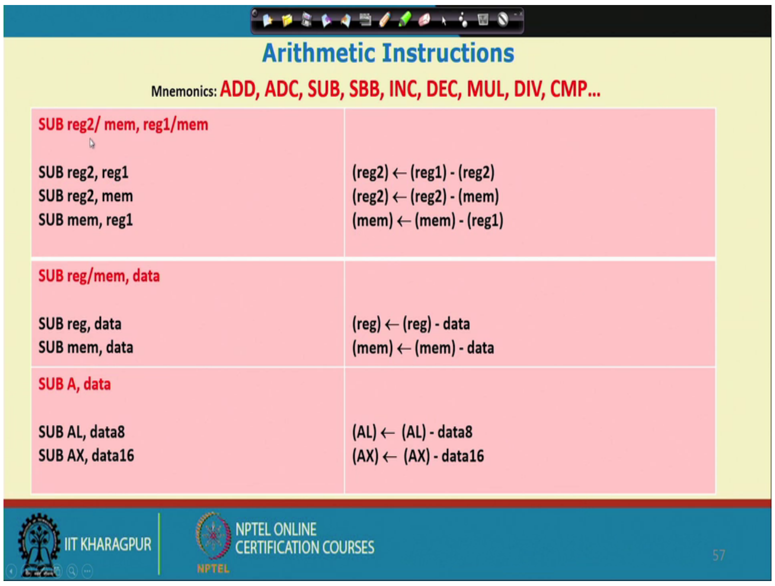
Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 65
8086 Microprocessor (Contd.)

(Refer Slide Time: 00:16)



Arithmetic Instructions

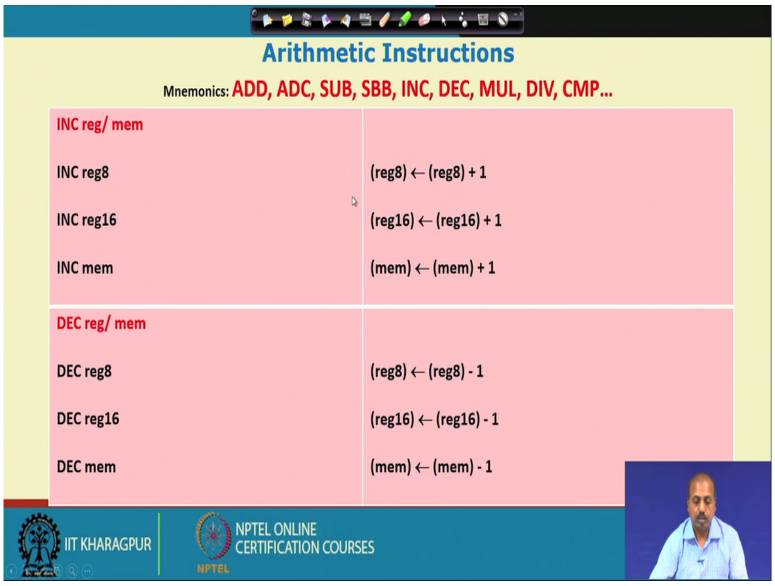
Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...

SUB reg2/ mem, reg1/mem	
SUB reg2, reg1 SUB reg2, mem SUB mem, reg1	(reg2) ← (reg1) - (reg2) (reg2) ← (reg2) - (mem) (mem) ← (mem) - (reg1)
SUB reg/mem, data	
SUB reg, data SUB mem, data	(reg) ← (reg) - data (mem) ← (mem) - data
SUB A, data	
SUB AL, data8 SUB AX, data16	(AL) ← (AL) - data8 (AX) ← (AX) - data16

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL
57

The subtract instruction so, we have got this sub normal subtract and the subtract with borrow so, that will come as SBB. So, subtract is similar as ADD. So, it is register 2 register 1 so, register 2 gets register 1 minus register 2. So, like that sorry this should be register 2 minus register 1. So, this is wrong this should be register 2 minus register 1, ok. Because that is the destination so, that comes first. And so, that way we have got the subtract operation. Next we have got the, subtract with borrow so; there here it is fine so, register 2 gets register 1 minus. So, this should also be register 2 minus register 1, this is register 2 minus 1 minus the carry flag.

(Refer Slide Time: 01:17)



Arithmetic Instructions

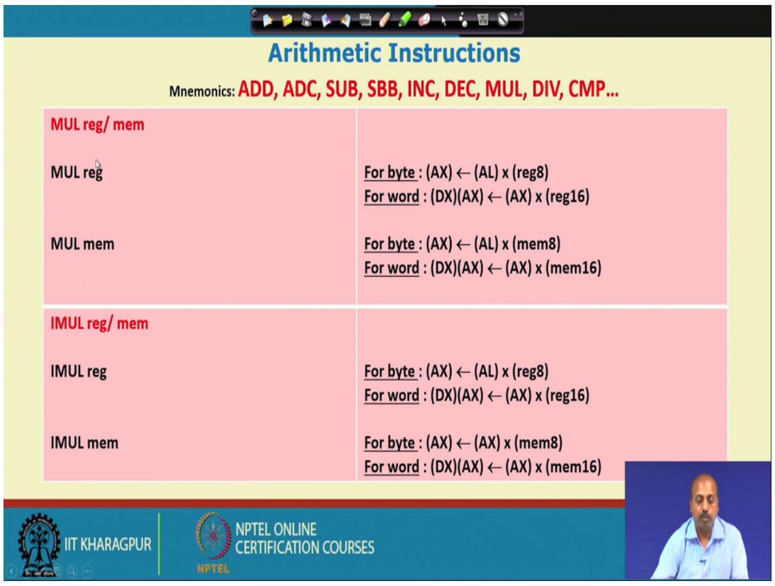
Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...

INC reg/ mem	
INC reg8	(reg8) ← (reg8) + 1
INC reg16	(reg16) ← (reg16) + 1
INC mem	(mem) ← (mem) + 1
DEC reg/ mem	
DEC reg8	(reg8) ← (reg8) - 1
DEC reg16	(reg16) ← (reg16) - 1
DEC mem	(mem) ← (mem) - 1

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So otherwise they are same, otherwise it is similar to the ADD operation, then increment operation. So, there is you can increment an 8-bit register, or you can increment 16-bit register or you can increment a memory location. So, this INC instructions similarly we have got decrement operation so, decrement a 8-bit register 16-bit register or memory location.

(Refer Slide Time: 01:34)



Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...

MUL reg/ mem	
MUL reg	For byte : (AX) ← (AL) x (reg8) For word : (DX)(AX) ← (AX) x (reg16)
MUL mem	For byte : (AX) ← (AL) x (mem8) For word : (DX)(AX) ← (AX) x (mem16)
IMUL reg/ mem	
IMUL reg	For byte : (AX) ← (AL) x (reg8) For word : (DX)(AX) ← (AX) x (reg16)
IMUL mem	For byte : (AX) ← (AX) x (mem8) For word : (DX)(AX) ← (AX) x (mem16)

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

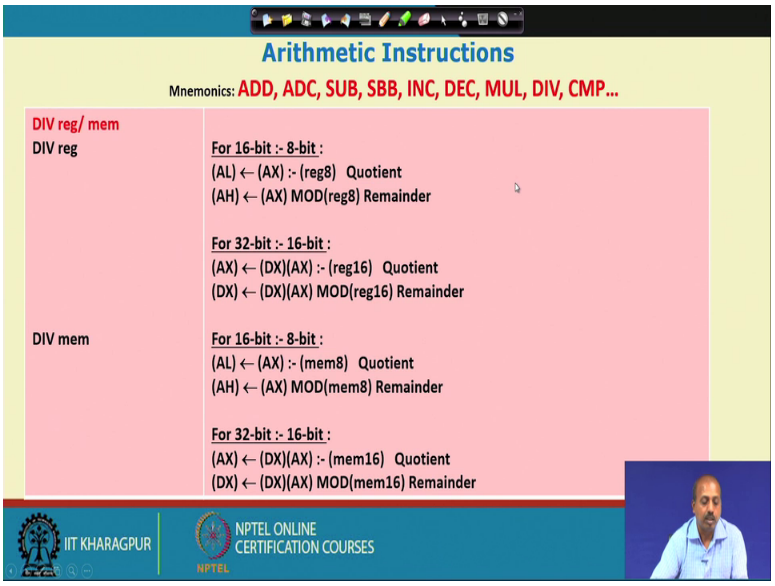
Multiply so, we have got a multiply and register can be specified. So, in this case, AX will get AL multiplied by register the 8-bit register. And for word operation DX colon

AX will get AX into register 16. Actually what happens is that if you multiply two 8 bit values so, you get a 16-bit value. So, in multiply operation one of the operand is always the AX or AL register. So, if you are doing 8-bit multiplication, then it is assumed that one of the operands is in AL registers, the other one is in the register that is mentioned here.

And the result is 16 bit and this AX register will hold that 16-bit value. On the other hand, if you are doing 16-bit multiplication, then whatever register you mentioned so, that will be it will be assumed that AX is the other source operand. So, the so, this is 16-bit register content will be multiplied by AX, and the result will be a 32-bit content, and that will be stored in DX and AX registers. Then the DX will hold the higher order 16 bit, and the AX will hold the lower order 16 bit.

So, similarly we have got this integer multiplication and a division operation. So, that way also it is similar to that multiplication division.

(Refer Slide Time: 02:54)



Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...

DIV reg/ mem	
DIV reg	For 16-bit :- 8-bit : (AL) ← (AX) :- (reg8) Quotient (AH) ← (AX) MOD(reg8) Remainder For 32-bit :- 16-bit : (AX) ← (DX)(AX) :- (reg16) Quotient (DX) ← (DX)(AX) MOD(reg16) Remainder
DIV mem	For 16-bit :- 8-bit : (AL) ← (AX) :- (mem8) Quotient (AH) ← (AX) MOD(mem8) Remainder For 32-bit :- 16-bit : (AX) ← (DX)(AX) :- (mem16) Quotient (DX) ← (DX)(AX) MOD(mem16) Remainder

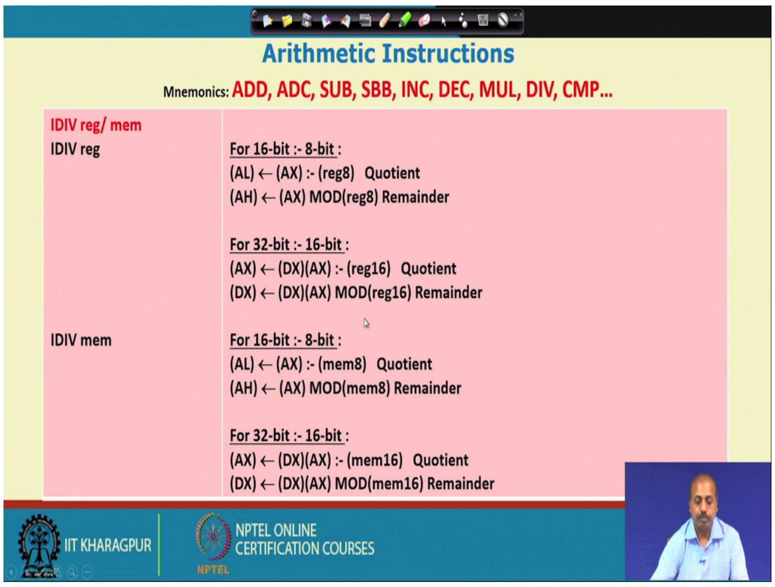
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then there is a div instructions so, they are the assumption is that AX divided by the register 8-bit register.

So, that will be done so, that will come to AL and will hold the reminder part so, quotient part will go to AL. So, AX is the dividend and register is the divisor and so, when this division is done quotient will be kept in the AL register and that reminder will go to the register, remainder will go to the register. For 32-bit operation DX AX pair so, it is a expected to hold the dividend, that will be divided by the 16-bit register the divisor kept in this 16-bit register.

And after division AX gets the quotient, and after the DX gets the remainder. So, this way this division operation will be carried out.

(Refer Slide Time: 03:47)



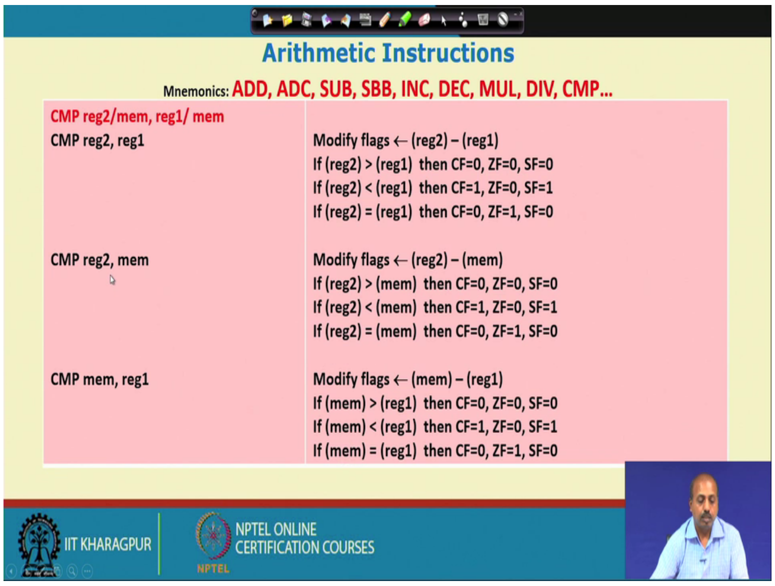
Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

IDIV reg/ mem	
IDIV reg	For 16-bit :- 8-bit : (AL) ← (AX) :- (reg8) Quotient (AH) ← (AX) MOD(reg8) Remainder For 32-bit :- 16-bit : (AX) ← (DX)(AX) :- (reg16) Quotient (DX) ← (DX)(AX) MOD(reg16) Remainder
IDIV mem	For 16-bit :- 8-bit : (AL) ← (AX) :- (mem8) Quotient (AH) ← (AX) MOD(mem8) Remainder For 32-bit :- 16-bit : (AX) ← (DX)(AX) :- (mem16) Quotient (DX) ← (DX)(AX) MOD(mem16) Remainder

We have similarly we have got this IDIV instruction.

(Refer Slide Time: 03:52)



Arithmetic Instructions

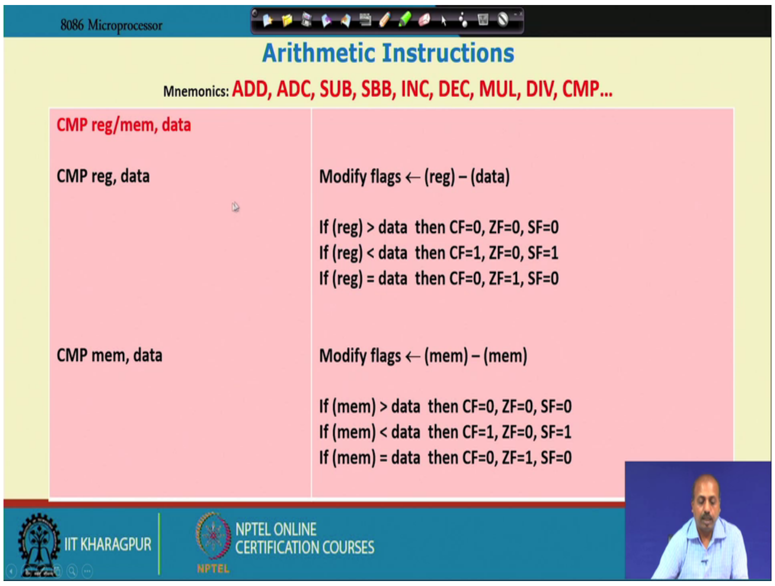
Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

CMP reg2/mem, reg1/ mem	
CMP reg2, reg1	Modify flags ← (reg2) – (reg1) If (reg2) > (reg1) then CF=0, ZF=0, SF=0 If (reg2) < (reg1) then CF=1, ZF=0, SF=1 If (reg2) = (reg1) then CF=0, ZF=1, SF=0
CMP reg2, mem	Modify flags ← (reg2) – (mem) If (reg2) > (mem) then CF=0, ZF=0, SF=0 If (reg2) < (mem) then CF=1, ZF=0, SF=1 If (reg2) = (mem) then CF=0, ZF=1, SF=0
CMP mem, reg1	Modify flags ← (mem) – (reg1) If (mem) > (reg1) then CF=0, ZF=0, SF=0 If (mem) < (reg1) then CF=1, ZF=0, SF=1 If (mem) = (reg1) then CF=0, ZF=1, SF=0

Then there is a comparison instructions. So, this comparison so, it will compare the operand like register 2 register 1 so, we will compare between them. So, if register 2 is greater than register 1, then all this is the this will be the setting CF equal to 0, ZF is equal to 0, SF is equal to 0.

And for different if register 2 is less than register 1 then the carry flag will be 1, ZF is 0 and SF is 1. So, this way this comparison will take place.

(Refer Slide Time: 04:19)



8086 Microprocessor

Arithmetic Instructions

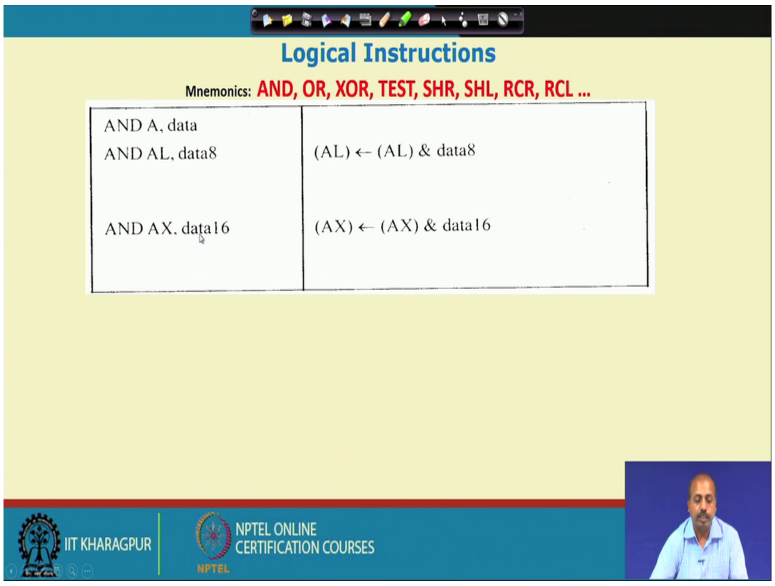
Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

CMP reg/mem, data	
CMP reg, data	Modify flags ← (reg) – (data) If (reg) > data then CF=0, ZF=0, SF=0 If (reg) < data then CF=1, ZF=0, SF=1 If (reg) = data then CF=0, ZF=1, SF=0
CMP mem, data	Modify flags ← (mem) – (mem) If (mem) > data then CF=0, ZF=0, SF=0 If (mem) < data then CF=1, ZF=0, SF=1 If (mem) = data then CF=0, ZF=1, SF=0

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES NPTEL

Then other arithmetic operation like if register is greater than data, then this will be the setting, these are the various settings when we have got this comparison instruction. Logical instructions and or XOR test shift right shift left etcetera.

(Refer Slide Time: 04:32)



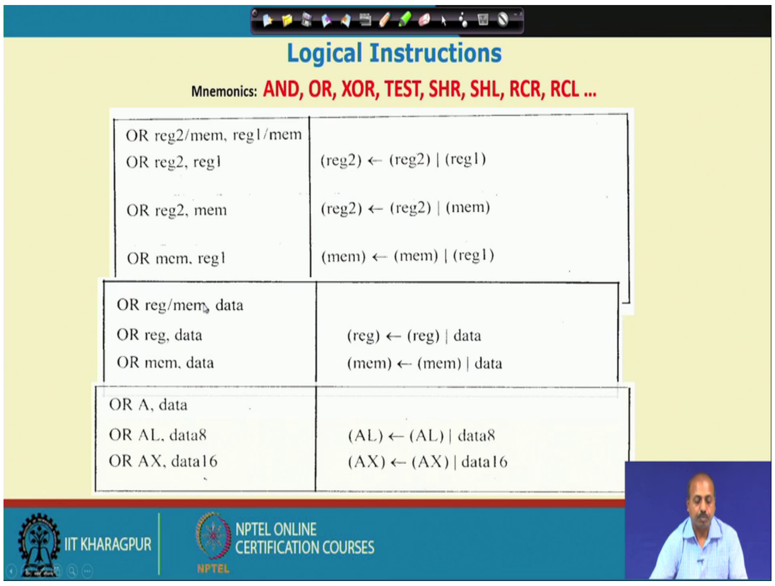
Logical Instructions

Mnemonics: AND, OR, XOR, TEST, SHR, SHL, RCR, RCL ...

AND A, data AND AL, data8	(AL) ← (AL) & data8
AND AX, data16	(AX) ← (AX) & data16

So, and is like just like we had got and in 8085. So, and A comma data so, this is I can have 8-bit data or 16-bit data. So, with AX register they will be ANDed.

(Refer Slide Time: 04:48)



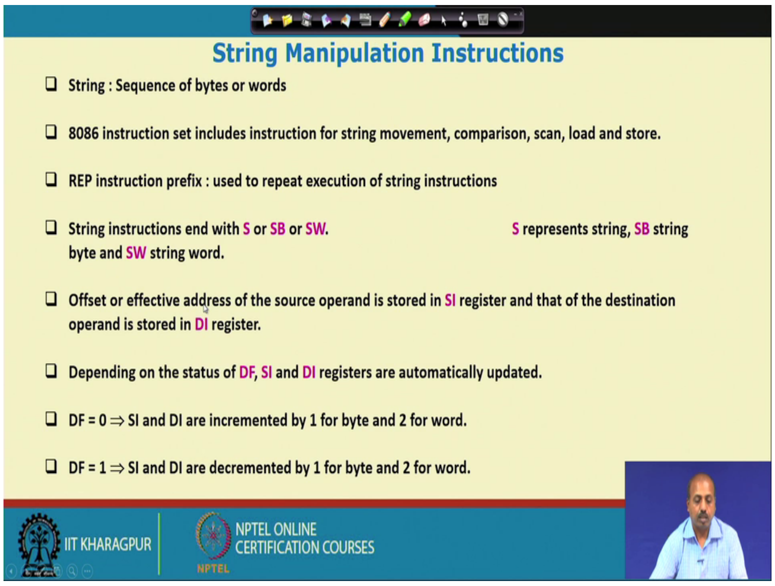
Logical Instructions

Mnemonics: AND, OR, XOR, TEST, SHR, SHL, RCR, RCL ...

OR reg2/mem, reg1/mem OR reg2, reg1	(reg2) ← (reg2) | (reg1)
OR reg2, mem	(reg2) ← (reg2) | (mem)
OR mem, reg1	(mem) ← (mem) | (reg1)
OR reg/mem, data OR reg, data	(reg) ← (reg) | data
OR mem, data	(mem) ← (mem) | data
OR A, data OR AL, data8	(AL) ← (AL) | data8
OR AX, data16	(AX) ← (AX) | data16

Or we can have or instruction again the same thing. So, or will be done and register 2 will get register 2 or register 1, sorry in this case register 2 will get a register 2 memory. So, like that the or operation will take place, and then we have got the string manipulation instructions.

(Refer Slide Time: 05:04)



String Manipulation Instructions

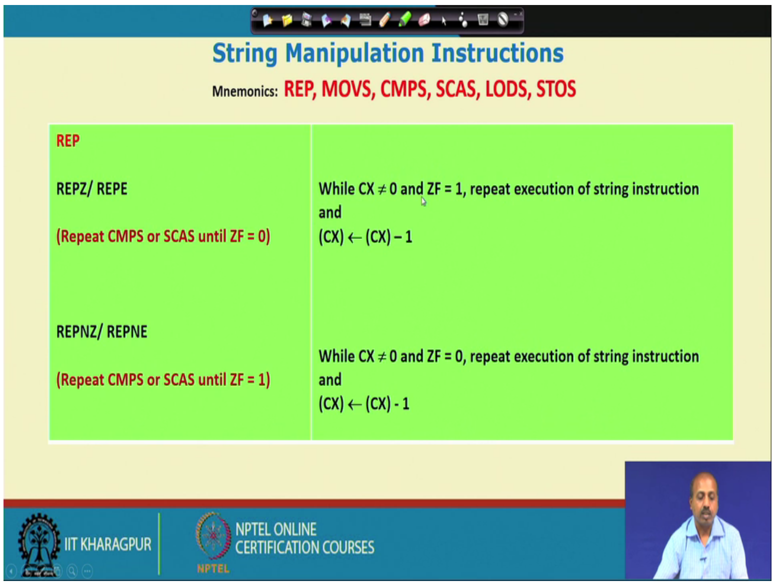
- String : Sequence of bytes or words
- 8086 instruction set includes instruction for string movement, comparison, scan, load and store.
- REP instruction prefix : used to repeat execution of string instructions
- String instructions end with S or SB or SW. S represents string, SB string byte and SW string word.
- Offset or effective address of the source operand is stored in SI register and that of the destination operand is stored in DI register.
- Depending on the status of DF, SI and DI registers are automatically updated.
- DF = 0 ⇒ SI and DI are incremented by 1 for byte and 2 for word.
- DF = 1 ⇒ SI and DI are decremented by 1 for byte and 2 for word.

So, string is nothing but a sequence of bytes or words. So, we can think of it as a sequence of bytes stored in the memory. So, 8086 instruction set includes instructions to for string movement comparison scan, load and store. There is one rep instruction prefix so, it is used to repeat the string instructions. So, if you want to repeat it for some times. So, you can put it repeat prefix before that instruction. And how many times it will be repeated? So, that will be determined by the content of CX register.

So, string instructions end with S SB or SW, where S represent the string SB is the string byte, and SW is the string word. Offset or effective address of the source operand is stored in the SI register, and destination is in the DI register. Depending upon the direction flag DF, SI, DI values will be updated. So, if DF is 0, a SI DI will be incremented by one by one for byte and 2 for words.

So, if the instruction is ending with this B in that case, in that case it will be incremented by 1. If it is ending with W so, it will be incremented by 2. So, if it is if DF is equal to 0, on the other hand if DF is equal to 1, they will be decremented by 1 or 2. So, that way we have got these string value manipulation instructions. So, we will see in more detail some more instruction examples. And then this rep flag that we were talking about this rep prefix.

(Refer Slide Time: 06:36)



String Manipulation Instructions

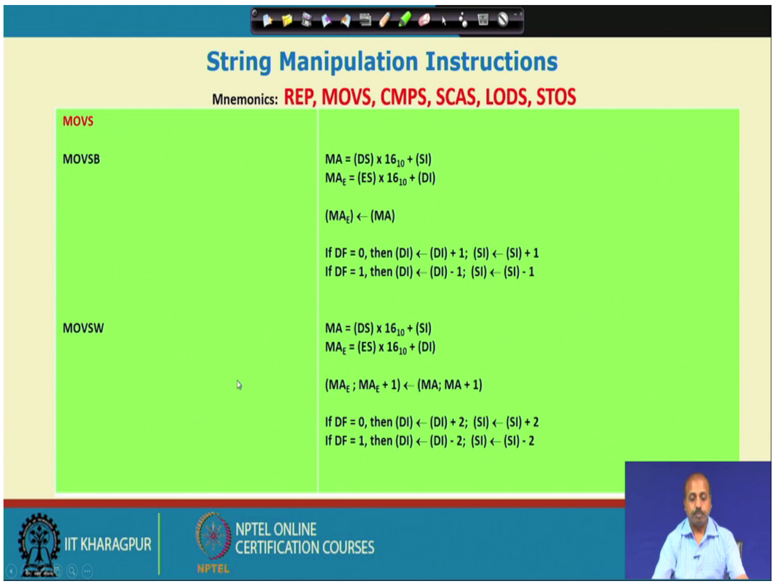
Mnemonics: **REP, MOVS, CMPS, SCAS, LODS, STOS**

REP	
REPZ/ REPE (Repeat CMPS or SCAS until ZF = 0)	While CX $\neq$ 0 and ZF = 1, repeat execution of string instruction and (CX) $\leftarrow$ (CX) – 1
REPNZ/ REPNE (Repeat CMPS or SCAS until ZF = 1)	While CX $\neq$ 0 and ZF = 0, repeat execution of string instruction and (CX) $\leftarrow$ (CX) - 1

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, this is repeat this CMPS orscan still ZF is equal to 0 so, 0 flag equal to 0. So, while CX is not equal to 0, and ZF is equal to 1, repeat execution of a string instruction and CX will be decremented by one every time. And then REPNZ so, if CX is not equal to 0 and DF equal to 0 repeat execution of string instructions, and CX will be made equal to CX is decremented it.

(Refer Slide Time: 07:20)



String Manipulation Instructions

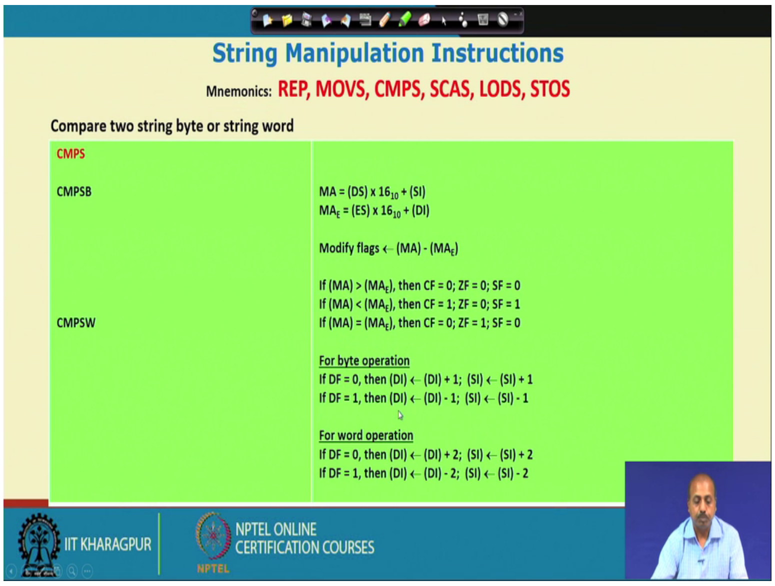
Mnemonics: **REP, MOVS, CMPS, SCAS, LODS, STOS**

MOVS	
MOVSB	MA = (DS) x 16_{10} + (SI) MA_E = (ES) x 16_{10} + (DI) (MA_E) $\leftarrow$ (MA) If DF = 0, then (DI) $\leftarrow$ (DI) + 1; (SI) $\leftarrow$ (SI) + 1 If DF = 1, then (DI) $\leftarrow$ (DI) - 1; (SI) $\leftarrow$ (SI) - 1
MOVSW	MA = (DS) x 16_{10} + (SI) MA_E = (ES) x 16_{10} + (DI) (MA_E ; MA_E + 1) $\leftarrow$ (MA; MA + 1) If DF = 0, then (DI) $\leftarrow$ (DI) + 2; (SI) $\leftarrow$ (SI) + 2 If DF = 1, then (DI) $\leftarrow$ (DI) - 2; (SI) $\leftarrow$ (SI) - 2

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

It is done till z flag is not becomes equal to 1 as long as z flag is 0. So, it will be continuing like that. So, this is the first instruction that we have moves B. So, moves byte. So, what it does? It the first address is calculated the source address is calculated as DS into 16 plus SI. Destination address is calculated at AS ES into 16 plus DI, then there this content is moved to this destination. And after that depending upon the DF flag setting DI and SI registers are updated. And moves W so, it is also similar to this only thing is that 2 bytes will be moved, and then depending upon the DF flag setting DI and SI will be a incremented or decremented by 2.

(Refer Slide Time: 07:58)



String Manipulation Instructions

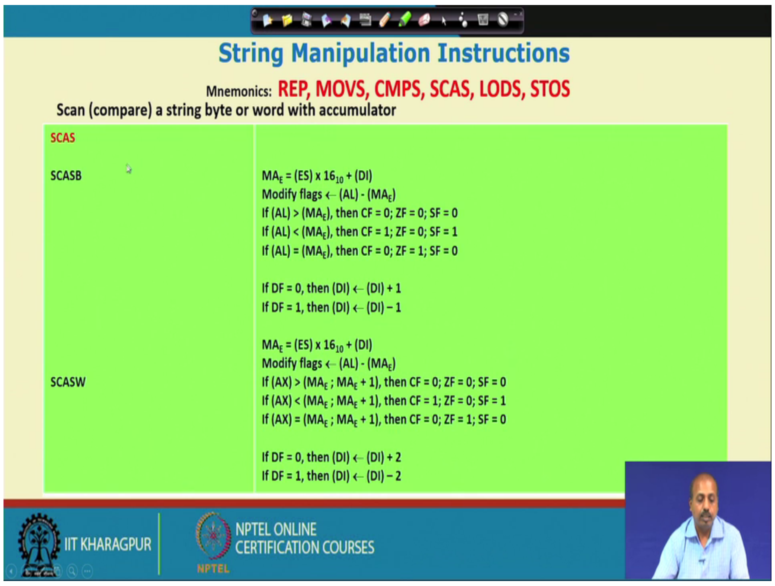
Mnemonics: **REP, MOVS, CMPS, SCAS, LODS, STOS**

Compare two string byte or string word

CMPS	
CMPSB	$MA = (DS) \times 16_{10} + (SI)$ $MA_E = (ES) \times 16_{10} + (DI)$ Modify flags $\leftarrow (MA) - (MA_E)$ If $(MA) > (MA_E)$, then $CF = 0; ZF = 0; SF = 0$ If $(MA) < (MA_E)$, then $CF = 1; ZF = 0; SF = 1$
CMPSW	If $(MA) = (MA_E)$, then $CF = 0; ZF = 1; SF = 0$ For byte operation If $DF = 0$, then $(DI) \leftarrow (DI) + 1;\ (SI) \leftarrow (SI) + 1$ If $DF = 1$, then $(DI) \leftarrow (DI) - 1;\ (SI) \leftarrow (SI) - 1$ For word operation If $DF = 0$, then $(DI) \leftarrow (DI) + 2;\ (SI) \leftarrow (SI) + 2$ If $DF = 1$, then $(DI) \leftarrow (DI) - 2;\ (SI) \leftarrow (SI) - 2$

Then this CMPS instruction to compare so, compare which have got compare byte and compare word. So, otherwise it is similar to this moves B, but it is it is checking the it is comparing between the strings. So, if this source string is larger source byte is larger than the destination byte then CF will be made equal to 0. So, this will be the flag setting, otherwise this will be the flag setting so, it will go like this. Then we have got scan a string byte or word with accumulator.

(Refer Slide Time: 08:26)



String Manipulation Instructions

Mnemonics: **REP, MOVS, CMPS, SCAS, LODS, STOS**

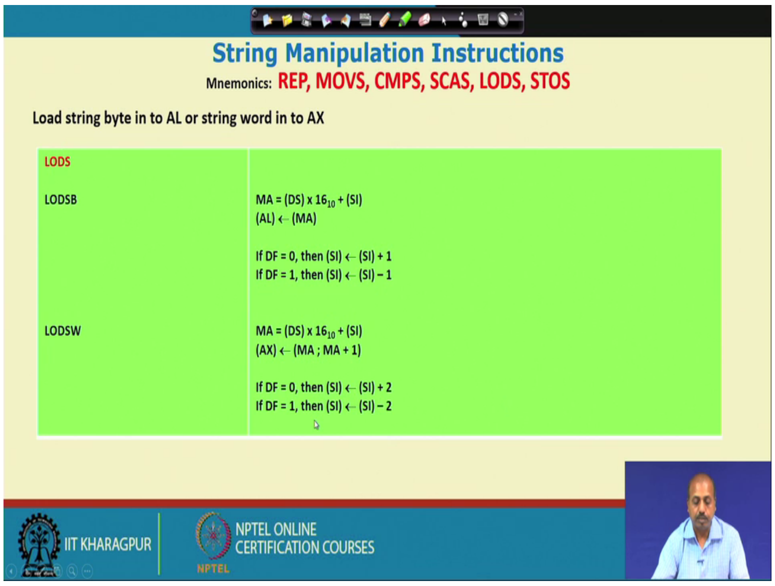
Scan (compare) a string byte or word with accumulator

SCAS	
SCASB	$MA_E = (ES) \times 16_{10} + (DI)$ Modify flags $\leftarrow (AL) - (MA_E)$ If $(AL) > (MA_E)$, then $CF = 0; ZF = 0; SF = 0$ If $(AL) < (MA_E)$, then $CF = 1; ZF = 0; SF = 1$ If $(AL) = (MA_E)$, then $CF = 0; ZF = 1; SF = 0$ If $DF = 0$, then $(DI) \leftarrow (DI) + 1$ If $DF = 1$, then $(DI) \leftarrow (DI) - 1$
SCASW	$MA_E = (ES) \times 16_{10} + (DI)$ Modify flags $\leftarrow (AL) - (MA_E)$ If $(AX) > (MA_E ; MA_E + 1)$, then $CF = 0; ZF = 0; SF = 0$ If $(AX) < (MA_E ; MA_E + 1)$, then $CF = 1; ZF = 0; SF = 1$ If $(AX) = (MA_E ; MA_E + 1)$, then $CF = 0; ZF = 1; SF = 0$ If $DF = 0$, then $(DI) \leftarrow (DI) + 2$ If $DF = 1$, then $(DI) \leftarrow (DI) - 2$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, it is basically comparison with the accumulator. So, it will compare AL value with this content of the memory location which is pointed to by DS ES colon DI. And the other hand, if this is SCASW so, this will compare the word. So, it will be comparing AX register with the 2 successive memory addresses, which is pointed to by this ES colon DI. And then DF will be depending upon the DF flag so, it is DI value will be either incremented or decremented.

(Refer Slide Time: 09:05)



String Manipulation Instructions

Mnemonics: **REP, MOVS, CMPS, SCAS, LODS, STOS**

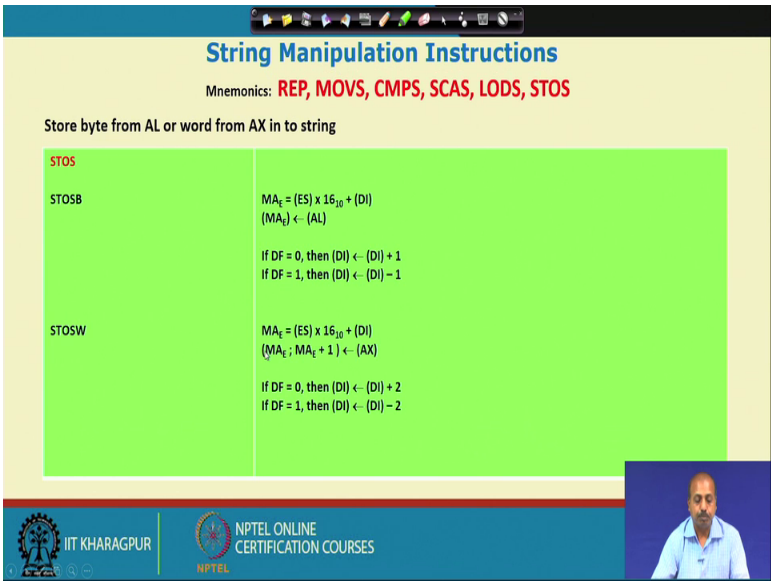
Load string byte in to AL or string word in to AX

LODS	
LODSB	$MA = (DS) \times 16_{10} + (SI)$ $(AL) \leftarrow (MA)$ If $DF = 0$, then $(SI) \leftarrow (SI) + 1$ If $DF = 1$, then $(SI) \leftarrow (SI) - 1$
LODSW	$MA = (DS) \times 16_{10} + (SI)$ $(AX) \leftarrow (MA ; MA + 1)$ If $DF = 0$, then $(SI) \leftarrow (SI) + 2$ If $DF = 1$, then $(SI) \leftarrow (SI) - 2$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then we have got load so, it will load the string byte into AL or string word into AX. So, this loads B so, it will be the source and the address is calculated as DS into 16 plus SI. And this AL register gets the content of that memory location. If DF is 0 then this SI is incremented if it is one, then SI is decremented. And this loads W so, this will be moving one-word from a memory locations in to the AX register.

(Refer Slide Time: 09:35)



String Manipulation Instructions

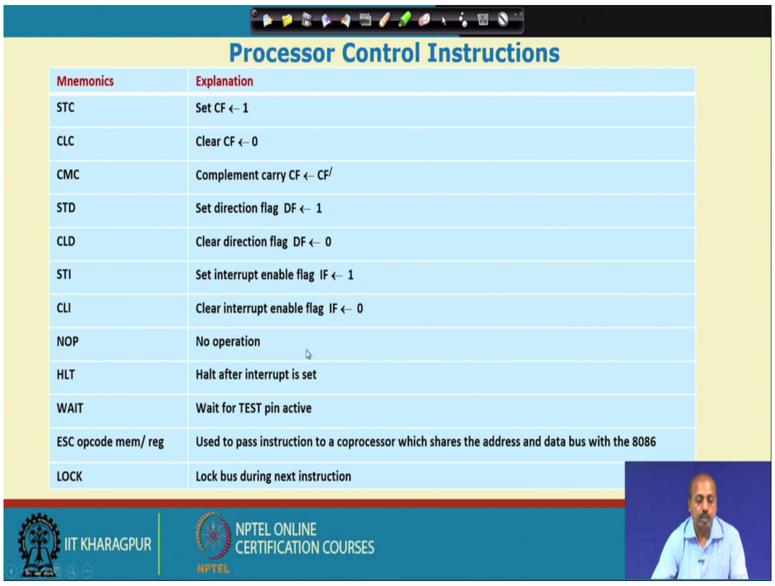
Mnemonics: REP, MOVS, CMPS, SCAS, LODS, STOS

Store byte from AL or word from AX in to string

STOS	
STOSB	$MA_E = (ES) \times 16_{10} + (DI)$ $(MA_E) \leftarrow (AL)$ If DF = 0, then $(DI) \leftarrow (DI) + 1$ If DF = 1, then $(DI) \leftarrow (DI) - 1$
STOSW	$MA_E = (ES) \times 16_{10} + (DI)$ $(MA_E ; MA_E + 1) \leftarrow (AX)$ If DF = 0, then $(DI) \leftarrow (DI) + 2$ If DF = 1, then $(DI) \leftarrow (DI) - 2$

So, just the reverse of these loads we have got STOS so, we have got STOSB and STOSW. So, this is a storing the content of AL register on to the memory address or it is called storing the content of a AX pair onto this 2 memory locations, MAE and MAE plus 1. So, that way this loading storing and loading will be done.

(Refer Slide Time: 10:02)



Processor Control Instructions

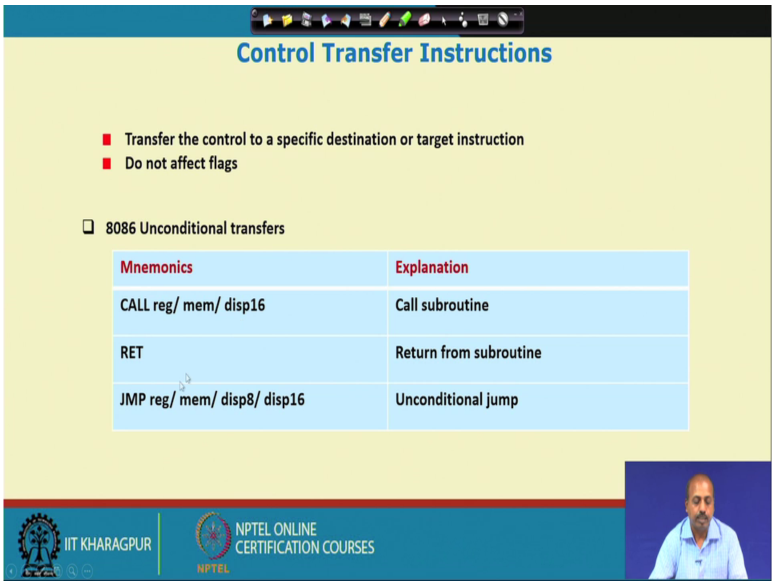
Mnemonics	Explanation
STC	Set CF $\leftarrow$ 1
CLC	Clear CF $\leftarrow$ 0
CMC	Complement carry CF $\leftarrow$ CF$^{/}$
STD	Set direction flag DF $\leftarrow$ 1
CLD	Clear direction flag DF $\leftarrow$ 0
STI	Set interrupt enable flag IF $\leftarrow$ 1
CLI	Clear interrupt enable flag IF $\leftarrow$ 0
NOP	No operation
HLT	Halt after interrupt is set
WAIT	Wait for TEST pin active
ESC opcode mem/ reg	Used to pass instruction to a coprocessor which shares the address and data bus with the 8086
LOCK	Lock bus during next instruction

So, then we have got a number of control instructions like this STC like set carry flag equal to 0. So, they are they are 0 operand instruction the CLC carry flag is 0, clear carry flag. CMC it will complement the carry, STD set it will set the direction flag like in the instruction.

So, you are telling if DF equal to 0, then this will happen, or this will happen. So, how to set the direction flag? So, that is given by this STD and CLD instructions, then we have got STI and CLI regarding interrupt flags enable and disable. So, we have got this NOP instruction NOP just like in 8085. So, you have got halt instruction for it is it is halt after the interrupt is set. And the wait, it is wait for the test pin to become active. So, as long as this test pin is high so, it will be remaining in the wait state, when the test pin will become low, then this condition is true so, it will be coming out.

Then there is an escape opcode so, that is used for the coprocessor. So, if you have an instruction with start with and starts with an escape opcode so, the 8086 will pass on the instruction to the coprocessor. And this lock prefix so, that will be locked the bus during the next instruction. So, this if you execute a lock instruction then in the next for the next instruction. So, bus will not be released by 8086 to others. So, next instruction will be executing in an uninterrupted fashion. So, this transfer control to a specific destination or target and they do not affect the flags. So, this control transfer so, we have got call instruction, return instruction and jump instruction.

(Refer Slide Time: 11:33)



Control Transfer Instructions

- Transfer the control to a specific destination or target instruction
- Do not affect flags

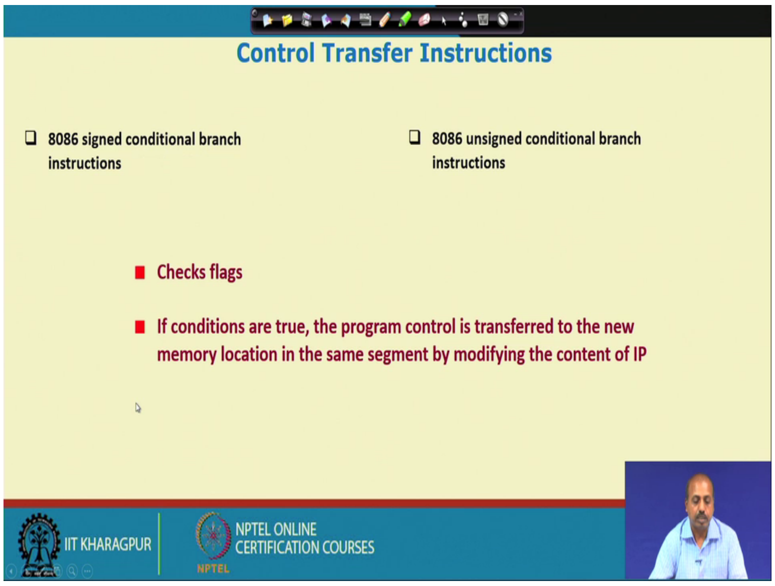
- 8086 Unconditional transfers

Mnemonics	Explanation
CALL reg/ mem/ disp16	Call subroutine
RET	Return from subroutine
JMP reg/ mem/ disp8/ disp16	Unconditional jump

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, call so, this is calling the subroutine. So, the address can be directly mention a 16-bit displacement a memory location address or a register. So, the register content will be used as the memory address. In this case, memory location content will be used as the target address. And here the address is specified directly. And similarly we have got the jump instruction.

(Refer Slide Time: 12:08)



Control Transfer Instructions

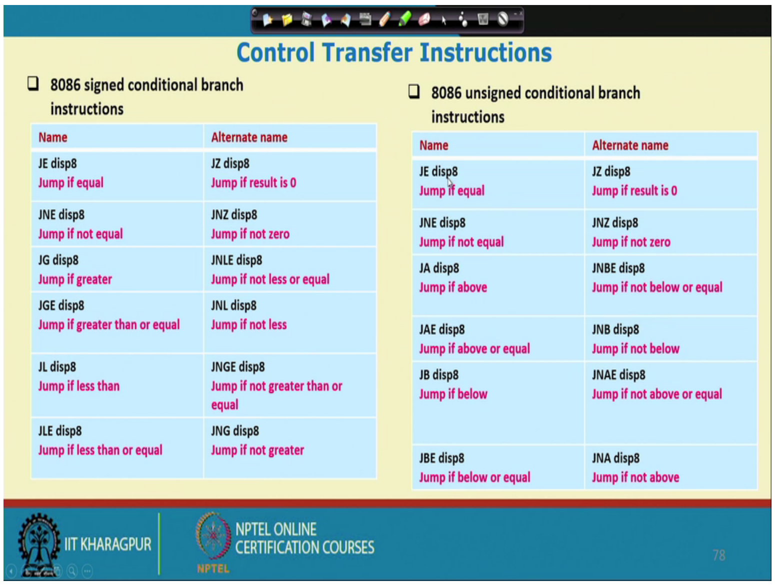
- 8086 signed conditional branch instructions
- 8086 unsigned conditional branch instructions

- Checks flags
- If conditions are true, the program control is transferred to the new memory location in the same segment by modifying the content of IP

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Then in 8086 conditional branch instruction so, they will they will check the flags, and then if the condition is true the program control will be transfer to the new memory location in the same segment by modifying the content of the IP register.

(Refer Slide Time: 12:25)



Control Transfer Instructions

- 8086 signed conditional branch instructions

Name	Alternate name
JE disp8 Jump if equal	JZ disp8 Jump if result is 0
JNE disp8 Jump if not equal	JNZ disp8 Jump if not zero
JG disp8 Jump if greater	JNLE disp8 Jump if not less or equal
JGE disp8 Jump if greater than or equal	JNL disp8 Jump if not less
JL disp8 Jump if less than	JNGE disp8 Jump if not greater than or equal
JLE disp8 Jump if less than or equal	JNG disp8 Jump if not greater

- 8086 unsigned conditional branch instructions

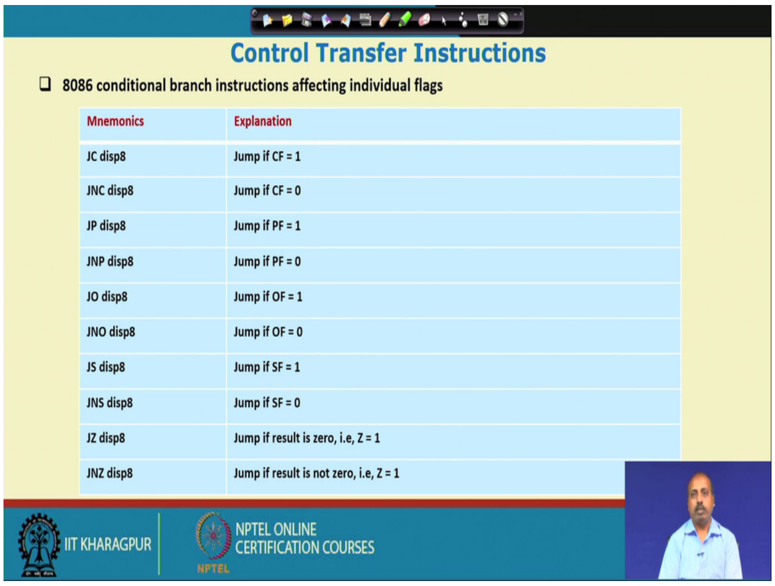
Name	Alternate name
JE disp8 Jump if equal	JZ disp8 Jump if result is 0
JNE disp8 Jump if not equal	JNZ disp8 Jump if not zero
JA disp8 Jump if above	JNBE disp8 Jump if not below or equal
JAE disp8 Jump if above or equal	JNB disp8 Jump if not below
JB disp8 Jump if below	JNAE disp8 Jump if not above or equal
JBE disp8 Jump if below or equal	JNA disp8 Jump if not above

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL | 78

So, this we have got JE so, JE displacement 8 jump if equal. So, if the equality condition is satisfied, then it is the 8-bit displacement specified. So, that will be added with IP and that will be making the 8 bit- jump. So, similarly we have got JNE or JNZ type of instruction. So, there also we have got this 8 bit, all this conditional branches so, they are related to the IP address with that to the IP value instruction pointer value. So, it will be jumping with respect to IP.

On the other end, this un conditional and unsigned conditional branch. So, we have got this JE displacement 8 and the things like that. So, this will be executing in a same fashion ok. So, we have so, this values may be unsigned, so, that is that is there.

(Refer Slide Time: 13:19)



Control Transfer Instructions

- 8086 conditional branch instructions affecting individual flags

Mnemonics	Explanation
JC disp8	Jump if CF = 1
JNC disp8	Jump if CF = 0
JP disp8	Jump if PF = 1
JNP disp8	Jump if PF = 0
JO disp8	Jump if OF = 1
JNO disp8	Jump if OF = 0
JS disp8	Jump if SF = 1
JNS disp8	Jump if SF = 0
JZ disp8	Jump if result is zero, i.e, Z = 1
JNZ disp8	Jump if result is not zero, i.e, Z = 1

So, conditional branch instruction that can affect a flags like JC. So, jump if carry flag equal to 1 JNC jump if carry flag equal to 0 so, it will go like that. So, there are many such conditional branches that are available in 8086 that can be used for doing branching in that.

So, in this way in 8086 we find that we have got a much Richard set of registers we have got much Richard set of instructions and operations that can be done as compare to 8085 and if you look into these further processors, then the structure will become more and more complex.

So, all these developments are done starting with digital circuits, where we know how to design individual logic modules or individual registers individual combinational functions and all. And from there it is develop towards such complex systems. So, that they gives a good idea like how we can design complex systems and what are the what type of complex systems we can digital systems we can think about with using those basic gates and flip flops.