

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

Lecture – 64
8086 Microprocessor
(Contd.)

(Refer Slide Time: 00:17)

Bus Interface Unit (BIU)

Segment Registers

Data Segment Register

- 16-bit
- Points to the current data segment; operands for most instructions are fetched from this segment.
- The 16-bit contents of the Source Index (SI) or Destination Index (DI) or a 16-bit displacement are used as offset for computing the 20-bit physical address.

20
 $16 * DS + SI \rightarrow 20$
 $16 * DS + DI$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

For the data segment register so we have got the DS register. So, that is also a 16 bit register and this data segment register, so this is used for accessing data and we have got this for the offset part. So, this registers this SI and DI registers so they actually act as the offset part. So, here the calculation will be like 16 into DS plus SI or 16 into DS plus DI. So, that way we can have this 20 bit address computed. So, when I say multi so this multiplied by 16 means it is left shifted by 4 bit. So, this result becomes 20 bit result and with that another 16 bit value is added, so overall a value is becoming 20 bit. So, this way we can these 8086 accesses the data part.

(Refer Slide Time: 01:08)

Bus Interface Unit (BIU)

Segment Registers

Stack Segment Register

- 16-bit
- Points to the current stack.
- The 20-bit physical stack address is calculated from the Stack Segment (SS) and the Stack Pointer (SP) for stack instructions such as PUSH and POP.
- In based addressing mode, the 20-bit physical stack address is calculated from the Stack segment (SS) and the Base Pointer (BP).

$16 \times SS + SP$
 $16 \times SS + BP$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

For stack part so it uses this stack segment register SS and the otherwise the calculation is same, so there are 2 registers which are dedicated for accessing the offset which is this the stack one is the stack pointer and another is the base pointer. So, both of them the calculation will be that 16 into SS plus SP or 16 into SS plus BP, so that way it accesses the 20 bit address for the within the stack segment. So, and there is another segment register which is extra segment.

(Refer Slide Time: 01:52)

Bus Interface Unit (BIU)

Segment Registers

Extra Segment Register

- 16-bit
- Points to the extra segment in which data (in excess of 64K pointed to by the DS) is stored.
- String instructions use the ES and DI to determine the 20-bit physical address for the destination.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, many times what happens is that we have the 1 data segment is not sufficient, so we need to simultaneously access 2 data segments. So, for that purpose another data segment has been provided by this ES register extra segment register. So, here also operation can be done plus this ES DI pair, so that also helps in the string operation. So, we will see that 8086 has got a powerful string operation so, for this string operation this is going to be useful.

(Refer Slide Time: 02:23)

Segment Registers

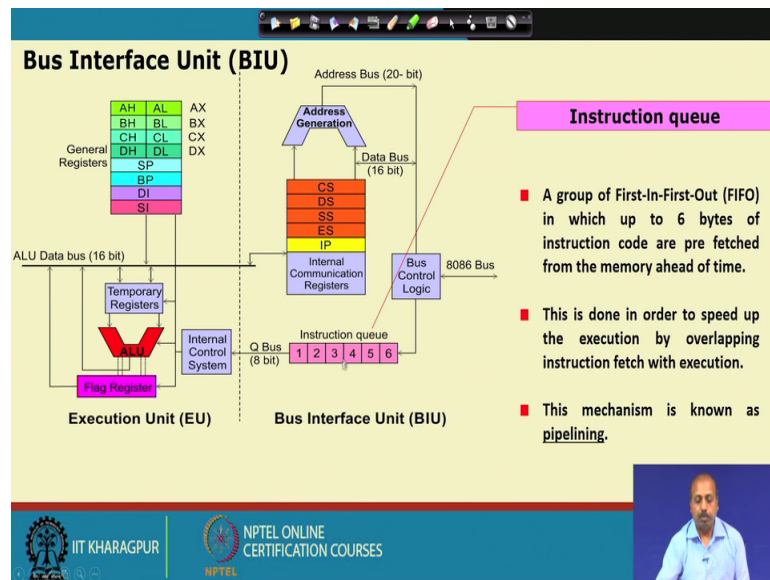
Instruction Pointer (IP)

- 16-bit
- Always points to the next instruction to be executed within the currently executing code segment.
- So, this register contains the 16-bit offset address pointing to the next instruction code within the 64Kb of the code segment area.
- Its content is automatically incremented as the execution of the next instruction takes place.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then this instruction pointer so that is a 16 bit register and instruction pointer or IP. So, this is written as IP so this points to the next instruction to be executed within the current code segment. So, this IP value is a 16 bit value so, that will be incremented after each memory access and that way it will go. So, this 16 into CS plus IP so that way the next addresses will be calculated.

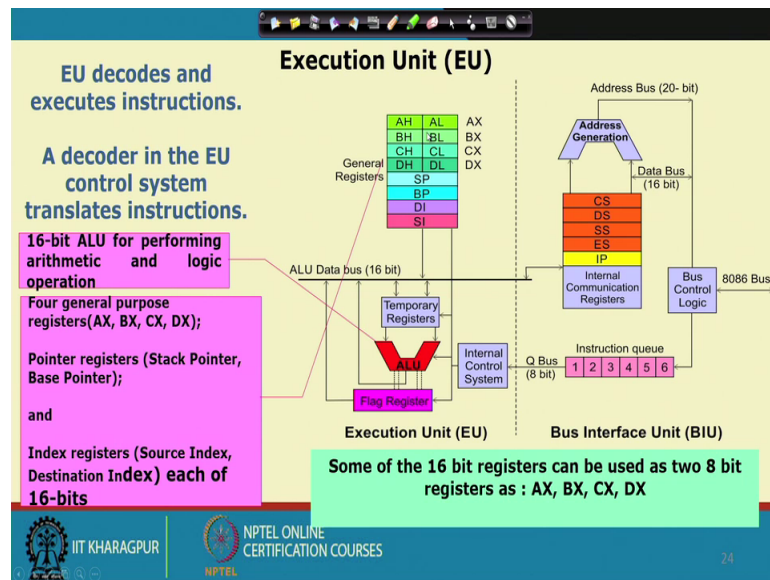
(Refer Slide Time: 02:53)



Next we will come to this instruction queue, so this is a group of first in first out in which we have got up to 6 bytes of instruction code are pre fetched from the memory ahead of time. So, as I said that whenever this bus is free this bus interface unit so it will fetch the successive bytes from memory. So, we have got instructions in 8086 which varies from 1 byte to 6 bytes, so that way this 6 byte instruction queue has been provided, so that those instructions can be pre fetched.

So, this is done in order to speed up the execution of overlapping instruction fetch with execution. So, as I was telling over left phase execution and this mechanism is also known as pipelining. So, this execution fetch and execution they are pipelined.

(Refer Slide Time: 03:43)



Next we look into the execution unit part, so this execution unit it decodes and executes the instruction so that is the major responsibility. A decoder in the execution unit control the execution unit control system translates instruction, so this is this control system. So, this is translating the instructions into the control signals to see how this instructions can be executed.

There is the 16 bit ALU which perform the arithmetic and logic operations, 4 general purpose registers AX BX CX DX, there are 2 pointer registers stack pointer and base pointer and there are 2 index registers SI and DI ok. So, these are the several registers that we have in the execution unit, some of the 16 bit registers can be used as 2 8 bit registers like this AX can be taken as and AL 2 8 bit registers, BX can be taken as BH BL.

(Refer Slide Time: 04:36)

The slide is titled "EU Registers" and "Accumulator Register (AX)". It contains a bulleted list of four points. The footer includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and the number 25.

EU Registers

Accumulator Register (AX)

- Consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX.
- AL in this case contains the low order byte of the word, and AH contains the high-order byte.
- The I/O instructions use the AX or AL for inputting / outputting 16 or 8 bit data to or from an I/O port.
- Multiplication and Division instructions also use the AX or AL.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 25

So, like that so this AX is the accumulator, so just like 8085 we had A, so here we have got the AX register. It consist of 2 8 bit registers AL and which can be combined together and used as a 16 bit register AX, AL in this case contains low order byte of the word and contains the higher order byte. The I O instructions that use AX and AL for inputting or outputting 16 or 8 bit data to or from an I O port. So, this is for I O access and multiplication division instructions also use AX or AL.

(Refer Slide Time: 05:13)

The slide is titled "EU Registers" and "Base Register (BX)". It contains a bulleted list of four points. The footer includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and the number 26.

EU Registers

Base Register (BX)

- Consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX.
- BL in this case contains the low-order byte of the word, and BH contains the high-order byte.
- This is the only general purpose register whose contents can be used for addressing the 8086 memory.
- All memory references utilizing this register content for addressing use DS as the default segment register.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 26

So, we have got multiplication division operation that use either AX or AL. Then there is another, the next register BX, so this is called base register so otherwise this is a general purpose register. So you can use it for other operations like so, holding some data operand performing addition subtraction operation, so for that you can hold it as some another operands there.

So, again BL is the lower order byte BH is the higher order byte, so this is the only general purpose register that can be used for addressing the 8086 memory. So, that is why it is called the base register, so particularly when you are accessing say array type of locations, so this BX register can be used as to hold the base address. So, we will see that later or memory references utilizing this register content for addressing use DS as the segment register. So, DS is the segment register and BS BX will be the offset, so like that it will be done.

(Refer Slide Time: 06:11)

EU Registers

START: CX ← 10
LOOP START

Counter Register (CX)

- Consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX.
- When combined, CL register contains the low order byte of the word, and CH contains the high-order byte.
- Instructions such as **SHIFT**, **ROTATE** and **LOOP** use the contents of CX as a counter.

Example:

The instruction **LOOP START** automatically decrements CX by 1 without affecting flags and will check if [CX] = 0.

If it is zero, 8086 executes the next instruction; otherwise the 8086 branches to the label **START**.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

27

Then there is a, another register CX this is also general purpose register which is given the special name counter register. So, the name comes from the fact that for some instructions like shift rotate loop etc, so we have to tell like how many bits you want to shift or rotate or for a loop instructions. So, 8086 allows you to repeat a block of statements in number of times. So, how many times that block will be repeated that is determined by the CX register content.

So, this will be that is why the name counter comes here, so this CX is called the counter register the instruction loop starts. So, these automatically decrement CX by 1 without affecting flags and we will check if CX equal to 0. So, these so loop start when whenever this instruction comes. So, the control start is actually a level. So, it will be going back to that start point and it will again start the execution of the loop. So, normally what we do is that if we want to make a loop, so we keep this body and. So, this is the level start and at this point we say loop start and before that the somehow the CX registers initialize to number of times we want to repeat the loop.

Suppose we want to repeat it 10 times, so it will be when it comes here. So, CX value will be decremented if it is nonzero, so control is automatically branch to this. So, we do not have to spend any more instruction for that. So, if it is 0 so it will be the 8086 will execute the next instruction otherwise it will branch a to the label start.

(Refer Slide Time: 07:46)

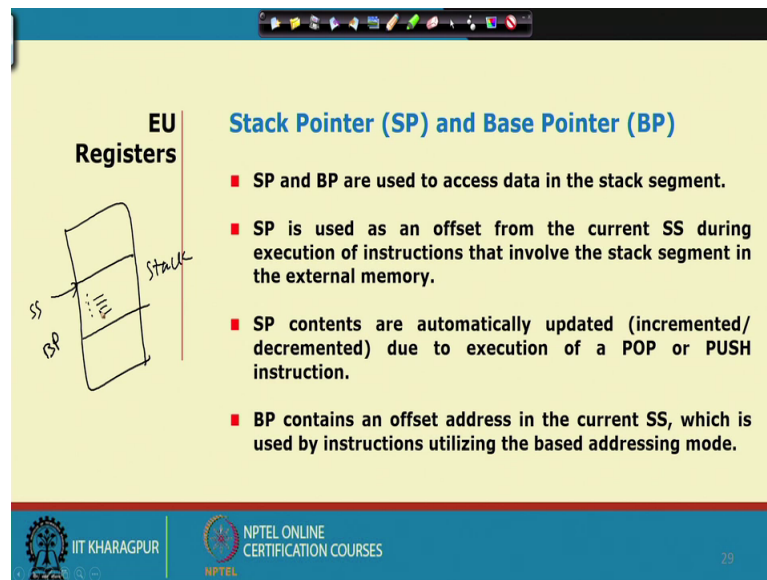
The slide is titled "8086 Microprocessor" and is part of an NPTEL online certification course from IIT Kharagpur. It focuses on the "EU Registers", specifically the "Data Register (DX)". The slide lists three key points about the DX register:

- Consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX.
- When combined, DL register contains the low order byte of the word, and DH contains the high-order byte.
- Used to hold the high 16-bit result (data) in 16 X 16 multiplication or the high 16-bit dividend (data) before a 32 ÷ 16 division and the 16-bit remainder after division.

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

Then there is a data register DX so this is this is for holding data. So, this does not have a other special functions. So, it is DL it can be divided into DL and DH it is used to hold the high 16 bit result in 16 into 16 by 16 multiplication or the high 16 bit dividend before a 32 by 16 division and the 16 bit remainder after a division. So, this is particularly used for this data operation that is why the name data register has come.

(Refer Slide Time: 08:16)



EU Registers

Stack Pointer (SP) and Base Pointer (BP)

- SP and BP are used to access data in the stack segment.
- SP is used as an offset from the current SS during execution of instructions that involve the stack segment in the external memory.
- SP contents are automatically updated (incremented/decremented) due to execution of a POP or PUSH instruction.
- BP contains an offset address in the current SS, which is used by instructions utilizing the based addressing mode.

The diagram shows a vertical rectangle representing the stack segment. The top edge is labeled 'SS' (Stack Segment register). The bottom edge is labeled 'BP' (Base Pointer register). The word 'Stack' is written vertically on the right side of the rectangle. Inside the rectangle, there are several small circles representing data elements, with a vertical ellipsis indicating they continue.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 29

Other 2 registers are stack pointer and base pointer, so this stack pointer is they are used to access both SP and BP they are used to access the stack segment SP is used as an offset from the current stack segment during execution of instructions that involved stack segment in the external memory. So, basically the local variables the parameters that you pass, so for the accessing those this stack pointer is used. And then this stack pointer this SP BP content so, they will automatically be updated are implemented or decremented for a POP or PUSH type of operations. And BP is a special register sometimes using stack pointer alone is not sufficient because stack pointer may get corrupted.

So, we if we want to access within the stack segment then this BP pointer may be used, so what I say is like this that is if this is the memory and maybe so this party is the stack so this whole thing is the stack. So, your stack segment registers points to this and now to access the individual locations within this. So, one option is you can use the stack pointer to do that. But the stack pointer if it gets corrupted then this return address will be a problem so return address may get maybe modified.

So, what we do instead of that stack pointer, the base pointer register is used to get offsets within this and stack pointer is not touched, so that this procedure call return can walk properly but for accessing parameters and all we can use this base pointer. So, both of them will use stack segment as the base register.

(Refer Slide Time: 10:05)

The slide is titled "EU Registers" and "Source Index (SI) and Destination Index (DI)". It lists two main points:

- Used in indexed addressing.
- Instructions that process data strings use the SI and DI registers together with DS and ES respectively in order to distinguish between the source and destination addresses.

The slide footer includes the IIT Kharagpur logo, NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a presenter.

So, and another pair of registers which are common in 8086 is a source index SI and destination index DI. So, they are used for indexed addressing and also for the string processing. So, this SI and DI purposes so we will see later when we go for this string operations.

(Refer Slide Time: 10:26)

The slide is titled "Flag Register" and shows a 16-bit register with bits 15 down to 0. The flags are:

- Sign Flag (SF)**: Bit 7. This flag is set, when the result of any computation is negative.
- Zero Flag (ZF)**: Bit 6. This flag is set, if the result of the computation or comparison performed by an instruction is zero.
- Auxiliary Carry Flag (AF)**: Bit 5. This is set, if there is a carry from the lowest nibble, i.e., bit three during addition, or borrow for the lowest nibble, i.e., bit three, during subtraction.
- Carry Flag (CF)**: Bit 0. This flag is set, when there is a carry out of MSB in case of addition or a borrow in case of subtraction.
- Parity Flag (PF)**: Bit 2. This flag is set to 1, if the lower byte of the result contains even number of 1's; for odd number of 1's set to zero.
- Over flow Flag (OF)**: Bit 11. This flag is set, if an overflow occurs, i.e., if the result of a signed operation is large enough to accommodate in a destination register. The result is of more than 7-bits in size in case of 8-bit signed operation and more than 15-bits in size in case of 16-bit sign operations, then the overflow will be set.
- Trap Flag (TF)**: Bit 8. If this flag is set, the processor enters the single step execution mode by generating internal interrupts after the execution of each instruction.
- Direction Flag (DF)**: Bit 10. This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address, i.e., auto incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e., auto decrementing mode.
- Interrupt Flag (IF)**: Bit 9. Causes the 8086 to recognize external mask interrupts; clearing IF disables these interrupts.

The slide footer includes the IIT Kharagpur logo, NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a presenter.

Flag registers so there are a number of flag bits like this carry bit, so carry flag is there this. So, this carry then we have got parity then there is auxiliary carry, then there is 0 then we have got this sign flag then this trap flag. So, this is this is it can generate some

internal interrupt, so this is mostly used for debugging type of operation. Then there is a interrupt flag so it will cause it will cause the 8086 to recognize the external mask interrupts.

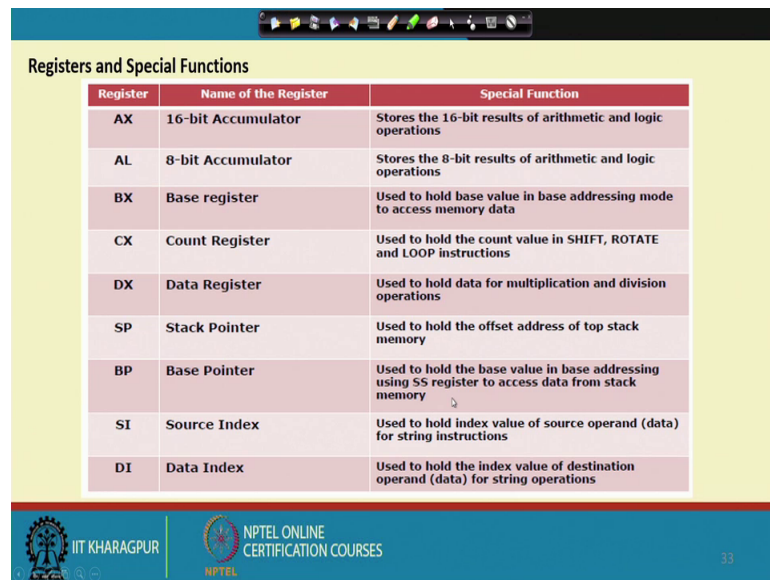
Then if you if you clear this if then it will disable all the interrupts, so that is the we as the same thing that we have we say rim SIM type of instructions in 8085 then there is a DF flag with the direction. So, direction flag it will it will be clear when you go to the string instructions and there is an overflow. So, if an over flow occurs then this flag will be safe, so these are various flag registers that we have in 8086.

(Refer Slide Time: 11:23)

Si.No.	Type	Register width	Name of register
1	General purpose register	16 bit	AX, BX, CX, DX
		8 bit	AL, AH, BL, BH, CL, CH, DL, DH
2	Pointer register	16 bit	SP, BP
3	Index register	16 bit	SI, DI
4	Instruction Pointer	16 bit	IP
5	Segment register	16 bit	CS, DS, SS, ES
6	Flag (PSW)	16 bit	Flag register

Now, this overall registers so they can be classified into a number of groups like general purpose registers, we have got a 16 bit or 8 bit 16 bit AX BX CX DX 8 bit AL BL etc. We have got pointer registers SP and BP we have got index registers SI DI instruction pointer IP segment registers we have got this CS DS ES SS as 16 bit segment registers and there is a flag which is flag register which is again 16 bit.

(Refer Slide Time: 11:55)

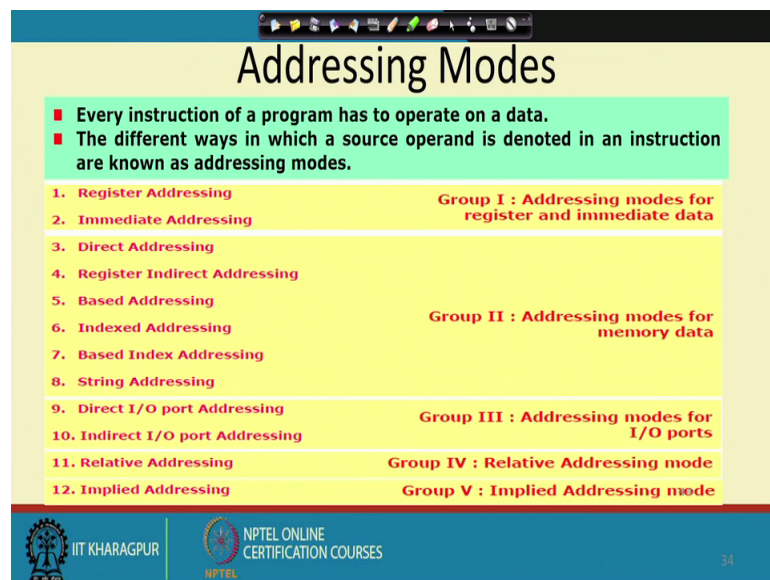


The slide displays a table titled "Registers and Special Functions" with three columns: Register, Name of the Register, and Special Function. The table lists ten registers: AX (16-bit Accumulator), AL (8-bit Accumulator), BX (Base register), CX (Count Register), DX (Data Register), SP (Stack Pointer), BP (Base Pointer), SI (Source Index), and DI (Data Index). Each register's special function is described, such as AX storing 16-bit results of arithmetic and logic operations, and SP holding the offset address of the top stack memory.

Register	Name of the Register	Special Function
AX	16-bit Accumulator	Stores the 16-bit results of arithmetic and logic operations
AL	8-bit Accumulator	Stores the 8-bit results of arithmetic and logic operations
BX	Base register	Used to hold base value in base addressing mode to access memory data
CX	Count Register	Used to hold the count value in SHIFT, ROTATE and LOOP instructions
DX	Data Register	Used to hold data for multiplication and division operations
SP	Stack Pointer	Used to hold the offset address of top stack memory
BP	Base Pointer	Used to hold the base value in base addressing using SS register to access data from stack memory
SI	Source Index	Used to hold index value of source operand (data) for string instructions
DI	Data Index	Used to hold the index value of destination operand (data) for string operations

So, this summarizes the operations, so we have already discussed about them next we come to the addressing mode.

(Refer Slide Time: 12:00)



The slide is titled "Addressing Modes" and lists 12 different modes. It is organized into five groups: Group I (Register and Immediate Data), Group II (Memory Data), Group III (I/O ports), Group IV (Relative Addressing mode), and Group V (Implied Addressing mode).

Addressing Mode	Group
1. Register Addressing	Group I : Addressing modes for register and immediate data
2. Immediate Addressing	
3. Direct Addressing	Group II : Addressing modes for memory data
4. Register Indirect Addressing	
5. Based Addressing	
6. Indexed Addressing	
7. Based Index Addressing	
8. String Addressing	Group III : Addressing modes for I/O ports
9. Direct I/O port Addressing	
10. Indirect I/O port Addressing	Group IV : Relative Addressing mode
11. Relative Addressing	
12. Implied Addressing	Group V : Implied Addressing mode

So, there are different addressing modes.

(Refer Slide Time: 12:08)

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

$(CL) \leftarrow (DH)$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Which are possible way in 8086 like the first addressing mode is the register addressing mode. So, the instruction will specify the name of the register which will hold the operate data to be operated by the instruction like MOV CL comma DH. So, the operands of the values are available in the registers only CL will get the content of DH. So, this is the content of 8 bit register DH is moved to another 8 bit register CL.

(Refer Slide Time: 12:33)

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 there is a immediate addressing when the where the source is an immediate 1 immediate value, like MOV DL comma 0 8 H. So, this 0 8 H this immediate value is

moved to the DL register in 8085 we had MBI type of instructions. So, here we do not have MBI so here it is MOV only. So, operand itself will identify that this is an immediate operand then MOV AX comma 0A9FH. So, this will be telling this is telling that we want to move this number to AX 16 bit number to AX register out of which this will get 0A and AL will get 9F.

(Refer Slide Time: 13:13)

Addressing Modes : Memory Access

- 20 Address lines \Rightarrow 8086 can address up to $2^{20} = 1\text{M}$ 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_{16}), then add the required offset to form the 20-bit address

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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

38

Then for memory access there are 20 address lines, so 8086 can address up to 2^{20} 1 mega bytes of memory. However, the largest register is only 16 bit. So, physical address that is formed so that is calculated as actual address of a byte in the memory, that is a values which will go to the address bus. So, this is in the form segment colon offset so this is the segment and this is the offset part.

So, what how is it so the that means, so, it will be starting the segment register will contain the value 89AB and the offset register will contain F012. And the calculation will be done like this; that this segment register value will be left shifted by 4 bits. So, in hexadecimal notation it becomes 89AB0 and with that this F012 will be added. So, this is the absolute address that is put on to the bus. So, as I was telling the value calculated is 16 into segment register plus offset 16 into segment plus offset, so that calculation is being done here.

(Refer Slide Time: 14:15)

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]`

$BX \leftarrow (16 * DS + 1354H)$

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.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, next will be looking into direct addressing, so in direct addressing the address is mentioned directly, so this MOV BX comma 1354 H. So, this 1354 denotes the content contents of the location to be accessed. So, this is when executed the instruction will copy the content of the memory location into BX register and the actual operation the memory contain memory the BX will get the content of memory location 16 into DS plus 1354 H.

So, content of this particular memory location will come to the BX register. Similarly, this BL 0400 so 16 into DS plus 0400 so that will come, so we can this addressing mode is called direct. Because, the displacement part of the operand from the segment base is specified directly in the instruction so, it is given here directly.

(Refer Slide Time: 15:18)

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:

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

(CX) ← (MA) or,

(CL) ← (MA)

(CH) ← (MA + 1)

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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then register indirect addressing mode, so here with the of the offset part is contained in some register like say MOV CX comma within bracket BX. So, BX content will be used as the offset, so this effective address is the content of the BX register and this is the memory will be this base address that is DS register multiplied by 16 plus this effective address. So, this way so this way will be calculating in that total actual address value. So, in this instruction so CX will get the content of memory location MA which is computed the address is computed here memory address or it is equivalent to this (Refer Time: 16:00), so CL getting MA and CH getting MA plus 1.

(Refer Slide Time: 16:04)

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: MOV AX, [BX + 08H]

Operations:

$$0008_H \leftarrow 08_H \text{ (Sign extended)}$$
$$EA = (BX) + 0008_H$$
$$BA = (DS) \times 16_{10}$$
$$MA = BA + EA$$

(AX) ← (MA) or,

(AL) ← (MA), (AH) ← (MA + 1)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then there is a based addressing mode so where these base register BX can be used to give the offset part. So, this BX will hold this effective address which is this 20 bit physical and 20 bit physical address will be calculated from BX and DS like here. So, BX with this BX this 0 8 H will be added ok. So, this BX plus 0008 H is added that gives the effective address and base address is DS into 16 and the actual memory address is MA plus BA plus EA, so that is giving the memory address. So, then this AX register gets the content of memory MA or it is equivalent to this, so this way this base register BX is used for based addressing.

(Refer Slide Time: 16:51)

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,
 $(CL) \leftarrow (MA)$
 $(CH) \leftarrow (MA + 1)$

The slide footer includes the IIT KHARAGPUR logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'.

Then we have got indexed addressing, so in this indexed addressing this index register SI or DI they can be used to hold the index value of the memory location. So, this MOV CX comma SI plus 0A2H, so this will be containing this ea the effective address will be SI plus this offset. So, this 0A2H so when it is sign extended so it will become FFA2H, so that will get added and this will be multiplied base address will be multiplied by 16 DS multiplied by 16 will give the base address and memory address is calculated like this. So, this is it is sign extended to 16 bit before adding to the base value in case of indexed addressing.

(Refer Slide Time: 17:34)

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



Then we have got this based indexed addressing where the base register and this index register. So, both are used like MOV DX comma BX plus SI plus 0AH. So, this is sign extended first of all this 0AH is sign extended, so it remains it become 0 0 0 A and effective address is BX plus SI plus 0 0 0 A and this base address is DS into 16 and actual memory address is BA plus EA. So, then the content of this memory address is transferred to the DX register that is the base addressing.

(Refer Slide Time: 18:07)

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



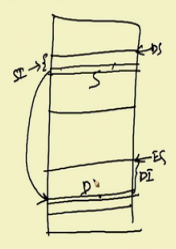
Example: MOVSB

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

Calculation of destination memory location:
 $EA_e = (DI)$ $BA_e = (ES) \times 16_{10}$ $MA_e = BA_e + EA_e$

$(MA_e) \leftarrow (MA)$

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



Then there is 1 string addressing so this is employed in string operations to operate on string data. So, we have got in 8086 instructions called like MOVS type of instruction, so here we have got 1 example MOVS byte. So, here the operation that is done the when I say MOVS; that means, we are trying to move a byte from 1 memory location to another memory location, without involving the without involving a transferred to a processor register.

So, how is it done so this effective address, so for the source memory location; so this effective address is available in the source index registered SI and DS contains the base of that actually the idea is like this. So, if this is your memory and then if this is so this maybe the source block S and this is the destination block D and we may be interested to transfer the content of this memory location from here to say here we want to transfer it to here.

Now, how to do that so DS register points to this ES register points to this address and then this SI contains this offset. So, SI is SI contains this difference and DI contains this difference ok. So, when this MOVS instruction is executed first the content of this memory location is taken so 16 into DS plus SI. So, that way gives the memory address source memory address and for the destination address, so this is 16 into ES plus DI so that is done. So, that that becomes the destination address then this locations content is transferred to this location and then depending upon the direction flag setting.

So, DF is made equal to if DF is equal to 1 then this SI and DI are decremented by 1, if DF equal to 0 they are incremented by 1 SI and DI are incremented by 1. So, this is the string addressing so we will be looking into instructions later, but these instructions are useful like when you are trying to transfer a chunk of memory block from some memory locations to some other memory location this can be used.

(Refer Slide Time: 20:37)

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.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

For I O port addressing so there are 3 addressing these addressing modes can I access standard I O mapped devices or ports. So, you can have in AL comma 0 9H. So, this is a 68 bit port address, so this is the direct port addressing. So, you can mention 8 bit port address like here 0 9 is the port address. So, this AL gets the content of the port whose address is 0 9 H and in indirect port addressing mode.

So, this port address is 16 bit address and that 16 bit address should be available in some register. So, for example, it may be 16 bit port address so it is stored in the DX register in that case, so we can say like out DX comma AX so that means, then the content of this AX register will be outputted to the port whose address is available in the DX register ok. So, this way we can go for this I O port addressing.

(Refer Slide Time: 21:36)

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

Operations:

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

If ZF = 1, then

$EA = (IP) + 000A_H$
 $BA = (CS) \times 16_{10}$
 $MA = BA + EA$

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

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

The diagram shows a horizontal line representing memory. A point on the line is labeled 'IP' with the value '10000' written to its right. A curved arrow points from the IP to a point below the line, which is labeled 'EA' with the value '50000' written below it. Another arrow points from the EA to a point on the line, which is labeled 'MA' with the value '10000' written to its left.

Then there is a relative addressing so in this addressing mode the effective address of a program instruction is specified relative to the instruction pointer by an 8 bit signed displacement. So, what is done suppose so this is in relative addressing is in relation to the jumping within the program, like if we want to jump from say suppose we have written a program, where we have got we have got these statements and at this point I from this point I want to jump to this address ok.

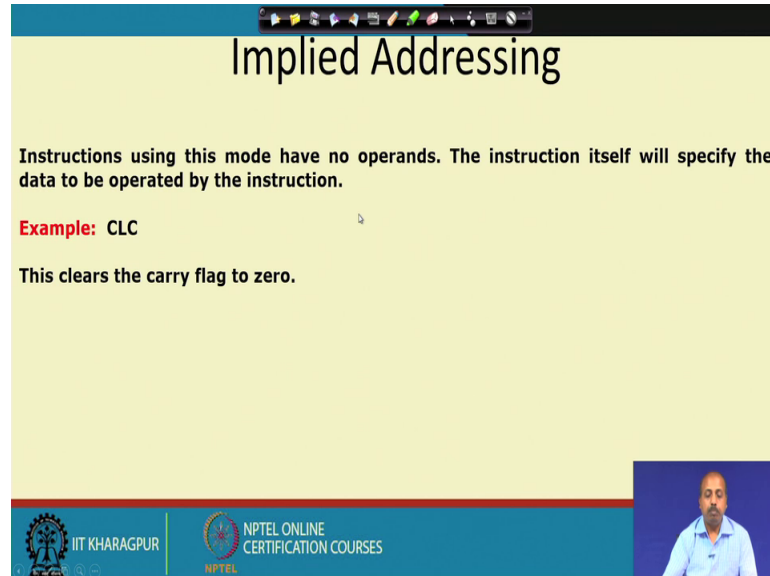
So, then one possibility this address is say 10000, so I can directly say here jump 10000, but the problem with this type of a concept is that if it is next time the program is loaded from a different address. Then this 10000 may not the address may not be valid; maybe this should actually be say 50000 if the program is loaded from a different address maybe this address should be 50000.

So, just to avoid that situation so in case of relative jump what we do we tell that distance between these 2 points, like here so we say there is a jump on 0 if the 0 flag is set then it will jump to 0A H. But, what is actually done is with the current instruction pointer value this 0A H will be added and when this addition is done then when this addition is done.

So, we have got this new address calculated and then wherever the program is loaded this distance between these 2 points always remain like 0A H ok. So, that way this distance is unchanged so we do not have any problem there. So, it is irrespectively it is it is not dependent on the address at which the program is loading. So, there in case of

8086 we have got this relative addressing and this relative addressing will help us in making the program relocated.

(Refer Slide Time: 23:39)



Implied Addressing

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

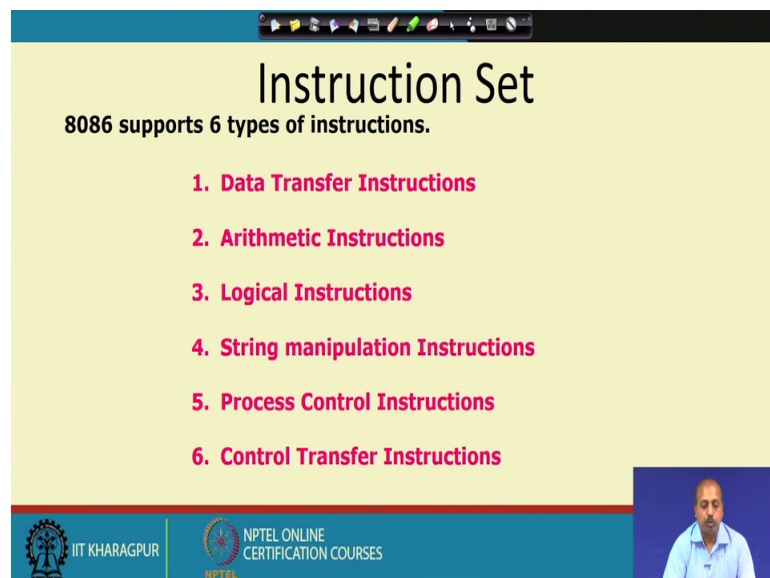
Example: CLC

This clears the carry flag to zero.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then there are implied addressing where you do not need to tell the operand like this CLC. So, clear carry so clear carry means the carry flag will be cleared. So, the no other operand need to be specified because it is directly specified in the carry flag.

(Refer Slide Time: 23:58)



Instruction Set

8086 supports 6 types of instructions.

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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we will looking into the instruction set there are 6 types of instructions supported in 8086, data transfer instruction arithmetic instruction logical instruction string manipulation process control and control transfer.

(Refer Slide Time: 24:11)

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 a 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

So, data transfer instructions so the instructions that are used to transfer data or address into registers memory locations and I O ports, so they come under this data transfer instruction. So, we have so whenever we have got data transfer we need to tell the source and the destination; so, the source operand and destination operand so, they are of same size. So, source may be register or memory location or an immediate data destination has to be a register or a memory location, so naturally immediate cannot be the destination.

So, we have to have some memory location or a register and we cannot have both the operands as memory. So, it is possible that both the operands of this data transfer instructions their memory expecting that string type of operation, so it is not possible. So, size should be either a byte or a word and the 8 bit data can be moved only to 8 bit register or memory say and 16 bit data can be moved to only 16 bit register or memory.

(Refer Slide Time: 25:10)

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)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, here we have got a few example like these are the common mnemonics that we have with the data transfer instructions MOV XCHG PUSH POP IN OUT etcetera; like we can have say MOV register 2 comma register 1, so register to MOV AX comma BX like that MOV memory comma register 1. So, we can specify some memory address and they have register 1 content is move there or this is from this memory location the value is loaded on to this register.

Then we have got this register this is this is for the immediate part. So, we have got some immediate data may be mentioned and that goes to register or that goes to memory location then we have got exchange. So, exchange means these two are these two contents are exchanged, so register 2 and register 1 are exchanged or this memory and register 1 are exchanged, so these are the MOV instructions.

(Refer Slide Time: 26:01)

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

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

Then the PUSH instruction so, we can have this PUSH register 16 bit register or PUSH memory location. So, if it is PUSH 16 bit register first the stack pointer is decremented address is calculated, so this is 16 into SS plus SP and then the this the 2 locations this location and the next location, so there it get the content of the 16 bit register. Similarly PUSH memory, so here also similar thing the content of this memory location and this next location total 16 bit location so that they move to stack. Similarly, the reverse of PUSH we have got POP instruction, so that can also work in conjunction is 16 bit register or memory.

(Refer Slide Time: 26:45)

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

IN A, [DX]		OUT [DX], A	
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		OUT addr8, A	
IN AL, addr8	$(AL) \leftarrow (addr8)$	OUT addr8, AL	$(addr8) \leftarrow (AL)$
IN AX, addr8	$(AX) \leftarrow (addr8)$	OUT addr8, AX	$(addr8) \leftarrow (AX)$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then this I operation like in operation in instruction, so IN AL comma DX; so, this is the DX register contain the 16 bit port address and that value comes to AL register. So, or you can move by 16 bit data can be moved, so by means of this AX ok. So, you can say like in AX comma with in bracket DX. So, that way the port 16 bit port value comes to the AX register or we can similarly we have got this if you are doing immediate addressing then this is the 8 bit address we can mention ok. So, this we have already seen similarly for the out instruction also we have got either 16 bit port or 8 bit port.

(Refer Slide Time: 27:26)

Arithmetic Instructions

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

<p>ADD reg2/ mem, reg1/mem</p> <p>ADD reg2, reg1 ADD reg2, mem ADD mem, reg1</p>	<p>$(reg2) \leftarrow (reg1) + (reg2)$ $(reg2) \leftarrow (reg2) + (mem)$ $(mem) \leftarrow (mem) + (reg1)$</p>
<p>ADD reg/mem, data</p> <p>ADD reg, data ADD mem, data</p>	<p>$(reg) \leftarrow (reg) + data$ $(mem) \leftarrow (mem) + data$</p>
<p>ADD A, data</p> <p>ADD AL, data8 ADD AX, data16</p>	<p>$(AL) \leftarrow (AL) + data8$ $(AX) \leftarrow (AX) + data16$</p>

Then under the arithmetic instruction category so we have got all this instruction add add with carry, subtract subtract with borrow increment, decrement, multiply, division compare etcetera. So, add is register to register 1 so this is register 2 will get register 1 plus register 2 add register 2 memory, so register 2 get register 2 plus memory. So, like that we can have a number or different types of add operation. Note one thing that there is nothing like add memory 1 comma memory 2, so both operands cannot be memory 1 of them must be a register.

Then these add register memory comma data, so here this is the immediate addition the ADI type of instruction that we had in 8085. So, this is this data value will be added to the register or here this data value will be added to the memory location, then add A comma data, so this is again the 8 bit addition. So, this way we can say AL AX so AL

sorry this is yeah this a comma data, so this 8 bit data can be added to AL or 16 bit data can be added to AX.

(Refer Slide Time: 28:33)

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

ADC reg2/ mem, reg1/mem	
ADC reg2, reg1	$(reg2) \leftarrow (reg1) + (reg2) + CF$
ADC reg2, mem	$(reg2) \leftarrow (reg2) + (mem) + CF$
ADC mem, reg1	$(mem) \leftarrow (mem) + (reg1) + CF$
ADC reg/mem, data	
ADC reg, data	$(reg) \leftarrow (reg) + data + CF$
ADC mem, data	$(mem) \leftarrow (mem) + data + CF$
ADDC A, data	
ADD AL, data8	$(AL) \leftarrow (AL) + data8 + CF$
ADD AX, data16	$(AX) \leftarrow (AX) + data16 + CF$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then this ADC add with carry so we have got a different types of add with carry instructions here so we can do that. So, otherwise it is same as the add instruction.