

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

Lecture – 63
8086 (Contd.)

(Refer Slide Time: 00:20)

The slide is titled "Instruction Set" and states "8086 supports 6 types of instructions." It lists the following categories:

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logical Instructions
4. String manipulation Instructions
5. Process Control Instructions
6. Control Transfer Instructions

The slide also features the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a small video inset of the professor in the bottom right corner.

So, instruction set of 8086, so if so it supports six types of instructions. So, just like other processors that we have seen, it supports a good number of instruction types. They can be grouped into data transfer instruction, arithmetic instruction, logical instructions, string manipulation instructions, process control instructions and control transfer instructions. So, some of them are common to us, they are well known to us like data transfer, arithmetic, logical etcetera. Whereas, string manipulation instruction, so they are something new. And this process control instruction, so there will be some instruction which are new to us. And this some control transfer instructions will also different types of branches calls etcetera, so they are processor specific. So, we will see that.

(Refer Slide Time: 01:00)

8086 Microprocessor

Data Transfer Instructions

Instructions that are used to transfer data/ address in to registers, memory locations and I/O ports.

Generally involve two operands: Source operand and Destination operand of the same size.

Source: Register or a memory location or an immediate data
Destination: Register or a memory location.

The size should be either a byte or a word.

A 8-bit data can only be moved to 8-bit register/ memory and a 16-bit data can be moved to 16-bit register/ memory.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

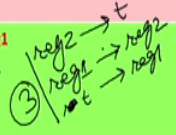
The first category that we have are the data transfer instructions. So, these instructions are used to transfer data or address into registers, memory locations and io ports. So, they involve two operands, now a source operands and a destination operand, and the operands are of size. Now, in case of 8086, the source can be a register or a memory location or an immediate data; and the destination can be a register or a memory location; naturally destination cannot be immediate, it cannot be an immediate data.

There is another restriction like you cannot have both source and destination as memory. So, you cannot have an instruction where the this you can you cannot say like MOV say BX comma SI. So, this is not possible. Here what I wanted to mean is the memory location BX gets the content of memory location SI, so that is not allowed. So, I can have source as a register destination has a memory or so this is one possibility, source as a register destination as a memory or I can have source as a memory and destination as a register, so that is possible or source as an immediate operand and destination as a register or destination as a memory.

But I cannot have that source as a memory, so this is also memory and this is also memory, so that is not possible. So, you cannot put both of them as memory. And definitely the size matters. So, size should be same. So, they should be either a byte or a word. One 8-bit data can only be moved to 8-bit AH register or memory, and a 16-bit data can be move to 16-bit register or a memory, so that is obvious.

(Refer Slide Time: 02:49)

Data Transfer Instructions	
Mnemonics: MOV, XCHG, PUSH, POP, IN, OUT ...	
MOV reg2/ mem, reg1/ mem MOV reg2, reg1 MOV mem, reg1 MOV reg2, mem	(reg2) ← (reg1) (mem) ← (reg1) (reg2) ← (mem)
MOV reg/ mem, data MOV reg, data MOV mem, data	(reg) ← data (mem) ← data
XCHG reg2/ mem, reg1 XCHG reg2, reg1 XCHG mem, reg1	(reg2) ↔ (reg1) (mem) ↔ (reg1)



So, if you look into different category of data transfer instructions, in the first category we have got MOV register 2 memory or first operand is register, second operand is memory or first operand is second operand can be register or a memory. So, like MOV register 2 comma register 1. So, register 2 gets register 1 memory comma register 1. So, memory gets a content of register 1. On MOV register 2 comma memory so from memory it comes to register 2.

So, as I was telling so there is no instruction level MOV memory comma memory, so that is not there ok. So, we cannot have both operands as memory. Then we can have if reg so MOV register or memory comma data, so this is the immediate mode. So, you can have MOV some immediate data to a register or in memory location. So, you can have in that category, you can have data movement to register or data movement to memory. There is another instruction which is not there in say 8085 type of processors too much which is the exchange.

So, it will exchange this content of two registers or a memory location and a register. So, this exchange register 2 register 1, the content of those two registers will get exchanged or we can have this exchange memory comma register 1 the content of this memory location will get exchanged with register 1. So, this way we can have this exchange instruction.

Now, this exchange instructions have apparently it since there is no point having such an exchange instruction, because this exchange is equivalent to moving first this AH register 2 to some temporary register. And then from then register 1 will get the content sorry register 2 will move it to register sorry this register 1, we can put this we can send this register 1 to register 2. And then finally, the register 1 this t value the temporary value can be moved to register 1, so that way this register 1 and register 2 content can get exchanged.

But the point is if we are trying to do this that means, it requires three instructions ok. And this creates difficulty in many cases for the operating system designers, we need some facility by which in a single instruction we are able to read the content of a memory location and change its content ok. So, for that type of instructions for that type of facilities, so this is a multi byte or multi instruction solution. So, they are not correct ok. So, we need to have single instruction which will be able to do all these things, so that is why this exchange instructions may be useful for them. So, this has been introduced for this operating system design purpose.

(Refer Slide Time: 05:50)

Data Transfer Instructions
Mnemonics: **MOV, XCHG, PUSH, POP, IN, OUT ...**

PUSH reg16/ mem	
PUSH reg16 <i>PUSH BX</i>	$(SP) \leftarrow (SP) - 2$ $MA_s = (SS) \times 16_{10} + SP$ $(MA_s; MA_s + 1) \leftarrow (reg16)$
PUSH mem	$(SP) \leftarrow (SP) - 2$ $MA_s = (SS) \times 16_{10} + SP$ $(MA_s; MA_s + 1) \leftarrow (mem)$
POP reg16/ mem	
POP reg16	$MA_s = (SS) \times 16_{10} + SP$ $(reg16) \leftarrow (MA_s; MA_s + 1)$ $(SP) \leftarrow (SP) + 2$
POP mem	$MA_s = (SS) \times 16_{10} + SP$ $(mem) \leftarrow (MA_s; MA_s + 1)$ $(SP) \leftarrow (SP) + 2$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we will look into the other data transfer instructions like push and pop instructions. This push instruction, so this works with push register 16. So, 16-bit registers can be pushed like you can have like push AX. So, you can have instructions like say push say BX. So, this BX register content will be pushed. So, first this stack pointer is

decremented by 2, and then this memory address effective address is formed by SS stack segment multiplied by 16 plus this stack pointer. Then this higher order byte will go to this memory address s and this lower order byte goes to MA s plus 1 that way the two successive bytes they will be set. And then with similarly the push memory, so you can also have this push memory.

So, here this so this is for here also it is the stack pointer is decremented by 2, then this memory address is formed. And then the two bytes from the starting at this memory location will be pushed into the stack. And we have just a reverse of push the pop instruction. So, here the only thing is that the stack pointer is incremented by two after getting the content from the stack. So, this register 16, so it will be the content will be coming from this two locations from the stack its and then the stack pointer will be incremented by 2. And similarly, if it is the pop memory here also the same thing. So, content from these two stack memory location they will come to the memory address specified here and then the stack pointer will be incremented by two. So, we can have ah this push pop also for this data transfer.

(Refer Slide Time: 07:46)

Data Transfer Instructions
 Mnemonics: **MOV, XCHG, PUSH, POP, IN, OUT ...**

IN A, [DX] <i>16 bit Indirect</i>	$PORT_{addr} = (DX)$ $(AL) \leftarrow (PORT)$	OUT [DX], A	$PORT_{addr} = (DX)$ $(PORT) \leftarrow (AL)$
IN AL, [DX]	$PORT_{addr} = (DX)$ $(AL) \leftarrow (PORT)$	OUT [DX], AL	$PORT_{addr} = (DX)$ $(PORT) \leftarrow (AL)$
IN AX, [DX]	$PORT_{addr} = (DX)$ $(AX) \leftarrow (PORT)$	OUT [DX], AX	$PORT_{addr} = (DX)$ $(PORT) \leftarrow (AX)$
IN A, addr8 <i>Direct</i>	$(AL) \leftarrow (addr8)$	OUT addr8, A	$(addr8) \leftarrow (AL)$
IN AL, addr8	$(AL) \leftarrow (addr8)$	OUT addr8, AL	$(addr8) \leftarrow (AL)$
IN AX, addr8 <i>16 bit</i>	$(AX) \leftarrow (addr8)$	OUT addr8, AX	$(addr8) \leftarrow (AX)$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then other instructions that we have is the input part the by which we can reach some port and the output part by which we can transfer some output to the output port. So, in A comma DX, so this is for 16-bit. So, this as you know that we said it is for 16-bit port address 16-bit port address. So, this is indirect one indirect address. So, port address this

DX value will be put onto the address bus for port address, and this content of that particular port will come to the AL register.

And this instruction, so destination, so this is AX, so it is taken as the port to be 16-bit. So, it is expected that on getting this DX value on to the address bus, the port will get selected, and this 16-bit port value will be coming to the AX register ok. And for 8-bit at port address, so we can specify it directly. So, address 8, so this is an 8-bit immediate value. So, we can directly specify some 8-bit port address there. So, this is the direct port access and that is indirect port access.

So, in AL comma address 8, so as a result from the from that port, the value will come to the AL register MOV in AL comma A MOV A in AX comma address 8. So, the 8-bit port address is put, but the port is a 16-bit port. So, port is a 16-bit port. So, it gives 16-bit value onto the data bus and that comes to the AX register. And just like in so we have got out instructions, so out DX comma A type of instructions or out address 8 comma A type of instruction. So, they are just counter parts for the in instructions.

(Refer Slide Time: 09:38)

Arithmetic Instructions	
Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...	
ADD reg2/ mem, reg1/mem	
ADD reg2, reg1	$(reg2) \leftarrow (reg1) + (reg2)$
ADD reg2, mem	$(reg2) \leftarrow (reg2) + (mem)$
ADD mem, reg1	$(mem) \leftarrow (mem) + (reg1)$
ADD reg/mem, data	
ADD reg, data	$(reg) \leftarrow (reg) + data$
ADD mem, data	$(mem) \leftarrow (mem) + data$
ADD A, data	
ADD AL, data8	$(AL) \leftarrow (AL) + data8$
ADD AX, data16	$(AX) \leftarrow (AX) + data16$

Next, we have got the arithmetic instructions so far we had the this data movement instruction or data transfer instructions, now we have got arithmetic instruction. Now, in arithmetic instruction we have got the add as the first exemplary instruction the so we can have this operands like register 2 and register 1 as the two operands. And the meaning is that register 2 will get the content of register 1 plus register 2. The flags are affected it is

not specified here directly, but while discussing the status flag, so we have seen that the flags will get affected by doing this operation.

So, this similarly the second operand can be memory also. So, so you can have like ADD register 2 comma memory. So, memory register 2 gets register 2 plus memory. So, you can have at least one of the operands as a register. So, both the operands cannot be memory. So, here the third instruction you have got ADD memory comma register 1. So, memory gets the content of memory plus the register 1. So, memory can be specified it is a address specified in different modes that we have seen previously. Then we have got immediate addition, so ADD register comma data where data is a is a is an immediate data.

So, register get register plus data or you can have ADD memory comma data, where the memory content will be added and the data will be added with the memory content the immediate data will be added and the content is updated. And we can have this ADD A comma data, so directly we can specify this immediate value AL and AX with AL this 8-bit data will be added or with AX this 16-bit data will be added. Then there is ad ADC there is ADD with carry, so similar to ADD, but it is with the carries ok. So, carry flag will also be added. So, this is register 2 will get register 1 plus register 2 plus carry flag. So, otherwise it is same as add, but this carry flag will also be added in the summation process.

(Refer Slide Time: 11:45)

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

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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then the subtract instruction SUB. So, it is again similar to this ADD instruction, but this for example, this SUB register 2 comma register 1, so it will register 2 will get register 1 minus register 2. So, otherwise it is same. So, whatever we have got for ADD, the same is true for SUB. And of course, the flag settings will be different, so overflow zero, so they will be set depending upon the content that you are getting there.

(Refer Slide Time: 12:16)

Arithmetic Instructions	
Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...	
SBB reg2/ mem, reg1/mem	
SBB reg2, reg1	$(reg2) \leftarrow (reg1) - (reg2) - CF$
SBB reg2, mem	$(reg2) \leftarrow (reg2) - (mem) - CF$
SBB mem, reg1	$(mem) \leftarrow (mem) - (reg1) - CF$
SBB reg/mem, data	
SBB reg, data	$(reg) \leftarrow (reg) - data - CF$
SBB mem, data	$(mem) \leftarrow (mem) - data - CF$
SBB A, data	
SBB AL, data8	$(AL) \leftarrow (AL) - data8 - CF$
SBB AX, data16	$(AX) \leftarrow (AX) - data16 - CF$

Then subtract with borrow just like we have got ADD with carry. So, we can have subtract with borrow. So, the operation is register 2 get gets register 1 minus register 2 minus the carry flag also. So, as a result this is the carry is also subtracted so borrow is also subtracted. So, if you are if you are having say multi byte addition or subtraction then this ADC and SBB instructions may be useful as we have seen previously that for multi byte addition. So, from one byte addition, some carry or borrow is generated and then that is taken the in next byte while adding a subtracting that next byte. So, we can take this carry or borrow into consideration, so that way we can have this subtract with borrow type of instructions useful.

(Refer Slide Time: 13:02)

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

INC reg/ mem	
INC reg8	$(reg8) \leftarrow (reg8) + 1$
INC reg16	$(reg16) \leftarrow (reg16) + 1$
INC mem	$(mem) \leftarrow (mem) + 1$
DEC reg/ mem	
DEC reg8	$(reg8) \leftarrow (reg8) - 1$
DEC reg16	$(reg16) \leftarrow (reg16) - 1$
DEC mem	$(mem) \leftarrow (mem) - 1$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then there is increment instruction. So, this is just incrementing some register by 1. So, increment register 8. So, this will be incrementing register 8 by 1 bit increment register 16. So, it will increment register 16 by one not one bit by the value one and increment memory. So, this will increment the memory location by one. And just the opposite of increment we have got the decrement instruction, so decrement register 8 will decrement the register 8 by the 8-bit register by one. So, like that you can specify any 8-bit register here, so that value will be decremented.

(Refer Slide Time: 13:41)

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

MUL reg/ mem	
MUL reg	$\begin{matrix} \text{MUL } \text{CL} \\ \text{MUL } \text{CL} \\ \text{AL} * \text{CL} \end{matrix}$ $(AX) \leftarrow (AX) \times (reg8)$ $(DX)(AX) \leftarrow (AX) \times (reg16)$
MUL mem	$(AX) \leftarrow (AL) \times (mem8)$ $(DX)(AX) \leftarrow (AX) \times (mem16)$
IMUL reg/ mem	
IMUL reg	$(AX) \leftarrow (AL) \times (reg8)$ $(DX)(AX) \leftarrow (AX) \times (reg16)$
IMUL mem	$(AX) \leftarrow (AX) \times (mem8)$ $(DX)(AX) \leftarrow (AX) \times (mem16)$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then the multiply instruction so multiply registers. So, what will happen is that AX register will get AL content of AL into register 8. So, this AL register should for multiplication there are two numbers that we want to multiply. So, the first number should be in the AL register. Second number you can put it onto some register, and mention that register name. For example, you can say like you can say like say MUL you can say like MUL say CL.

So, what will happen in the AL register, I will have I have some content, so that will be multiplied by this CL register and the results, so both of them being 8-bit register, so this result is a 16-bit register value. And this out of this 16-bit this AX register will hold the this AX, AX register will hold the 16-bit. So, out of that AL will hold the lower order 8-bit, AH will hold the higher order 8-bit. Now, if the operation is on 16-bit multiplication like if you say if you say like an instruction is a MUL CX.

So, instead of 8-bit is 16-bit multiplication. So, it is expected that the AX register holds the other operand, and CX is the other register operand that we have. So, it will do like AX will be multiplied by CX. So, as a result this multiplication will produce a 32-bit output. So, in the out of this 32-bit output that DX register will hold the higher order 16-bit and the AX register will hold the lower order 16-bit, so that is how this multiplication instruction will take place. Then there is I MUL I MUL instructions, so this will be integer multiplication. So, it will be it will be doing this multiplication operation is similar, but it is just for the integers.

(Refer Slide Time: 15:46)

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

Operation	16-bit	32-bit
DIV reg/mem	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 reg	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

Handwritten notes:
DIV BX
 $\frac{DX:AX}{BX} = Q \rightarrow AX$
 $R \rightarrow DX$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, similarly we have got this division operation. So, division with register or memory, so this division AL will hold this AX divided by register 8. So, this ah. So, this is the this is the quotient part that will be going to the AL register, and the AH will get the remainder part. So, AX mod register 8. So, that will go to the AH register. So, so this whatever register is specified, so that so that will be used to divide the content of AX register, and the AL register will get this quotient part and AH register will hold the remainder part. For 32 bit division, so this quotient the dividend is contained in DX and AX pair, DX will hold 16-bit higher order 16-bit, AX will have lower order 16-bit and that is multiplied by that is divided by this register 16.

Like you can say write like something like say DIV you can say like DIV BX where a 16-bit division. So, what it will do. So, DX colon AX, so that will be divided by BX. So, this will produce a quotient and a remainder part. So, the quotient part will go to the AX register. So, this will be available in the AX register, and the remainder part will be available in the DX register. So, this way it does the divisions. Similarly, instead of registers, you can have this memory operand also and the otherwise it is same, the operation is similar.

(Refer Slide Time: 17:22)

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

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

Handwritten notes:
DIV BX
DX:AX = Q → AX
R → DX

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we have got this other division also, this IDIV is instruction for this another division operands operation that we can have there.

(Refer Slide Time: 17:34)

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

CMP reg2/mem, reg1/ mem	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, reg1	
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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then we have got the comparison instruction then this comparison instruction. So, it will compare two registers or one register with a memory location ok, so but you cannot have the same thing that is you cannot have both the locations both the operands as memory, so that is not possible. So, the flags will be modified like say compare a registered 2 comma register 1. So, what it will do the modified flag, so that they would depend on the

value of this register 2 minus register 1. If register 2 is greater than register 1, then carry flag will be 0, zero flag will be 0, and sign flag will also be 0. If register 2 is less than register 1 in that case carry flag is 1, zero flag is 1, 0, and the sign flag is 1.

So, basically the if so by this we can check for this less than condition. After this comparison so if you put a jump on jump on less or something like that then if the if this carry flag and sign flag they are set that will mean that there is an there is a comparison where register 2 was really less than register 1. And for equality condition this carry flag is 0, sign flag is also 0, and this zero flag is set to 1. So, with that you can have instructions like jump on 0; and all that after the comparison instruction. So, rest of the thing they are similar. So, you have got this comparison with one of the operand may be memory for the comparison operation.

(Refer Slide Time: 19:03)

The slide is titled "Arithmetic Instructions" and lists mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP... It is divided into two sections for the CMP instruction:

- CMP A, data**
Instruction: $CMP\ AL, data8$
Modify flags $\leftarrow (AL) - data8$
If $(AL) > data8$ then $CF=0, ZF=0, SF=0$
If $(AL) < data8$ then $CF=1, ZF=0, SF=1$
If $(AL) = data8$ then $CF=0, ZF=1, SF=0$
- CMP AX, data16**
Instruction: $CMP\ AX, data16$
Modify flags $\leftarrow (AX) - data16$
If $(AX) > data16$ then $CF=0, ZF=0, SF=0$
If $(mem) < data16$ then $CF=1, ZF=0, SF=1$
If $(mem) = data16$ then $CF=0, ZF=1, SF=0$

The slide footer includes the IIT Kharagpur logo and the text "NPTEL ONLINE CERTIFICATION COURSES". A small video inset of a speaker is visible in the bottom right corner.

So, we can also have comparison with immediate data like compare a compare register comma data where this data is an immediate operand. So, you can do this comparison like if register is greater than data then this say flags will be set like this. If it is less than data then the flag setting will be like this. So, this way we can have different flag settings ok. And we can we can have this comparison instruction for the arithmetic operation. For immediate operands so again the two thing you can have an 8-bit data or you can have a 16-bit data.

So, for 8-bit data, so we can have instruction like compare AL comma data 8, so that way you can have the 8-bit data there. So, again the operation is similar. So, AL minus data 8 will be done. If AL is greater than data 8, then the flags will be set in some fashion and the otherwise the flags are set in different fashion so as it is specified here. So, if this if you are specifying this AX, then you can have a 16-bit comparison also that way it is very much flexible both 8-bit and 16-bit comparison can be done.

(Refer Slide Time: 20:16)

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

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

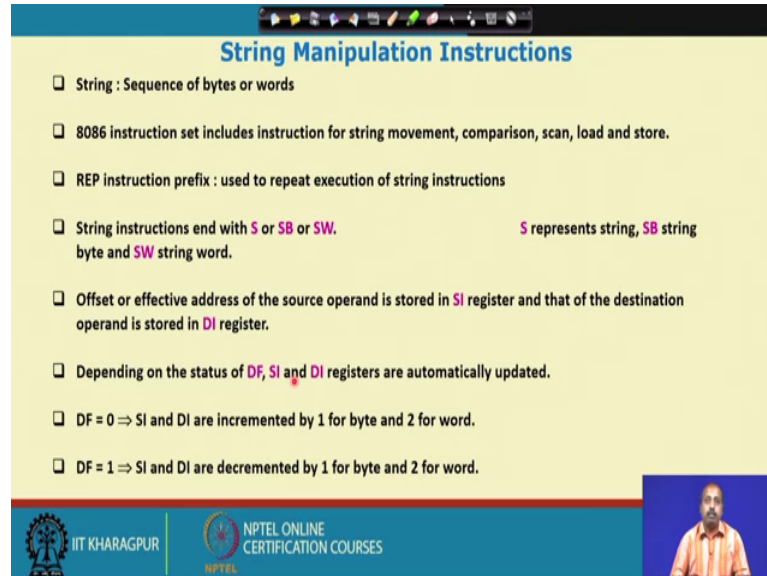
The slide also features the IIT KHARAGPUR logo, NPTEL ONLINE CERTIFICATION COURSES logo, and a small video inset of a presenter in the bottom right corner.

Next, we will look into the logical instruction. So, logical instructions were we are doing the logic logical operation like AND, OR, XOR testing or shifting and all that. So, the first one the AND operation, so AND a comma data so this will do ANDing of a with the AL register with some data 8 like say 8-bit data. So, AL will get AL and ANDed with the 8-bit data that the under the immediate mode or 16-bit data under the immediate mode AX and data 8.

So, for OR instruction, so again the same thing we can have this registered 2 or memory or we can have it like this registered two memories of they will be odd these two register contents will be odd, and the result will be available in register 2. So, you can have this that immediate data also like you can write like OR register memory comma data. So, you can say like or say AX comma some 16-bit data, so that 16-bit data will be OR with this register bit by bit, and the result will be available in the AX register. So, this way we can have this OR with 16-bit data. Or if you are directly writing so instead of writing

register, so you can also take this value to AL register and CX AX registers for doing the 8-bit operations of this OR.

(Refer Slide Time: 21:44)



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.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The next important category of instructions that we have that is the string manipulation instructions. So, this is a new instruction that type that we category that we have in 8086. So, string is any sequence of bytes or word. So, unlike say what we know about a string that is a character string or things like that. So, in case of 8086 processors, so it cannot distinguish between a numbers and characters ok, so that is why it is any sequence of bytes or words so that is taken as a string. So, this instruction set it includes instructions for string movement comparison, scan, load and store. So, these are the various string manipulation instructions that are available in the 8086 processor.

There is one REP instruction prefix repeat instruction prefix. So, it is used to repeat execution of string instruction. So, if you say REP then the instruction will be repeated, execution of the instruction will be repeated. Then string instructions that end with S, SB or SW. So, S like MOV is the normal movement between registers, MOVS is the string movement that way and MOVS B is the bytes string MOV byte movement and MOVS W is the word string movement. So, we can have this moves MOVS B, MOVS W like that, then CMPs, CMPs B, CMPs W like that.

Offset or effective address of the source operand is in SI register and that of the destination operand is the is in the DI register. So, we have got the SI DI register pair, so

which are holding the source and destination addresses depending upon these status of this direction flag SI and DI registers are automatically updated. So, after every byte or word transfer this SI and DI registers, so they will be updated automatically. So, DF flag if it is so that will tell us the direction, and based on that it will be updated.

If DF equal to 0, then this SI and DI, they are incremented by one by one for byte and 2 for word. So, if it is MOVSB type of instruction then after doing one transfer this SI and DI will be incremented by 1. If it is MOVSD type of instruction, then it will be updating by two incrementing the value by 2. And if DF equal to 1, then instead of incrementing the values will be decremented ok, so this way we can have this string manipulation instruction.

(Refer Slide Time: 24:15)

String Manipulation Instructions
Mnemonics: **REP, MOVS, CMPS, SCAS, LODS, STOS**

<p>REP</p> <p>REPZ/ REPE (Repeat CMPS or SCAS until ZF = 0)</p>	<p>While CX ≠ 0 and ZF = 1, repeat execution of string instruction and (CX) ← (CX) - 1</p>
<p>REPZ/ REPE (Repeat CMPS or SCAS until ZF = 0)</p> <p>REPNE/ REPZ (Repeat CMPS or SCAS until ZF = 1)</p>	<p>While CX ≠ 0 and ZF = 0, repeat execution of string instruction and (CX) ← (CX) - 1</p>

IIT KHARAGPUR
 NPTEL ONLINE CERTIFICATION COURSES

So, this REP prefix. So, this REP prefix can be if the REPE or the REPZ, So it is the repeat the compare CMPS or SCAS the scan instruction till Z zero flag equal to ZF flag becomes equal to 0. So, while CX not equal to 0, and ZF equal to 1, so it will repeat the string instruction. So, and after that after every transfer, this CX bit will be updated. So, CX will be updated to CX minus 1. So, this using this repeat construct, so we can we can just repeat prefix. So, we can repeat this compare or scan instructions similarly REPZ so repeat this CMPS or this SCAS instruction until this 0 flag is equal to 1. So, until ZF equal to 1, so it will on repeating the operation and after every execution, so it will decrement the CX register value by one in both the cases.

(Refer Slide Time: 25:19)

String Manipulation Instructions
Mnemonics: **REP, MOVS, CMPS, SCAS, LODS, STOS**

MOVSB	$MA = (DS) \times 16_{10} + (SI)$ $MA_E = (ES) \times 16_{10} + (DI)$ $(MA_E) \leftarrow (MA)$ 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	$MA = (DS) \times 16_{10} + (SI)$ $MA_E = (ES) \times 16_{10} + (DI)$ $(MA_E; MA_E + 1) \leftarrow (MA; MA + 1)$ 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

So, the MOVS instruction, so we have got two variants MOVS B and MOVS W. So, for the MOVS B instruction what happens is that the memory address is computed as DS into 16 plus SI for the source part for the destination part it is MA E which is ES into sixteen plus DI then ma the content of memory location MA is transferred to the memory location M MA E. Now, if DF equal to 0, then after doing this transfer. So, DI value is incremented by 1, and SI value is decremented by 1. And if DF is equal to 1, then DI and SI those values are decremented similarly this MOVS W instruction. So, this is similar to MOVS b, but this incrementing and decrementing. So, So, or two bytes will be transferred like this MA E will get the content of memory location MA; and in memory location MA E plus 1 will get the content of memory location ma plus 1. And after that this DI and SI values they are decremented or incremented by 2. So, this way this MOVS instruction can be useful.

(Refer Slide Time: 26:32)

String Manipulation Instructions
Mnemonics: **REP, MOVS, CMPS, SCAS, LODS, STOS**

Compare two string byte or string word

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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

We have got CMPS or compare instruction. So, this CMPS instruction also has got two variants compare CMPS B and CMPS W. So, CMPS B the address calculations are same as the MOVS instruction. So, this MA and MA E values are calculated. And then the ASIC key values of those locations are compared. So, if ma value content of memory location MA is greater than content of memory location MA E then this will be the flag setting. If MA is content of memory location MA is less than content of memory location MA E then this will be the flag settings. So, this you have got different flag settings and. And then for the byte operation this DI values will be updated by one and for by word operation they are updated by two. And depending upon the DF bit, so the this updation may be an addition or updation may be a subtraction. If DF equal to 0, the value may be incremented DI and SI values may be incremented, and if DF equal to 1 in this DI and SI values may be decremented. So, this way we can compare between byte string byte or string word.

(Refer Slide Time: 27:50)

String Manipulation Instructions
Mnemonics: **REP, MOVS, CMPS, SCAS, LODS, STOS**
Scan (compare) a string byte or word with accumulator

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

Handwritten note: $AX - (MA_i; MA_i + 1)$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 77

Then there is a scan instructions. So, scan a string byte or word with accumulator. So, SCASB. So, SCASB what is it doing so here it is only this destination address is taken. So, ES colon DI, so that address is computed, so that is MA E and then it will be compared with the AL register. So, it is not compared with any other source block, but it is compared with the AL register. So, basically in the AL register, you can put a particular character, and you can try to scan for that character into the into a string, so that is very common of common operation to find the occurrence of a character in a string, so that can be done by using this SCAS B instructions.

So, if AL is greater than MA E then this is the settings of the comparison standard comparison. So, if they are same, if the character are character matches with the AL, then this zero flag will be same. So, you can compare this and get the matching. And if DF equal to 0, then DI will be incremented if DF equal to 1, DI will be decremented.

And we also have a 16-bit comparison. So, SCASW. So, here the this AL, so this is AL, so this is AX will be compared in fact, with this AX and MA E. And then if AX is greater than MA E, and MA E plus 1, so then this flags will be set ok. So, this content should be ah. So, this one should be this should be AX minus MA E colon MA E plus 1. So, these two values should be compared. And if AX is greater than this quantity then the flag setting will be like this; otherwise the flag settings will be the otherwise. So, this way

you can have this SCAS instruction for doing this memory operation this string operation comparison and all ok.