

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

Lecture – 64
8087

(Refer Slide Time: 00:28)

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

So, next we have got these loads instructions. So, you want to load some byte which is pointed to by this source indexed register and load that into the accumulated register. And that for that purpose; we have got this LODSB. So, LODSB instruction what it what is it doing it is that this DS into 16 plus SI so that memory address location, so it will be copied into AL register. And then if DF 0, then SI value will be incremented; if DF is 1 then SI value is decremented. So, if this LODSB is having that rep prefix the DP prefix then these operations will carried on till the CX becomes equal to 0 ok, so that way we have got this repeat this REP prefix useful for doing that for scanning through a list of characters a string of characters or loading a string of characters one after the other into the accumulator and doing some operation there. Similarly, we have got this LODSW. So, here also it is doing the same operation where it is loading into the AX register the loc the memory location content pointed to by MA and MA plus 1.

(Refer Slide Time: 01:38)

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

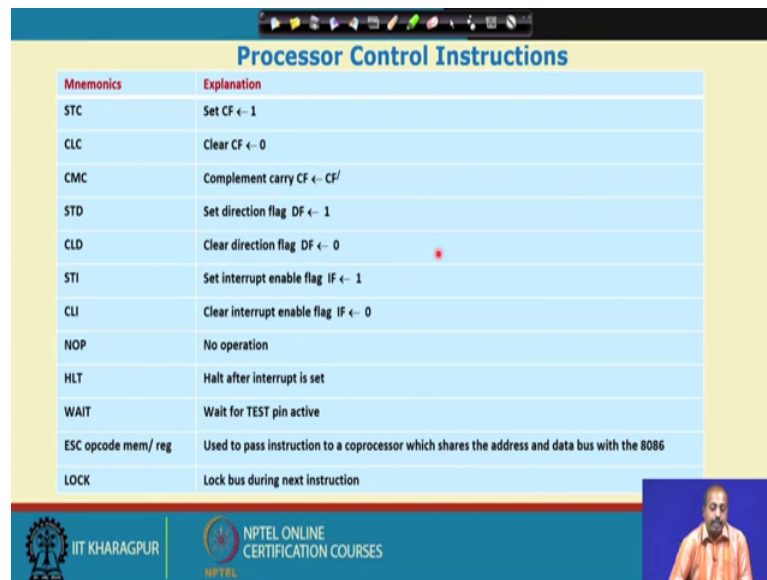
Store byte from AL or word from AX in to string

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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Then just opposite of LODS, we have got STOS store string. So, this store string again it is byte or word. So, this is. So, memory address is calculated as ES into 16 plus DS. So, store STOS. So, these uses the destination index register or DI register because now we are going to store. So, that is the destination. So, it is taken as ES colon DI, so that is the ES colon DI will be the destination address and that will be used. So, this STOSB, it will be copying the content of the registered AL on to this memory address MA E. And if again the same thing if DF is 0, then DI is incremented; if DF is 1, DI is decremented and we have got STOSW where this ah word will be stored the ax will be stored in two successive bytes at MA E and MA E plus 1. So, that way we can have this destinations this store operation carried out.

(Refer Slide Time: 02:44)



Mnemonics	Explanation
STC	Set CF ← 1
CLC	Clear CF ← 0
CMC	Complement carry CF ← CF'
STD	Set direction flag DF ← 1
CLD	Clear direction flag DF ← 0
STI	Set interrupt enable flag IF ← 1
CLI	Clear interrupt enable flag IF ← 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, there are next category of instructions that we have. So, they are called processor control instructions like STC set carry. So, it will set the carry flag CF to 1. Then CLC or clear carry will set will clear the carry flag. Then CMC or say it will be CMC will be compliment the carry. So, whatever be the complimented value of carry CF complimented, so that will come to CF. Then set direction flag, so DF is a bit in the into the PSW register. So, for string operations, so you need to set it and reset it. So, this STD it will set that direction flag to 1 and clear D, it will reset the direction flag to 0. Similarly, for the interrupts we can have STI for enabling the interrupts; and we have got CLI for clearing the interrupts.

Then there is NOP which is no operation, then halt because the so it is halt after interrupt is set. So, after some time the interrupt will come and it will raise the processor back to the normal operation, but this halt instruction is there. Then there is a wait instruction, so this is wait for test pin to be active. So, this is useful when you have got a number of masters in the system. So, you have so the processor executes an wait instruction and in the wait instruction it wait still some outside device or outside master sends a signal to the test pin. And this test pin when it is when the signal comes then the processor will know that this now it is my time to continue. So, they it will continue with its previous instruction whatever it was doing.

So, this wait instruction is used for waiting for the test pin to be active. Then we have got these escape opcode memory registers type of instructions. So, instructions they are prefixed by this escape. So, this is for passing the instruction to a co processor. So, 8086 has got the feature that it can support co processor for some operations and those if such that co processors instructions their opcodes if you have, so 8086 will not be able to understand that. So, if it finds that there is the escape bit set ok, then the opcode has got an escape part then it will understand that it is further the co processors, it will just pass the instruction to the co processor. And there is a lock instruction it will lock the bus during next instruction.

(Refer Slide Time: 05:20)

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 | 76

So, this control transfer instruction, so they are used for transferring control between from to a specific dest[ination] destination or target instruction. So, basically the jump type of instruction from one location, you want to jump to another location or call type of instruction you want to call a sub program so that way so it is this destination can be a jump or a call instructions target or it. So, in this control transfer instructions they do not affect the flags so because they are not doing any arithmetic operation so they do not affect the flag.

So, the control transfer instructions in 8086, so there is call instruction call can be registered memory or displacement 16. So, it will be calling a sub routine. So, this call register so call register so register will have the offset from the from the code segment

register at which this sub routine is loaded and it will be going to that particular address. So, similarly the call memory so this call memory so memory will have the offset and then with respect to the CS that memory will be added and it will be jumping to that particular address.

We have displacement 16 the immediate value immediate displacement can be specified, and that can be executed in the that can taken as the destination address for call. Similarly, we have got jump instruction, register memory displacement 8 and displacement 16. So, you see that there is this displacement 8. So, this is for this relative jump and rest of the thing they are ah they are they are the normal jump instructions. And then there is a return instruction to return from the sub routine.

(Refer Slide Time: 07:04)

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

So, this conditional branch instructions of 8085, so they are they can this branching branch instruction there are some conditional branch instruction. So, they will check that the conditional branch instruction can be signed or they can be unsigned. Now, this they will check flags and if the condition is true then the program control will be transferred to the new memory location in the same segment by modifying the content of the IP. So, the IP register content is updated so that it jumps to the next address.

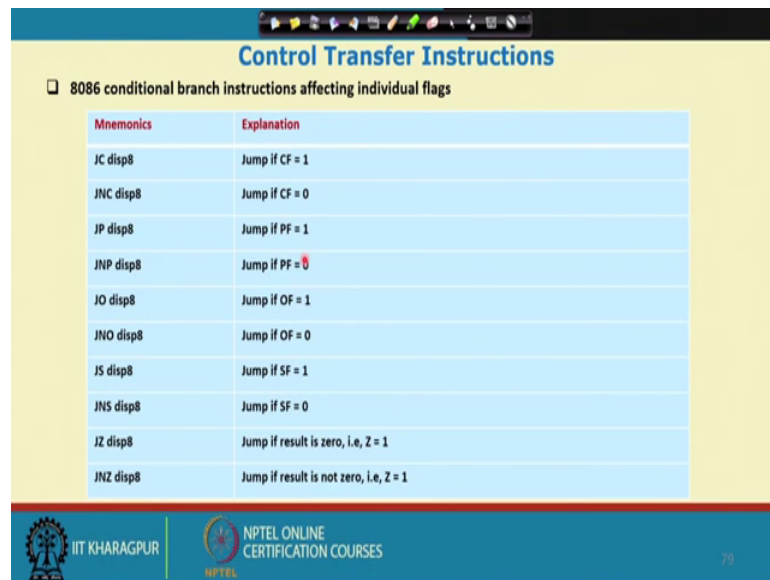
(Refer Slide Time: 07:35)

Control Transfer Instructions			
<input type="checkbox"/> 8086 signed conditional branch instructions		<input type="checkbox"/> 8086 unsigned conditional branch instructions	
Name	Alternate name	Name	Alternate name
JE disp8 Jump if equal	JZ disp8 Jump if result is 0	JE disp8 Jump if equal	JZ disp8 Jump if result is 0
JNE disp8 Jump if not equal	JNZ disp8 Jump if not zero	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	JA disp8 Jump if above	JNBE disp8 Jump if not below or equal
JGE disp8 Jump if greater than or equal	JNL disp8 Jump if not less	JAЕ disp8 Jump if above or equal	JNВ disp8 Jump if not below
JL disp8 Jump if less than	JNGE disp8 Jump if not greater than or equal	JB disp8 Jump if below	JNAE disp8 Jump if not above or equal
JLE disp8 Jump if less than or equal	JNG disp8 Jump if not greater	JBE disp8 Jump if below or equal	JNA disp8 Jump if not above

So, if you try to differentiate between two, so it is like this that in this instructions which is conditional branch is instructions. So, we have got this JE, jump on e, jump if equal, then jump are not equal, jump on greater jump on greater or equal jump on less than jump on less or equal or so. Similarly, we can have this JE is we can have JZ also JE and JZ they are same, because this is taking for equality. So, it will check the zero flag. So, JZ will also check the zero flag. Similarly, JNE and JNZ, they are similar, so that way there are redundancy in the instructions set, but they can be useful. So, the one programmer may like one class of mnemonics that can be done.

On the other hand, for this conditional unsigned conditional branch instructions so we have got some of the instructions are different like say this jump on less. So, we have got jump if below JB. So, sim similarly jump if are not above or equal, they are same as this JB. So, there are. So, they are cannot be unsigned things. So, they do not check the signed bit whereas, they actually check the sign bit. So, both of them otherwise they are same.

(Refer Slide Time: 08:53)



Control Transfer Instructions

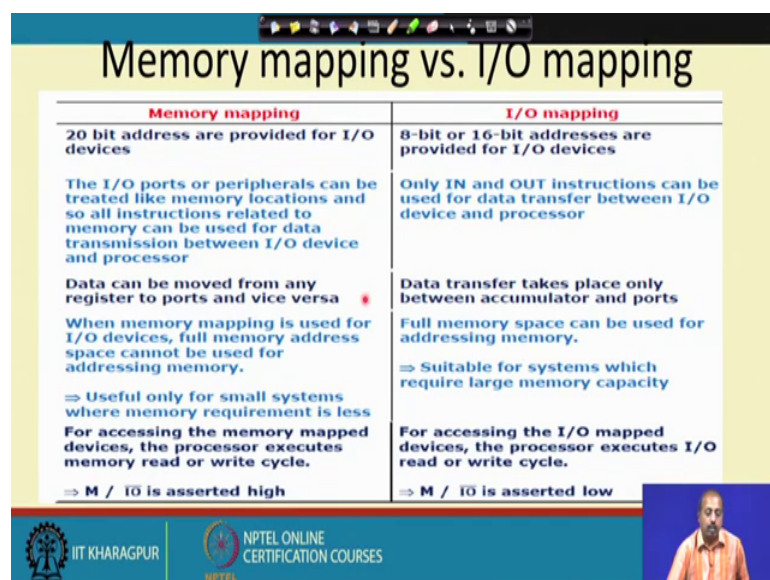
8086 conditional branch instructions affecting individual flags

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 = 0

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, so these are the instructions that will conditional branch instructions that affect the flags and this is the flags that are affected. So, jump on carries so this if the carry flag is said then it will go. So, So, then it for JNC will check the carry flag to be nonzero, so that way this individual status bits will be checked and the corresponding jump will take place.

(Refer Slide Time: 09:16)



Memory mapping vs. I/O mapping

Memory mapping	I/O mapping
20 bit address are provided for I/O devices	8-bit or 16-bit addresses are provided for I/O devices
The I/O ports or peripherals can be treated like memory locations and so all instructions related to memory can be used for data transmission between I/O device and processor	Only IN and OUT instructions can be used for data transfer between I/O device and processor
Data can be moved from any register to ports and vice versa	Data transfer takes place only between accumulator and ports
When memory mapping is used for I/O devices, full memory address space cannot be used for addressing memory.	Full memory space can be used for addressing memory.
⇒ Useful only for small systems where memory requirement is less	⇒ Suitable for systems which require large memory capacity
For accessing the memory mapped devices, the processor executes memory read or write cycle.	For accessing the I/O mapped devices, the processor executes I/O read or write cycle.
⇒ M / $\bar{I/O}$ is asserted high	⇒ M / $\bar{I/O}$ is asserted low

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if you just try to compare between this I O port mapping I O port access, then they can be there are two types of accesses, one is that memory mapped I O, another is I O

mapped I O. So, in case of memory mapped I O, the I O port locations, so they are accessed as memory locations only. So, the memory address that we have. So, 20-bit address that 20-bit address is used for pro access the I O device also. The memory the address decoder it will have some selection line, so that will be fed to this I O device, so that those devices will get selected when a particular when the desired address is desired 20-bit address is put on to the address bus.

The I O port or peripherals can be treated like memory locations, and all instructions related to memory can be used for data transfer between I O device and processor. So, this is the memory map I O operation. So, I O ports are treated memory locations. Data can be moved from any register to ports and vice versa. Just like you can move between the memory location and the register, so here also you can do you can use all those instructions.

So, as a result your the movement of data becomes easier and when memory mapping is used for I O devices. So, full memory address space cannot be used for addressing memory, so that is obvious because some of the locations we are reserving for I O device access. So, we cannot use all the memory locations accessible to the program. So, it is useful only for small systems where memory requirement is less. So, if you are not using full one megabyte of memory space, then in your system then possibly you can also put that I O device pious part of this memory, and so that you do not have to have separate decoder for the I O access.

So, the processor will execute memory read and write operations for accessing the I O device and this M by I O bar pin, so that is asserted high because all accesses are memory access. So, even if you are doing an I O access, so this is basically memory operation. So, this is this M by I O bar pin, so it set high. On the other hand, we have got this I O mapped I O type of operation, where we have seen that I O port addressing 8-bit or 16 bit. So, 8-bit is direct operation, direct addressing; and 16 bit is register indirect addressing by a some register.

And this the only instruction that we can use in I O mapped I O where feature is in and out. So, unlike that MOV type of instructions, unrestricted MOV type of instructions between register and memory locations, so here it is restricted two instructions like in and out for doing data transfer between I O device and the processor. So, data transfer

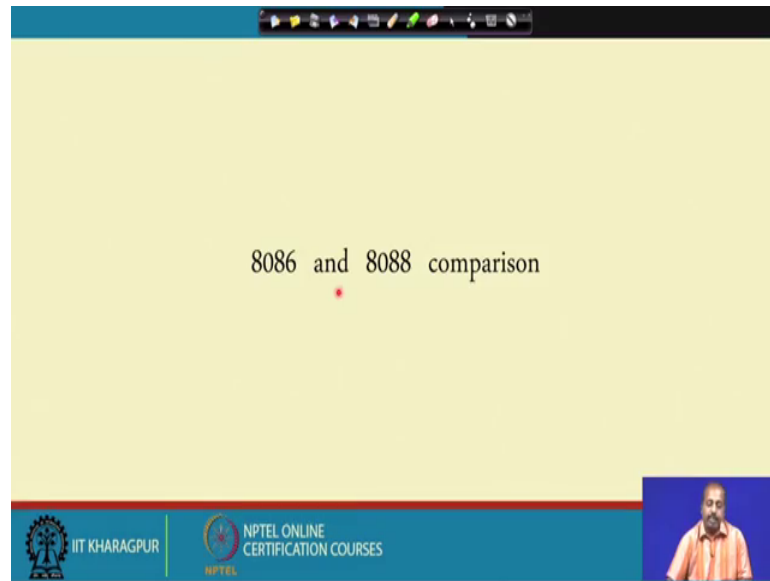
takes place only between accumulator and ports, so it cannot be used to transfer between any set of registers. Whereas, in case of memory mapped I O, you can have data transfer between register and any register and the I O port.

Full memory space can be used for address addressing memory, because now memory space is not restricted by this I O devices. So, you do not have to reserve memory locations for IO. So, it is ah. So, this full memory space can go for I O is suitable for system that require, large memory capacity. So, if you have got large memory capacity then that may be full one megabyte is used for the for addressing the for putting memory into the system.

So, in that case for I O devices, we have we have to separate I O mapped I O. And for accessing this I O mapped devices, the processor executes I O read or write cycle. So, we have got this as the execution is like instead of memory read write cycle, it is I O read write cycle. And those status bits like S 0, S 1, S 2, S 3 those status flags that we have, so those status flag will indicate that it is doing a different type of access and this I O M bar line, so i M by I O bar line, so that is asserted low so that is so it is low means it will be it means that it is accessing some memory location. So, that way this m by I O bar pin, so that is made low and that is ah.

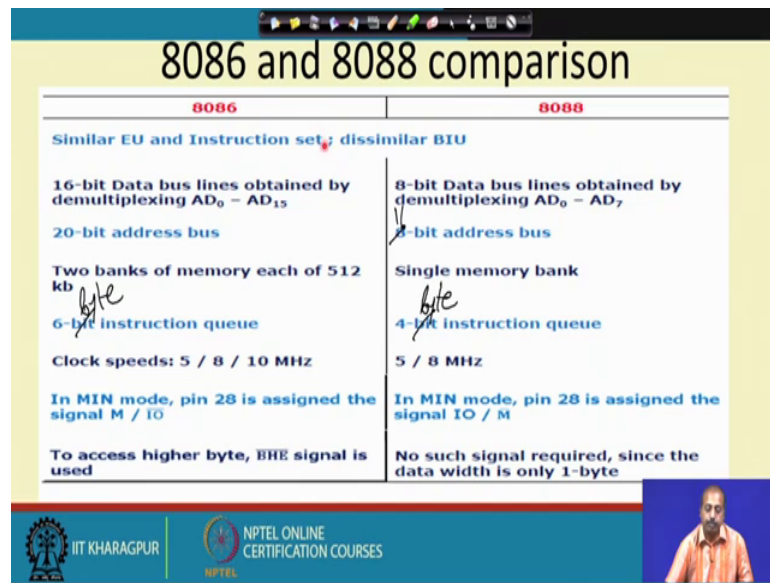
So, this is the basic difference between memory mapped and I O mapped IO. So, you see the major point to note is the memory capacity is a is an issue. Second important thing is a instructions that you can use the in and out here and all you memory movements here as a second thing second thing. Thirdly the extra hardware that is needed for decoding this I O addresses that is required for I O mapped I O and not required for the memory mapped I O. So, otherwise they are same. So, depending upon the application that we have, so we have to choose between these two mappings.

(Refer Slide Time: 14:21)



So, another interesting thing that has happened with 8086 is that 8086 when it came to the market, so previously there was 8085. So, 8085 interface it has got 16 bit address data bus ok. So, this data bus was also multiplex. So, to all together it was a 16 bit interface. So, all the on a on a board so on a on a PCB, so the processor was occupying some pin locations and the bus was only 16 bit. But now if you just take out that 8085 chip and put the 8086 chip, it becomes incomparable hardware device because now you have got say 20-bit bus so that makes it difficult. So, what this Intel people did is that they came up with another processor 8088 internally which is similar to 8086, but externally the interfaces are modified. So, we will look into this 8086 and 8088 comparison, the execution unit is similar instruction set are similar, but the bus interface unit part that is there is different, there are differences.

(Refer Slide Time: 15:30)



The slide is titled "8086 and 8088 comparison" and is presented in a table format. The table has two columns: "8086" and "8088". The text is as follows:

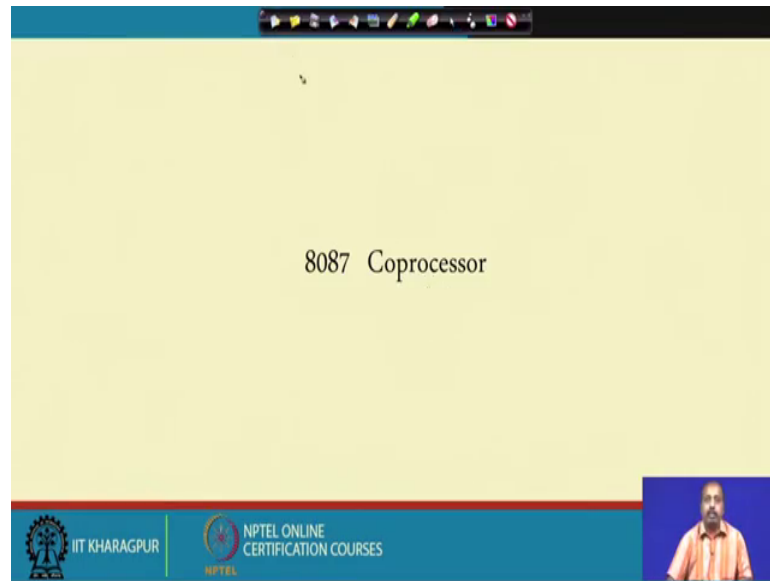
8086	8088
Similar EU and Instruction set; dissimilar BIU	
16-bit Data bus lines obtained by demultiplexing AD ₀ - AD ₁₅	8-bit Data bus lines obtained by demultiplexing AD ₀ - AD ₇
20-bit address bus	16-bit address bus
Two banks of memory each of 512 kb	Single memory bank
6-bit instruction queue	4-bit instruction queue
Clock speeds: 5 / 8 / 10 MHz	5 / 8 MHz
In MIN mode, pin 28 is assigned the signal M / IO	In MIN mode, pin 28 is assigned the signal IO / M
To access higher byte, BHE signal is used	No such signal required, since the data width is only 1-byte

Handwritten annotations in blue ink are present: "byte" is written next to "kb" in the 8086 memory bank row, and "byte" is written next to "4-bit" in the 8088 instruction queue row. A small video inset of a man in an orange shirt is visible in the bottom right corner of the slide.

So, in case of 8086 we have got 16 bit data bus lines obtained by de multiplexing AD 0 to AD 15. In case of 8088 we have got 8-bit data bus like obtained by de multiplexing AD 0 to AD 7. In 8086, we have got 20-bit address bus, so here this is 16 bit this is not 8 bit. So, this is a 16-bit this is 16-bit address bus. So, we have got this two banks of memory each of 512 kilo byte here it is the single bank of memory then this 6-bit instruction queue 6 depth 6 byte instruction queue, but here it is 4 byte instruction queue. So, these are these are not bits. So, these are bytes. So, this is a 4 byte instruction queue, and here it is 6 byte instruction queue.

And clock speeds that you can have in 8086 are 5, 8 and 10 mega hertz, but here it is 5 or 8 megahertz. The now in minimum mode of operations, so pin 28 is assigned the signal M by I O bar and in case of 8088 pin 28 is assigned as signal I O by M bar because this was a variation in 8085 the signal was called I O by M bar in 8086 it is M by I O bar. So, that that modification is reverted, so that it is I O by M bar in 8088. So, to access higher order byte in case of 8086 BHE bar signal was used, but in 8088 that is not required because the data width is only one byte. So, this BHE bar signal is not there.

(Refer Slide Time: 17:18)



So, next we will look otherwise the internal operation everything is same. Next thing that will look into is the co processor 8087. So, 8086 it can operate in along with other masters. And as we have seen that in instruction set of 8086, there is a provision by which you can tell that this instructions are for the co processors. So, you can say that instructions are escape sequence instructions. And if it if the 8086 processor finds that there is an it is an escape type if instruction, so it will pass that instruction to the co processor. So, 8087 is one of the co processor that we have that is very widely used with 8086. So, this is basically a co processor for doing complex mathematical operation. So, 8086 you see that it can do basic arithmetic in arithmetic and logic instructions, but it cannot have floating point operations and all that. So, those can be done by this co processor.

(Refer Slide Time: 18:24)

Co-processor – Intel 8087

Multiprocessor system

- A microprocessor system comprising of two or more processors
- Distributed processing: Entire task is divided in to subtasks

Advantages

- Better system throughput by having more than one processor
- Each processor have a local bus to access local memory or I/O devices so that a greater degree of parallel processing can be achieved
- System structure is more flexible.
One can easily add or remove modules to change the system configuration without affecting the other modules in the system

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, a multiprocessor system that comprises of two or more processors. So, and entire task is divided in to sub tasks, and they are distributed to different processors. So, advantage better system throughput of instead of having a single processor now we have got more than one processor, so that way we have got the advantage that it is giving us more computational power.

Each processor will have its a local bus to access local memory or I O devices, so that greater degree of parallel processing can be achieved. So, there is a local memory and the processor will have a local memory, and local I O ports mean which it will be accessing the locations memory locations and I O ports ah locally, but and whenever it needs to access some global memory location, so there has to be some synchronization between the processors.

So, system structure becomes more flexible. So, one can easily add or move modules to change the system configuration without effecting other modules in the systems. So, you can have more number of multiprocessors attached to the system in a multi processor system without effecting the others and so that is the essence of this distributed processing.

(Refer Slide Time: 19:44)

Co-processor – Intel 8087

8087 coprocessor

- Specially designed to take care of mathematical calculations involving integer and floating point data
- “Math coprocessor” or “Numeric Data Processor (NDP)”
- Works in parallel with a 8086 in the maximum mode

Features

- 1) Can operate on data of the integer, decimal and real types with lengths ranging from 2 to 10 bytes
- 2) Instruction set involves square root, exponential, tangent etc. in addition to addition, subtraction, multiplication and division.
- 3) High performance numeric data processor \Rightarrow It can multiply two 64-bit real numbers in about 27 μ s and calculate square root in about 36 μ s
- 4) Follows IEEE floating point standard
- 5) It is multi bus compatible

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, in case of 8087, so this is specifically designed to take care of the mathematical calculations involving integer and floating point data, so that way it relieves the processor the 8086 processor for doing complex operations ok. And those can be taken care of by 8087. So, they are also 8087 is also called math co processor or numeric data co processer numeric data processor or NDP, so that way we can have a number of instruction number of mathematical operations done by 8087 while 8086 is doing some other operation in terms of in terms of internal computations. So, other instructions, so you have to put a 8086 in the maximum mode, and then you can connect this 8087 with the 8086 through that request grant lines and all that.

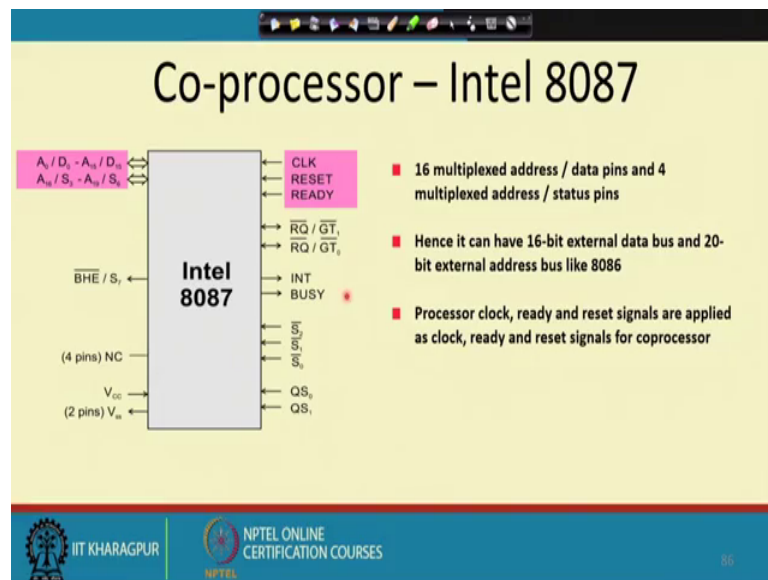
So, if you look into the features of 8087, so it can operate on data of type integer decimal and real types with length ranging from 2 to 10 bytes. So, you see up to 10 bytes means so the huge numbers can be handled by this tens bytes means so it can handle 80 bit numbers. So, 80 bit number is really a huge number. So, instruction set the of 8087, it has got the square root exponential tangent function. So, in addition to normal addition, subtraction, multiplication, division, so you can also compute square root of a number exponential. This numbers can be 10 bytes long also as we have seen in the previous points so numbers can be 10 byte long also.

And you can easily compute the all this things thats square root exponential tangent of some value. So, that way and the other operations are definitely there like addition

subtraction multiplication division, so they are very much there. High performance numeric data processor, so it can multiply 264 bit real numbers in above 27 micro seconds, and calculate square root in above 36 micro seconds, so that is a big thing because 264 bit real numbers, if it can multiply in 27 micro second. In case of 8086, multiplication operation the division operation particularly so it takes about 80 cycles for normal 16 bit numbers.

So, here it is in 27 micro seconds, so it will finish it off this for this 64 bit real numbers, so that is a huge improvement. And a square can also be obtained in 36 micro seconds. So, this way many of the mathematical calculation, we can put it on to the 8087 for relieving the 8086 processor for doing those. And if you want to do then using 8086 means you have to write programs for doing that. So, it will be software driven module, so that way it will take more time ok. And it follows I triple E floating point standards, so there is a standard given by I triple E, so it is following that standard, so that any other party or any other processor who follows this I triple E standardized floating point standard can use this 8087 as the co processor this is a multi bus compatible. So, it is compatible with this 8086 bus structure.

(Refer Slide Time: 23:11)

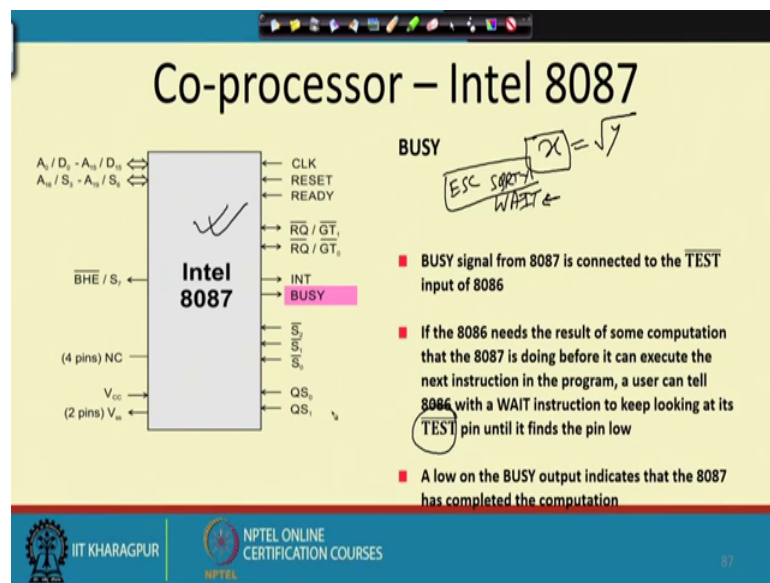


So, so next if you look into this pin diagrams, so it has got this A D 0 to A D 7, so outer so A D, so this is A D 0 to A D 15. And this address 16 to address 19, so this is a exactly same as 8086 this address bar structure multiplexed with the data bus and this status line.

So, 16 bit 16 multiplexed address data pins and 4 multiplexed address data pins so that are there. So, so it can have 16 bit external data bus and 20-bit external address bus just like 8086, so that way it is going to be useful.

Then the processor clock ready and reset signals are applied as clock ready and reset signals for co processors. So, this clock reset and ready, so they are coming for the from the 8086 processor and they are connected to this 87 also. So, if there is a global reset, then this reset pin will be effecting 8087 also, so it will be this will this will reset the 8087 processor also, then this request grant line, so they are useful for getting control of this address and data bus from 8086. And then interrupt line is there by which it can tell the processor 8086 that ok, I am done you can continue now like this over this request and grant lines. So, it can be done.

(Refer Slide Time: 24:38)

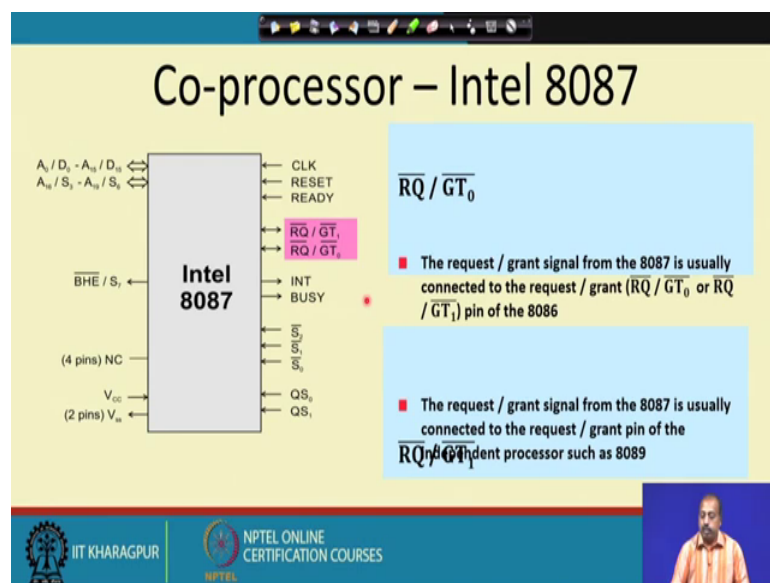


So, there is a busy signal so busy signal from 8087 is connected to the test bar input of 8086. So, why because if the 8086 there is a need some result to be computed by 8087, before it can execute the next instruction then the user can tell 8086 with a wait instruction and keep looking for the test bar pin until it finds the pin low. So, this test this wait instruction, we are seen previously, so it checks for the test bar pin. So, suppose we have got a complex computation and the user finds that it needs the value of x to be computed first, but this value of x can only be computed by this 8087, it may be say

square root of some value say square root of y or something like that. So, it takes it requires 8087.

So, what the processor what the user will do he needs program. So, it will put this wait instruction. And before that it will put this SQRT instruction for this y and put an escape before that; to tell that this is basically an instruction to be executed by the co processor. Now, it puts this wait instruction that means that the program or the 8086 processor will now wait till this test bar line becomes active ok. So, this is so when this busy signal is becoming low that means, 8087 has finished completing doing the square root computation. So, possibly the user program can continue with the next instruction, so that can be that is the purpose. So, a low on the busy output indicates that the 8087 has completed the computation so the 8086 this can continue now.

(Refer Slide Time: 26:27)

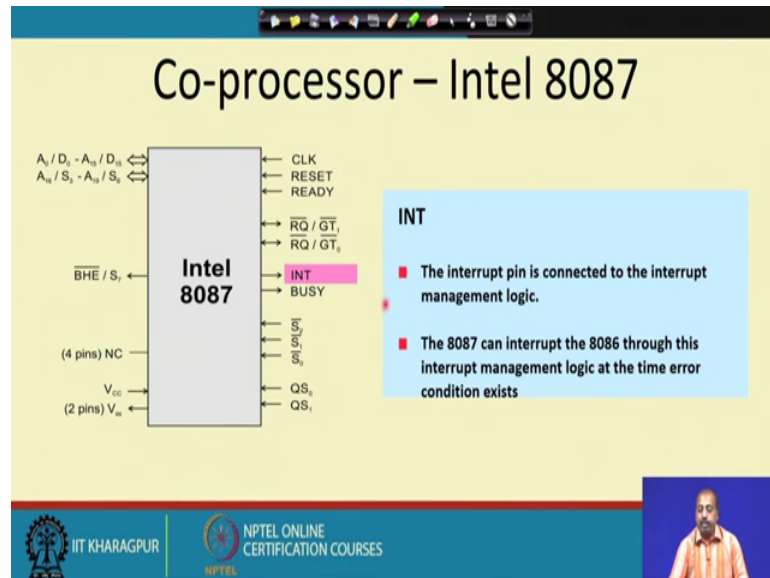


Then for this request and grant lines. So, this RQ, GT 0, and RQ GT 1, so this request grant lines, so this request in grant lines of 8087 are connected to the request grant lines of pins of 8086. So, I so we can have a number of such devices number of such I should say number of masters and so 8086 provides two such request grant lines, so that way you can in a 8087 also we have got two such request grant lines. So, this line is used to request the processor 8086 to release the address and data bus lines.

So, once they are released then this request is granted. So, then this GT line the grant signal will come and then ah the processor will be accessing this address and data bus

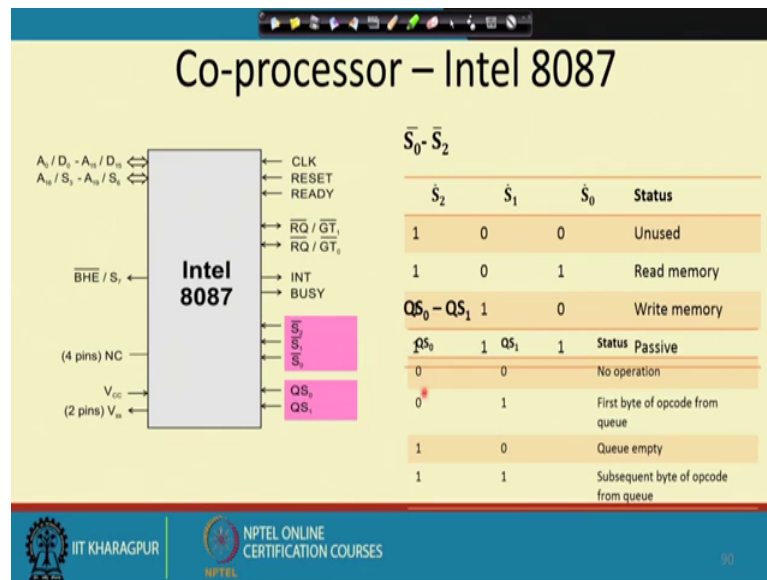
without any problem. So, it will know that the 8086 has released that address and data bus lines. So, it will be able to proceed with that. So, since there can be multiple such devices connected it to 8086 to two such devices can be connected on line zero and line one. So, here also the same thing it is two lines two devices can be connected on these lines.

(Refer Slide Time: 27:54)



Then the interrupt pin, so interrupt pin is connected to the interrupt management logic, so 8087 can interrupt the 8086 through the this interrupt management logic at the time ah at the time error condition exists. So, if there is some error condition, so during the execution some error has occurred may be its divided by zero or something like that. Then it can send an interrupt to the 8086 telling that there was an error in the last computation, so that way this interrupt line can be activated and the 8086 can be informed that there is some error in the last computation.

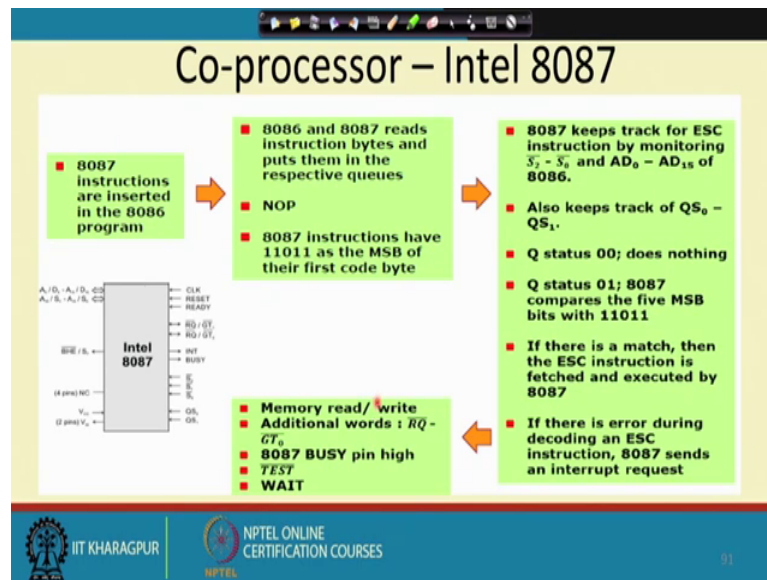
(Refer Slide Time: 28:32)



Then this status lines is S_0 by \overline{S}_0 , \overline{S}_1 , \overline{S}_2 , so this is going like this that if it is 1 0 0. So, this bar this is not used. Then 1 0 1, so this read memory; 1 1 0 write memory and 1 1 1, it is passive. So, this settings are more or less similar to what we have in 8086 also, the similar lines where there. And the interesting thing is only for 1 1 1, where the processor is not doing anything ok, so that is either it is passive mode and 1 0 0, 1 0 1 so they are so 1 0 0 is also that is that is not used so this is that is left by the designers for some other purpose that is not documented. And this 1 0 1 and 1 1 0 they are for read and write memory operations.

Then we have got this QS_0 and QS_1 lines. So, QS_0 , QS_1 lines, so they if it is if it is 1 1 then the subsequent byte of off code will be taken from the queue, if it is 1 0 then the queue is empty 0 1. So, they are same as that 8086 also whatever we have got in 8086. So, similarly we have got this QS_0 and QS_1 bits controlling the showing the status of the instruction queue that we have. So, those values are coming to 8087 processor. So, the 8087 will know that this is the status. So, based on that it will try to fetch the next instruction from the queue or if the if the queue is empty, it will not do anything it will raise the busy signal telling that it is over, it will make the busy signal low telling that the instruction is over.

(Refer Slide Time: 30:19)



So, this overall connection is like this. So, this eight 8087 instructions are inserted in the 8086 program. So, this is the first thing. Now, 8086 and 8087 reads instruction bytes and put them in the respective queues. So, 8086 and 8087 they have got separate instructions they the instructions queue. So, the instruction comes both of them, and then they are put on to the queue. Then one cycle is no operation then 8087 instructions they have 11011 as the MSB of their first code byte. So, this is the thing the opcode part. So, 1011 this is that escape prefix that I was talking about.

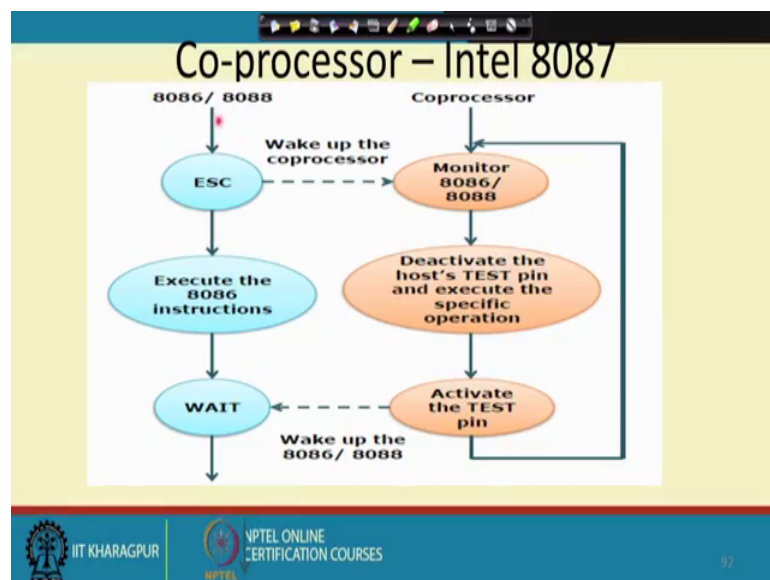
Then, this when this escape prefix is seen. So, 8086 will understand that this is not for me to execute is for 8087. So, 8086 will keep over that instruction and 8087 will take that instruction for execution. So, 8087 keeps track for these ESC instruction by monitoring this S_2 bar to S_0 bar 3. And AD_0 to AD_{15} of 8086, but also keeps track of this QS_0 to QS_1 . So, queue status being 00, it will do nothing queue status is 01 then 8087 compares the five MSB bits 1101, 01 means there is some byte in the memory location in the in the queue. So, it will compare this the five ms most significant bits with 11011 if there is a match that means, it can escape instruction.

So, there the instruction is faced an executed by 8087. If there is an error during execution, then it will send an interrupt to the 8087 will send the interrupt, if there is a error during decoding this ESC the escape instruction, you will send an interrupt otherwise it will do normal memory read write additional parts may be required for that

this RQ GT 0 those lines will be activated by which it will access the memory for getting additional operands and 8087 busy pin is high.

So, so that pin is connected to this busy pin is connected to the test bar pin. So, this test bar pin using the is not active now. So, after sometime when 8087 will be finishing the operation then this busy signal will go low the 8086 processor may be executing an wait instruction for this time, because as I said that if the instruction value that is computing is useful for 8086 the next instruction, then it will be waiting with the wait instruction. And when this test bar pin goes low that means, the instruction was over execution was over and then it was then it will be the 8087 has written the value onto the memory location and 8086 can take that value.

(Refer Slide Time: 33:35)



So, this co processors, so if you just compare that their operation, so 8086 86 or 88, so if it finds an escape sequenced instructions, it will wake up the co processor. So, it will monitor this 8086 or 8088. And then if it is if it is not an escape, then it will 8086 will be executing the instruction then it will go in to the wait state, if there is a wait instruction is there. On the other hand, if it is if it is an escape sequenced instruction, the co processor which was monitoring the 8086 or 88, so it will be triggered that it will it will find that it is there is an escape instruction. It will deactivate the hosts test pin and execute the specific operation. So, here the operation will be executed.

Of course, it will take a help of that request grant lines to take to get accessed to the memory locations where the operands may be there. Then once the instruction is over instruction execution is over it will deactivate that busy line and does activate the test pin for the 8086 processor. So, it will wake up the processor and this then it will continue and this co processor goes back to monitor this 8086, 88 again. So, this way this co processor can be interfaced with 8086 and they work in an integrated fashion, so that many of the operations are done by the co processor and 8086 is relieved of those operations.