

So, it will be using DX as the segment register. So, it is DS into 16 and the memory address is this based address plus this effective address and if it is BP instead of BX if it is BP then you will be it will be using the SS register as the stack segment register as the segment register instead of the DS the data segment register next will look into string addressing.

(Refer Slide Time: 16:33)



String Addressing

Employed in string operations to operate on string data.

The effective address (EA) of source data is stored in SI register and the EA of destination is stored in DI register.

Segment register for calculating base address of source data is DS and that of the destination data is ES

Example: MOVS BYTE $\Rightarrow$ MOV CX, $\textcircled{100}$ MOVSB

Operations:
Calculation of source memory location:
$EA = (SI)$ $BA = (DS) \times 16_{10}$ $MA = BA + EA$

Calculation of destination memory location:
$EA_E = (DI)$ $BA_E = (ES) \times 16_{10}$ $MA_E = BA_E + EA_E$

$(MAE) \leftarrow (MA)$

If $DF = 1$, then $(SI) \leftarrow (SI) - 1$ and $(DI) = (DI) - 1$
If $DF = 0$, then $(SI) \leftarrow (SI) + 1$ and $(DI) = (DI) + 1$

So, string addressing for doing the string operations. So, string is one of the very important data type you can say because it is used in many applications particularly for doing comparison when you have got some data from outside you also need to compare or many a times, for in high level programs, we have got lot of string manipulation instead operations to be taken to be done and if the underlined processor supports this string manipulation directly, then in the software you do not need to do much complex programs for supporting this string. So, overall the string operations will be made faster.

So, this was the objective with which this string addressing was introduced. So, this is employed in string operation to operate on string data the effective address for the source data is stored in SI register and for the destination it is stored in the DI register and the segment register for the source part it is DS and for the destination part it is ES. So, if typical example is this MOVS instruction the MOVSB so on MOVS Byte.

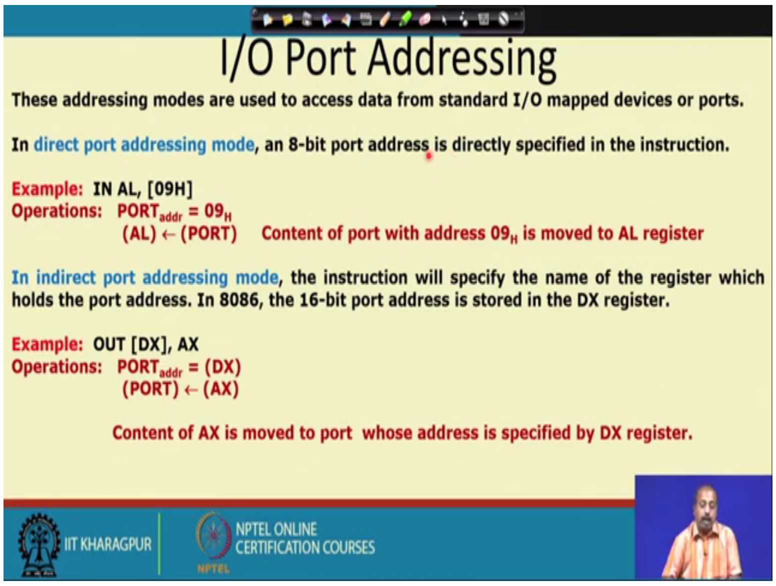
So, you can write like byte or you can write like word etcetera. So, or in short you can also simply write it as MOVSB. So, this can also be written in short as MOVSB or you can write in full form like MOVS Byte. So, how is it done like this effective address the. So, for the source is available in the source index registered SI then this base address is DS DS register will contain the segment address. So, base address is DS into 16.

So, the memory address is BA plus EA. So, based address plus effective address for the destination part then for the effective address will be calculated as DI and this for the destination part this base address is from the extra segment register ES. So, ES into 16 that will give us the base address for the destination part and the memory address is BA plus ES. So, so this is this extra segment effective address and extra segment base address. So, they will be added to get these MAE; the memory address for X EX extra segment or for the destination and finally, this MAE will get the memory location MAE will get content of MA and after that if DF flag is set then SI will be decremented and DI will be decremented and if DF is 0 then SI will be incremented and DI will be decremented.

So, as I was telling that if you are moving from source block to the destination block and if you have got if there is overlapping. So, if you are overlapping is such that the overlapping is. So, this is the overlapping. So, this is the source and this is the destination then what you need to do is that this SI value SI value should point to the bottom of the source and DI should point to the bottom of the destination and from there we should start copying. So, in these I have to decrement the values of SI and DI after each byte transfer. So, I have to set this DF value to be equal to 1, in that case and otherwise, if the if the overlapping is in the other direction in that case, I have to set this DF value equal to 0. So, that this SI and DI values will be incremented after fd transfer and we can have this CS register initialize to the some count.

So, to see how to have how many bytes you want to transfer. So, before calling moves b; so if I say like MOV CX comma hundred; that means, this CX register will have the value hundred. So, in that case, 100 bytes of data will be transferred will be copied from source to destination and after every copy this SI and DI registers will be updated to point to the next byte and this CX register value will be decremented to see that whether hundred byte transfer is over or not. So, when it is done then it will come out of this MOVSB instruction.

(Refer Slide Time: 20:58)



I/O Port Addressing

These addressing modes are used to access data from standard I/O mapped devices or ports.

In direct port addressing mode, an 8-bit port address is directly specified in the instruction.

Example: IN AL, [09H]
Operations: $PORT_{addr} = 09_H$
$(AL) \leftarrow (PORT)$ Content of port with address 09_H is moved to AL register

In indirect port addressing mode, the instruction will specify the name of the register which holds the port address. In 8086, the 16-bit port address is stored in the DX register.

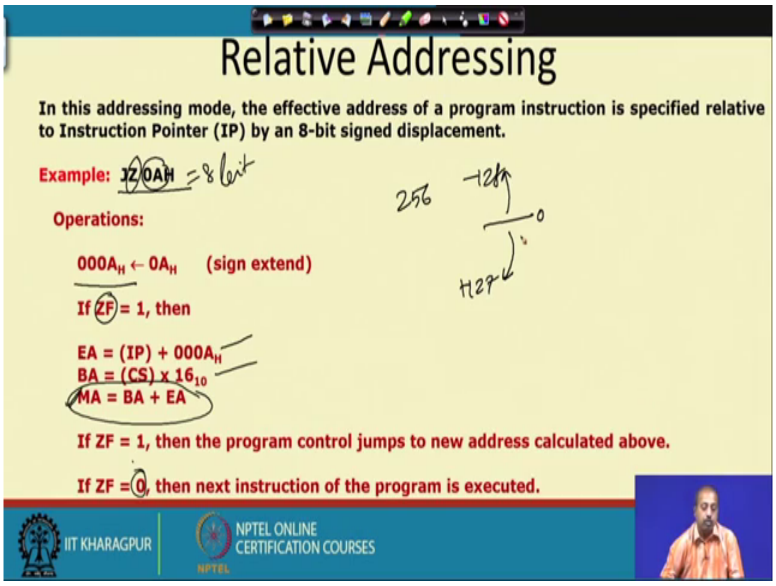
Example: OUT [DX], AX
Operations: $PORT_{addr} = (DX)$
$(PORT) \leftarrow (AX)$

Content of AX is moved to port whose address is specified by DX register.

So, next will look into IO port addressing; so there are these addressing modes can be used for to access data from standard IO map devices or port. So, IO map IO can be used. So, where the port address is the IO port address and in or we can have memory mapped IO where this memory value locations are the IO ports are taken as memory locations only and simple MOV type of instructions can be used. So, in direct port addressing; so, we can have this 8 bit port address directly specified like this I can have in AL comma 09H. So, this port address is 09H. So, on the address box this 09H will be put and it is expected that the corresponding device will get selected and the value will be available on the data bus.

So, at the end; so this AL will get into the content of the port whose address is was 09H and in indirect port addressing. So, some register value will be used as the port address. So, we can specify a 16 bit port address and some registered value will have the address whatever be the content of this DX register. So, that is put on to this address bus and then this AX register will be content will be put on to the data bus. So, that this AX register content will go to the corresponding port; so, so this way, we can have this IO port addressing both direct IO that is having 8 bit address or indirect IO having 16 bit address bia some register.

(Refer Slide Time: 22:38)



Relative Addressing

In this addressing mode, the effective address of a program instruction is specified relative to Instruction Pointer (IP) by an 8-bit signed displacement.

Example: JZ 0AH $= 8$ bit

Operations:

$000A_H \leftarrow 0A_H$ (sign extend)

If ZF = 1, then

EA = (IP) + $000A_H$

BA = (CS) x 16_{10}

MA = BA + EA

If ZF = 1, then the program control jumps to new address calculated above.

If ZF = 0, then next instruction of the program is executed.

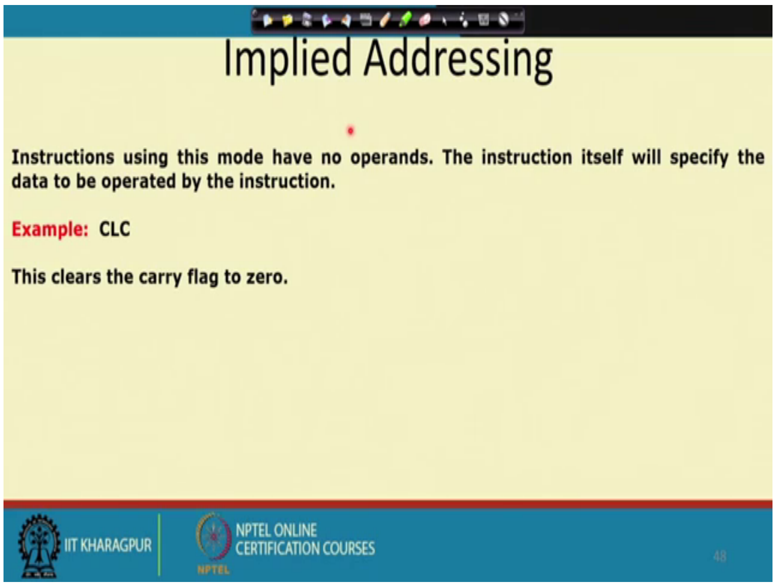
Next we will look into relative addressing. So, relative addressing means that with respect to the current program location. So, we are trying to update the instruction pointer to go to the next instruction. So, the effective address of the program instruction is specified relative to the instruction pointer by an 8 bit signed displacement. So, this is like this JZ 0AH. So, this will be this JZ 0AH. So, this will be doing like this first of all this 0AH. So, this will be sign extended. So, it will become a 16 bit value; now if the 0. So, if jump on 0. So, it will check the 0 flag, if 0 flag is equal to 1, then the effective address will be IP plus this 00 0AH, the extended value based address is CS into 16.

So, that is good for any code segment access and then the memory address will be this BA plus ea. So, that will be the finally, this will be the memory location, now the 0 if the 0 flag is one then the program control will jump to this particular address and if it is not if it is the 0 flag is 0, then as if the instruction is not there. So, it will be executing the next instruction that is coming in the sequence one thing to note here is that this displacement that is specified.

So, this is only 8 bit displacement; that means, you can you can do this relative jumping up to 256 locations minus 128 to plus 127 from the current location. So, if this is the current location. So, this offset is 0. So, from there you can go back by minus 128 locations or you can come forward by plus 127 locations. So, unlike say arm processor where it was plus minus 32 MB so that was a quite big number, but here it is restricted.

So, here it is for relative addressing. So, it is only 8 bit of course, you have got other type of branching in which you do not need to be restricted by this 8 bit, but relative addressing is 8 bit. Next, we will look into next we will look into another addressing mode which is implied addressing like some of the instructions where there is no operand operands are not needed the instruction itself will specify the data on which to operate say CLC. So, this will clear the carry flag to 0.

(Refer Slide Time: 24:54)



Implied Addressing

Instructions using this mode have no operands. The instruction itself will specify the data to be operated by the instruction.

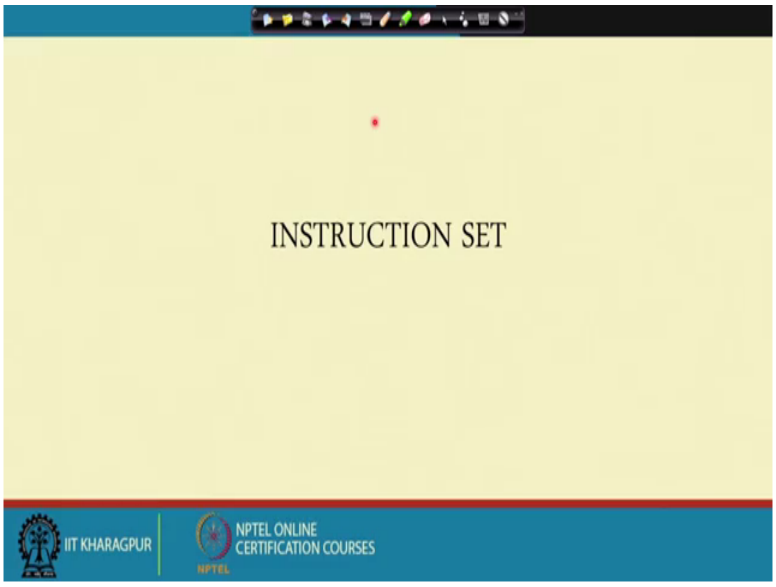
Example: CLC

This clears the carry flag to zero.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
48

So, no operand is needed or say halt or say knob; so this type of instructions where the operands are not required. So, they are it will be that is the implied addressing mode. So, you see that 8086 has got a large number of instra, the large number of addressing modes and this addressing modes are useful in different ways of course, actually these started this Intel they started with 8086 with lot having lots of addressing modes and later on when processors are being designed like when you look go to. For example, arm processor, we have seen that many of this addressing modes have been kept and many of them have been removed like when you are going to this risk feature then many of the complex addressing mode like based indexed and all that. So, they have been removed whereas this, whereas this is the indexed addressing indirect addressing. So, they have been kept. So, that way it goes.

(Refer Slide Time: 26:09)



INSTRUCTION SET

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

So, next we go to the instruction set of this 8086 processor and we will see that using this instructions the addressing modes this instruction sets can be designed by which we can program this 8086 processor to do several operations.