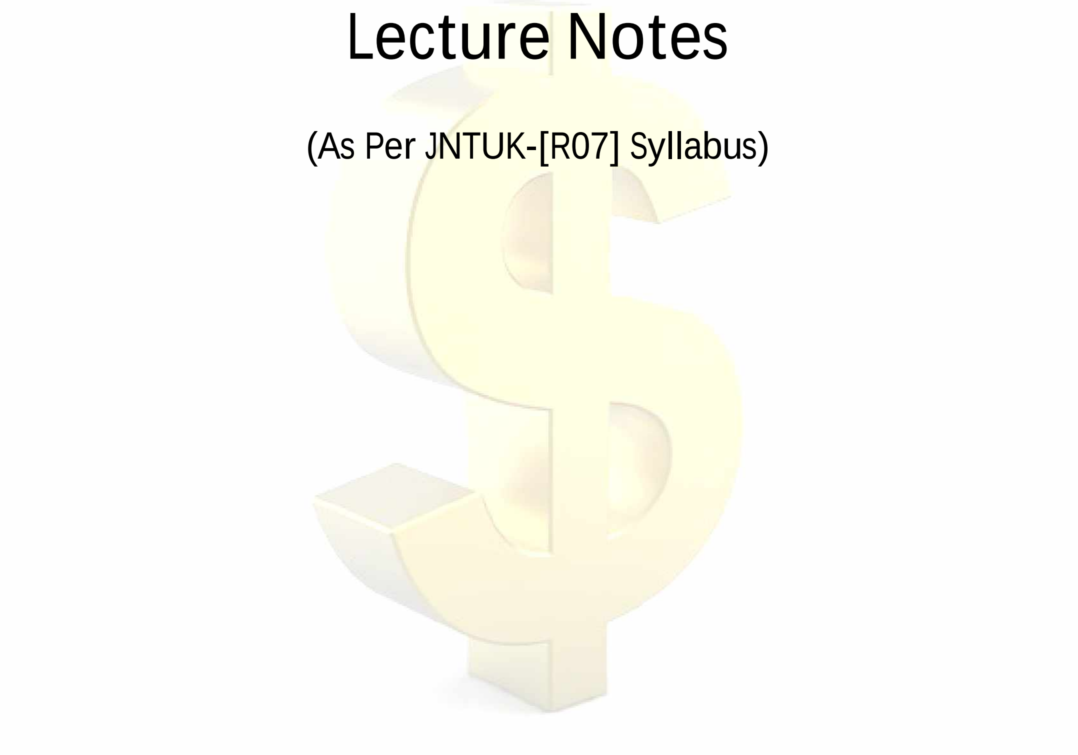
Micro Processors & Micro Controllers
Lecture Notes

(As Per JNTUK-[R07] Syllabus)



Prepared By,
Umasankar.Ch M.Tech(EC)
(€-mail: sankarch.4u@gmail.com)

CONTENTS

CHAPTER – 1

OPERATION OF 8086 MICROPROCESSOR

8086 Microprocessor- Features

•It is a 16-bit μp.

•8086 has a 20 bit address bus can access up to 220 memory locations (1 MB).

•It can support up to 64K I/O ports.

•It provides 14, 16 -bit registers.

•It has multiplexed address and data bus AD0- AD15 and A16 – A19.

•It requires single phase clock with 33% duty cycle to provide internal timing.

•8086 is designed to operate in two modes, Minimum and Maximum.

•It can prefetches upto 6 instruction bytes from memory and queues them in order to speed up instruction execution.

•It requires +5V power supply.

•A 40 pin dual in line package

Internal Architecture of 8086

•8086 has two blocks BIU and EU.

•The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue.

•EU executes instructions from the instruction system byte queue.

•Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance.

•BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder.

•EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.

BUS INTERFACR UNIT:

•It provides a full 16 bit bidirectional data bus and 20 bit address bus.

•The bus interface unit is responsible for performing all external bus operations.

Specifically it has the following functions:

•Instruction fetch, Instruction queuing, Operand fetch and storage, Address relocation and Bus control.

- The BIU uses a mechanism known as an instruction stream queue to implement pipeline ***architecture.*** This queue permits prefetch of up to six bytes of instruction code. When ever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by prefetching the next sequential instruction.
- These prefetching instructions are held in its FIFO queue. With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle.
- After a byte is loaded at the input end of the queue, it autDMAtically shifts up through the FIFO to the empty location nearest the output.
- The EU accesses the queue from the output end. It reads one instruction byte after the other from the output of the queue. If the queue is full and the EU is not requesting access to operand in memory.
- These intervals of no bus activity, which may occur between bus cycles are known as ***Idle state***.
- If the BIU is already in the process of fetching an instruction when the EU request it to read or write operands from memory or I/O, the BIU first completes the instruction fetch bus cycle before initiating the operand read / write cycle.
- The BIU also contains a dedicated adder which is used to generate the 20bit physical address that is output on the address bus. This address is formed by adding an appended 16 bit segment address and a 16 bit offset address.
- For example**:** The physical address of the next instruction to be fetched is formed by combining the current contents of the code segment CS register and the current contents of the instruction pointer IP register.
- The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write.

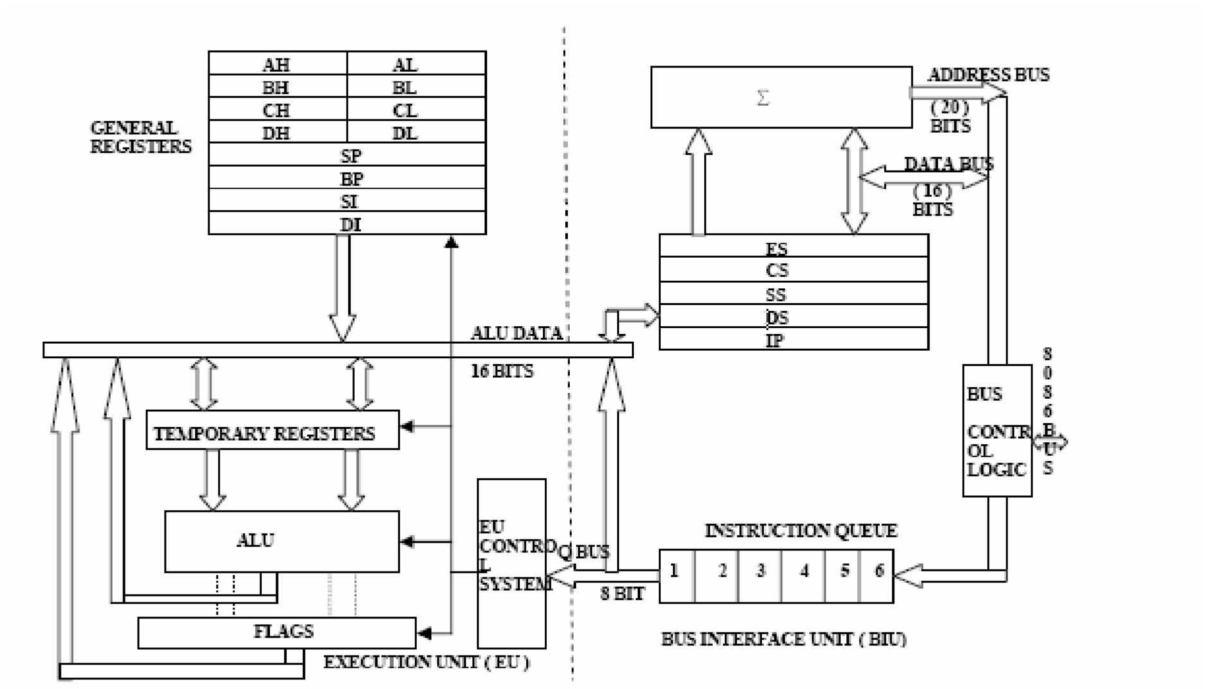


Figure 1 Architecture of 8086

EXECUTION UNIT :

The Execution unit is responsible for decoding and executing all instructions.

•The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bys cycles to memory or I/O and perform the operation specified by the instruction on the operands.

•During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.

•If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue.

•When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions.

•Whenever this happens, the BIU autDMAtically resets the queue and then begins to fetch instructions from this new location to refill the queue.

Module 1 and learning unit 4:

Signal Description of 8086•The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin CERDIP or plastic package.

•The 8086 operates in single processor or multiprocessor configuration to achieve high performance. The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).

•The 8086 signals can be categorized in three groups. The first are the signal having common functions in minimum as well as maximum mode.

•The second are the signals which have special functions for minimum mode and third are the signals having special functions for maximum mode.

Minimum and Maximum Modes:

The minimum mode is selected by applying logic 1 to the MN / MX input pin. This is a single microprocessor configuration.

The maximum mode is selected by applying logic 0 to the MN / MXinput pin. This is a multi micro processors configuration.

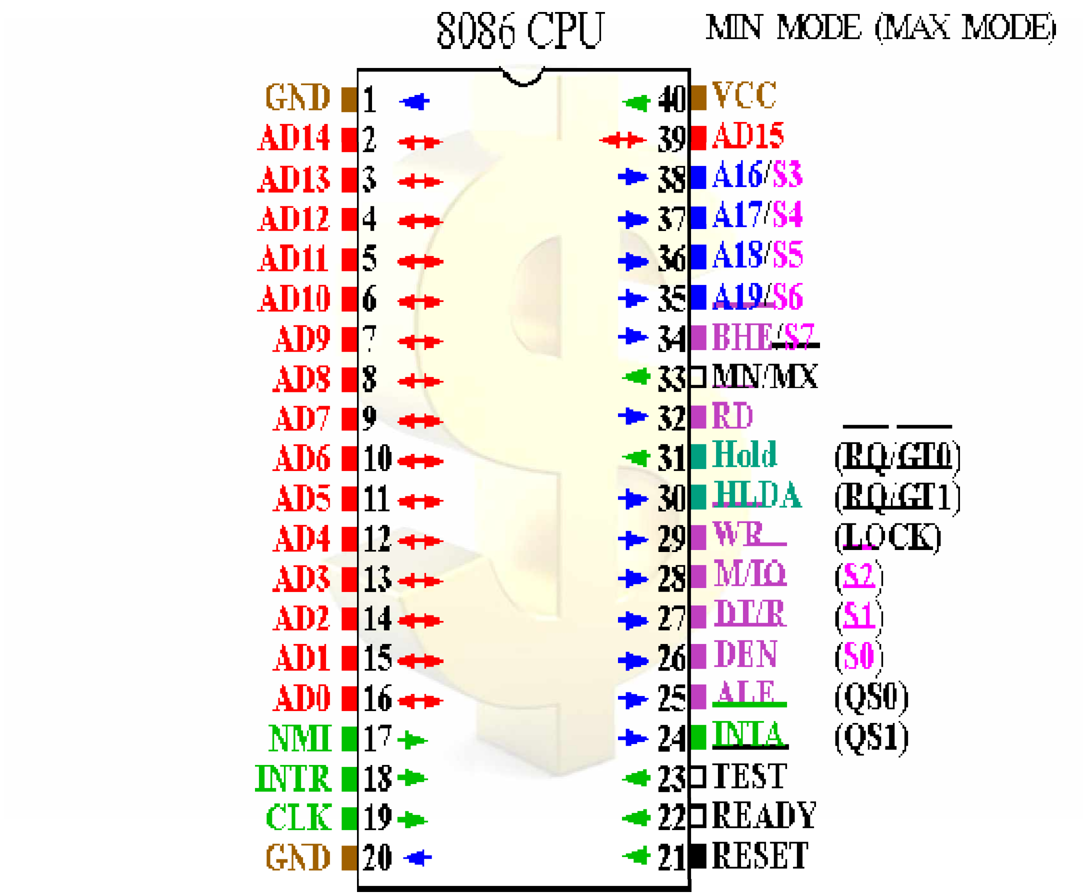


Figure 2 Pin Description of 8086

Signal Description of 8086

The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin CERDIP or plastic package.

•The 8086 operates in single processor or multiprocessor configuration to achieve high performance. The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).

•The 8086 signals can be categorised in three groups. The first are the signal having common functions in minimum as well as maximum mode.

•The second are the signals which have special functions for minimum mode and third are the signals having special functions for maximum mode.

•**The following signal descriptions are common for both modes.**

•**AD15-AD0**: These are the time multiplexed memory I/O address and data lines.

• Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, Tw and T4.

•These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

•**A19/S6,A18/S5,A17/S4,A16/S3:** These are the time multiplexed address and status lines.

•During T1 these are the most significant address lines for memory operations.

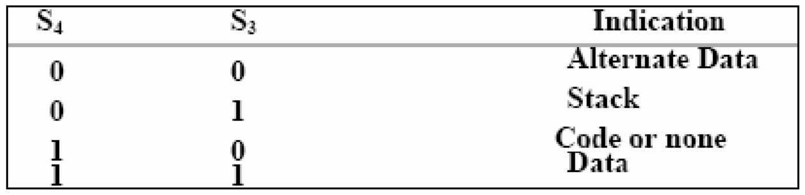
•During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T2,T3,Tw and T4.

•The status of the interrupt enable flag bit is updated at the beginning of each clock cycle.

•The S4 and S3 combinedly indicate which segment register is presently being used for memory accesses as in below fig.

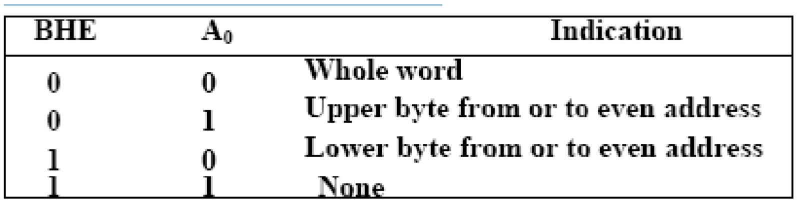
•These lines float to tri-state off during the local bus hold acknowledge. The status line S6 is always low.

•The address bit are separated from the status bit using latches controlled by the ALE signal.



S_4	S_3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or none
1	1	Data

•BHE/S7: The bus high enable is used to indicate the transfer of data over the higher order (D15-D8) data bus as shown in table. It goes low for the data transfer over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals. BHE is low during T1 for read, write and interrupt acknowledge cycles, whenever a byte is to be transferred on higher byte of data bus. The status information is available during T2, T3 and T4. The signal is active low and tristated during hold. It is low during T1 for the first pulse of the interrupt acknowledges cycle.



BHE	A_0	Indication
0	0	Whole word
0	1	Upper byte from or to even address
1	0	Lower byte from or to even address
1	1	None

•RD-Read: This signal on low indicates the peripheral that the processor is performing s memory or I/O read operation. RD is active low and shows the state for T2, T3, Tw of any read cycle. The signal remains tristated during the hold acknowledge.

•**READY**: This is the acknowledgement from the slow device or memory that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086. the signal is active high.

•**INTR-Interrupt Request**: This is a triggered input. This is sampled during the last clock cycles of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle.

•This can be internally masked by resulting the interrupt enable flag. This signal is active high and internally synchronized.

•**TEST**This input is examined by a 'WAIT' instruction. If the TEST pin goes low, execution will continue, else the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.

•**CLK-** Clock Input: The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle.

•**MN/MX**: The logic level at this pin decides whether the processor is to operate in either minimum or maximum mode.

•**The following pin functions are for the minimum mode operation of 8086.**

•**M/IO – Memory/IO**: This is a status line logically equivalent to S2 in maximum mode. When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation. This line becomes active high in the previous T4 and remains active till final T4 of the current cycle. It is tristated during local bus “hold acknowledge “.

•**INTAInterrupt Acknowledge**: This signal is used as a read strobe for interrupt acknowledge cycles. i.e. when it goes low, the processor has accepted the interrupt.

•**ALE – Address Latch Enable**: This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches. This signal is active high and is never tristated.

•**DT/R – Data Transmit/Receive**: This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low.

•**DEN – Data Enable**: This signal indicates the availability of valid data over the address/data lines. It is used to enable the transreceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle of T2 until the middle of T4. This is tristated during ‘ hold acknowledge’ cycle.

•**HOLD, HLDA- Acknowledge**: When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access.

•The processor, after receiving the HOLD request, issues the hold acknowledge signal on HLDA pin, in the middle of the next clock cycle after completing the current bus cycle.•At the same time, the processor floats the local bus and control lines. When the processor detects the HOLD line low, it lowers the HLDA signal. HOLD is an asynchronous input, and is should be externally synchronized.

•If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T4 provided:

1.The request occurs on or before T2 state of the current cycle.

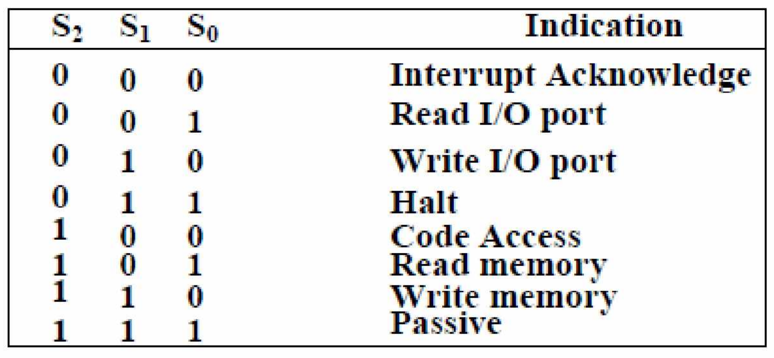
2.The current cycle is not operating over the lower byte of a word.

3.The current cycle is not the first acknowledge of an interrupt acknowledge sequence.

4. A Lock instruction is not being executed.

•***The following pin function are applicable for maximum mode operation of 8086***.

•**S2, S1, S0 – Status Lines**: These are the status lines which reflect the type of operation, being carried out by the processor. These become activity during T4 of the previous cycle and active during T1 and T2 of the current bus cycles.



S_2	S_1	S_0	Indication
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

•**LOCK**This output pin indicates that other system bus master will be prevented from gaining the system bus, while the LOCK signal is low.

•The LOCK signal is activated by the 'LOCK' prefix instruction and remains active until the completion of the next instruction. When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus.

•The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connected to an external bus controller.

•**QS1, QS0 – Queue Status:** These lines give information about the status of the code-prefetch queue. These are active during the CLK cycle after while the queue operation is performed.

•This modification in a simple fetch and execute architecture of a conventional microprocessor offers an added advantage of pipelined processing of the instructions.

•The 8086 architecture has 6-byte instruction prefetch queue. Thus even the largest (6-bytes) instruction can be prefetched from the memory and stored in the prefetch. This results in a faster execution of the instructions.

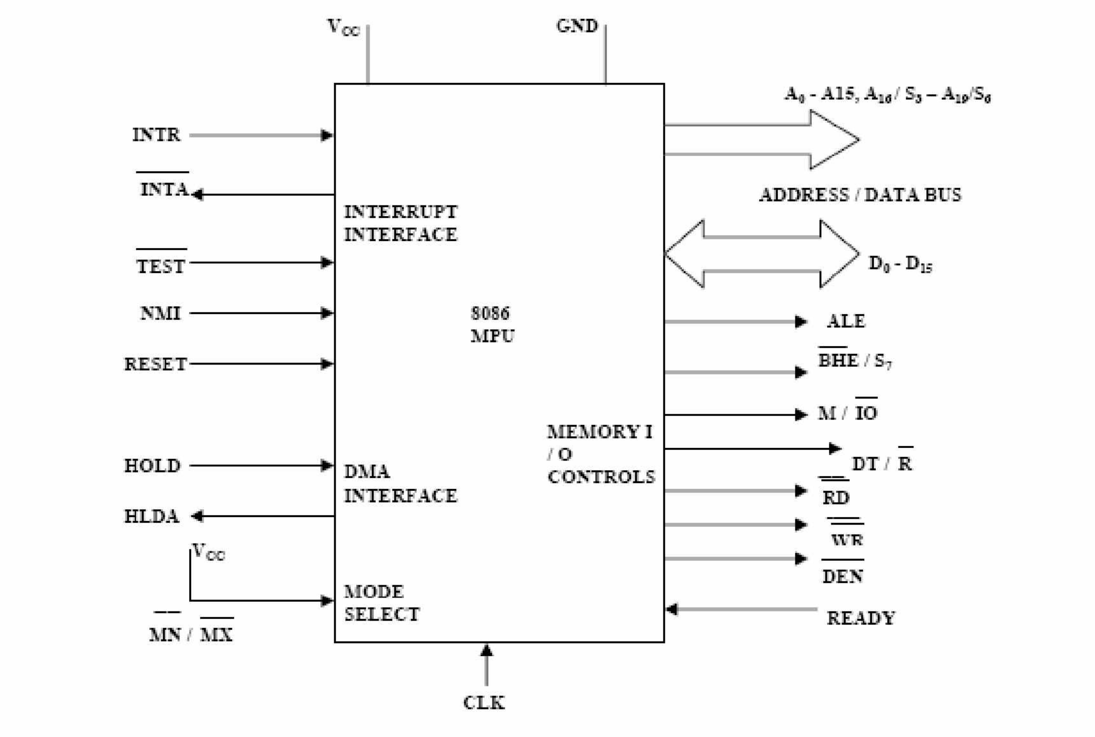
•In 8085 an instruction is fetched, decoded and executed and only after the execution of this instruction, the next one is fetched.

•By prefetching the instruction, there is a considerable speeding up in instruction execution in 8086. This is known as ***instruction pipelining***.

•At the starting the CS:IP is loaded with the required address from which the execution is to be started. Initially, the queue will be empty an the microprocessor starts a fetch operation to bring one byte (the first byte) of instruction code, if the CS:IP address is odd or two bytes at a time, if the CS:IP address is even.

•The first byte is a complete opcode in case of some instruction (one byte opcode instruction) and is a part of opcode, in case of some instructions (two byte opcode instructions), the remaining part of code lie in second byte.

- The second byte is then decoded in continuation with the first byte to decide the instruction length and the number of subsequent bytes to be treated as instruction data.



Signal Groups of 8086

- The queue is updated after every byte is read from the queue but the fetch cycle is initiated by BIU only if at least two bytes of the queue are empty and the EU may be concurrently executing the fetched instructions.
- The next byte after the instruction is completed is again the first opcode byte of the next instruction. A similar procedure is repeated till the complete execution of the program.•The fetch operation of the next instruction is overlapped with the execution of the current instruction. As in the architecture, there are two separate units, namely Execution unit and Bus interface unit.
- While the execution unit is busy in executing an instruction, after it is completely decoded, the bus interface unit may be fetching the bytes of the next instruction from memory, depending upon the queue status.

QS_1	QS_0	Indication
0	0	No operation
0	1	First byte of the opcode from the queue
1	0	Empty queue
1	1	Subsequent byte from the queue

•**RQ/0GT,RQ/1GT – Request/Grant:** These pins are used by the other local bus master in maximum mode, to force the processor to release the local bus at the end of the processor current bus cycle.

•Each of the pin is bidirectional with RQ/GT0 having higher priority than RQ/GT1.

•RQ/GT pins have internal pull-up resistors and may be left unconnected.

•**Request/Grant sequence is as follows:**

1.A pulse of one clock wide from another bus master requests the bus access to 8086.

2.During T4(current) or T1(next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 has allowed the local bus to float and that it will enter the 'hold acknowledge' state at next cycle. The CPU bus interface unit is likely to be disconnected from the local bus of the system.

3.A one clock wide pulse from the another master indicates to the 8086 that the hold request is about to end and the 8086 may regain control of the local bus at the next clock cycle. Thus each master to master exchange of the local bus is a sequence of 3 pulses.

There must be at least one dead clock cycle after each bus exchange.

•The request and grant pulses are active low.

•For the bus request those are received while 8086 is performing memory or I/O cycle, the granting of the bus is governed by the rules as in case of HOLD and HLDA in minimum mode.

General Bus Operation:

•The 8086 has a combined address and data bus commonly referred as a time multiplexed address and data bus.

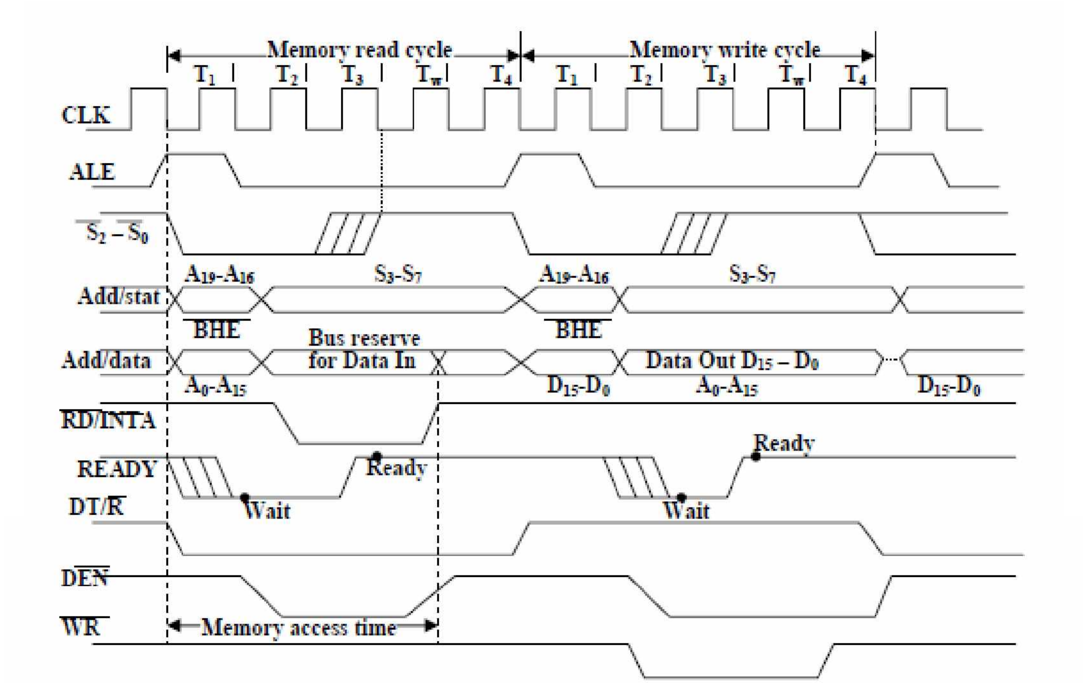
•The main reason behind multiplexing address and data over the same pins is the maximum utilisation of processor pins and it facilitates the use of 40 pin standard DIP package.

•The bus can be demultiplexed using a few latches and transreceivers, when ever required.

•Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T1, T2, T3, T4. The address is transmitted by the processor during T1. It is present on the bus only for one cycle.

•The negative edge of this ALE pulse is used to separate the address and the data or status information. In maximum mode, the status lines S0, S1 and S2 are used to indicate the type of operation.

•Status bits S3 to S7 are multiplexed with higher order address bits and the BHE signal. Address is valid during T1 while status bits S3 to S7 are valid during T2 through T4.



General Bus Operation Cycle in Maximum Mode

Minimum Mode 8086 System

•In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.

•In this mode, all the control signals are given out by the microprocessor chip itself. There is a single microprocessor in the minimum mode system.

•The remaining components in the system are latches, transreceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.

•Latches are generally buffered output D-type flip-flops like 74LS373 or 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.

•Transreceivers are the bidirectional buffers and some times they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signals.

•They are controlled by two signals namely, DEN and DT/R.

•The DEN signal indicates the direction of data, i.e. from or to the processor. The system contains memory for the monitor and users program storage.

•Usually, EPROM are used for monitor storage, while RAM for users program storage. A system may contain I/O devices.

•The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations.

•The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.

•The read cycle begins in T1 with the assertion of address latch enable (ALE) signal and also M / IO signal. During the negative going edge of this signal, the valid address is latched on the local bus.

•The BHE and A0 signals address low, high or both bytes. From T1 to T4 , the M/IO signal indicates a memory or I/O operation.

•At T2, the address is removed from the local bus and is sent to the output. The bus is then tristated. The read (RD) control signal is also activated in T2.

•The read (RD) signal causes the address device to enable its data bus drivers. After RD goes low, the valid data is available on the data bus.

•The addressed device will drive the READY line high. When the processor returns the read signal to high level, the addressed device will again tristate its bus drivers.

•A write cycle also begins with the assertion of ALE and the emission of the address. The M/IO signal is again asserted to indicate a memory or I/O operation. In T2, after sending the address in T1, the processor sends the data to be written to the addressed location.

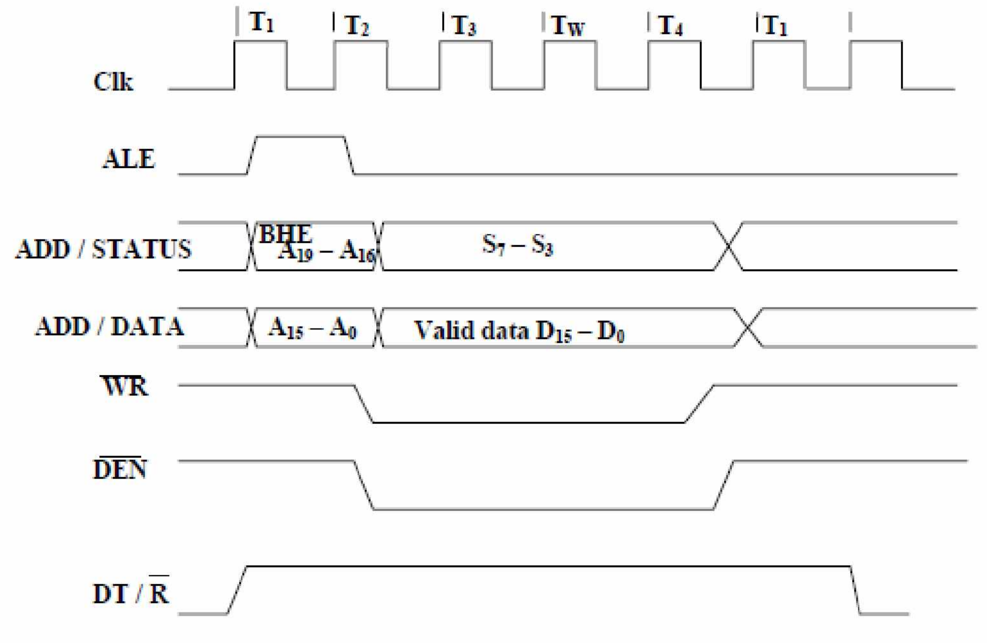
•The data remains on the bus until middle of T4 state. The WR becomes active at the beginning of T2 (unlike RD is somewhat delayed in T2 to provide time for floating).

•The BHE and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or write.

•The M/IO, RD and WR signals indicate the type of data transfer as specified in table below.

•***Hold Response sequence***: The HOLD pin is checked at leading edge of each clock pulse. If it is received active by the processor before T4 of the previous cycle or during T1 state of the current cycle, the CPU activates HLDA in the next clock cycle and for succeeding bus cycles, the bus will be given to another requesting master.

•The control of the bus is not regained by the processor until the requesting master does not drop the HOLD pin low. When the request is dropped by the requesting master, the HLDA is dropped by the processor at the trailing edge of the next clock.



Write Cycle Timing Diagram for Minimum Mode

Maximum Mode 8086 System

•In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.

•In this mode, the processor derives the status signal S2, S1, S0. Another chip called bus controller derives the control signal using this status information.

•In the maximum mode, there may be more than one microprocessor in the system configuration.

•The components in the system are same as in the minimum mode system.

•The basic function of the bus controller chip IC8288, is to derive control signals like RD and WR (for memory and I/O devices), DEN, DT/R, ALE etc. using the information by the processor on the status lines.

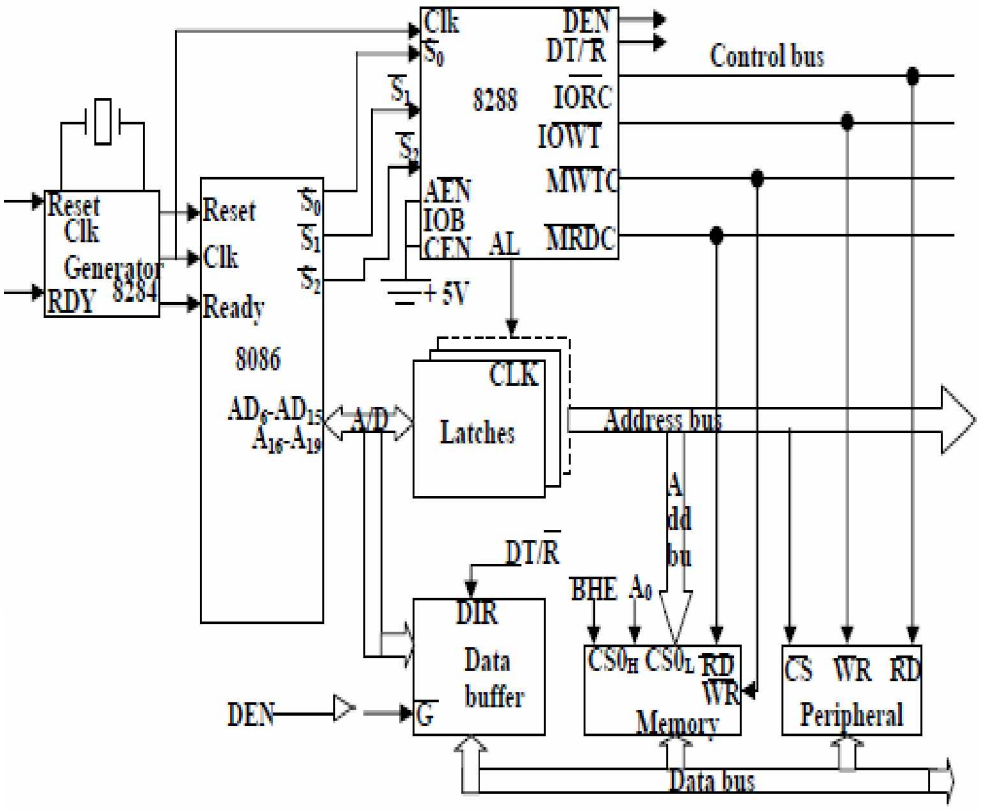
•The bus controller chip has input lines S2, S1, S0 and CLK. These inputs to 8288 are driven by CPU.

•It derives the outputs ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC, IOWC and AIOWC. The AEN, IOB and CEN pins are specially useful for multiprocessor systems.

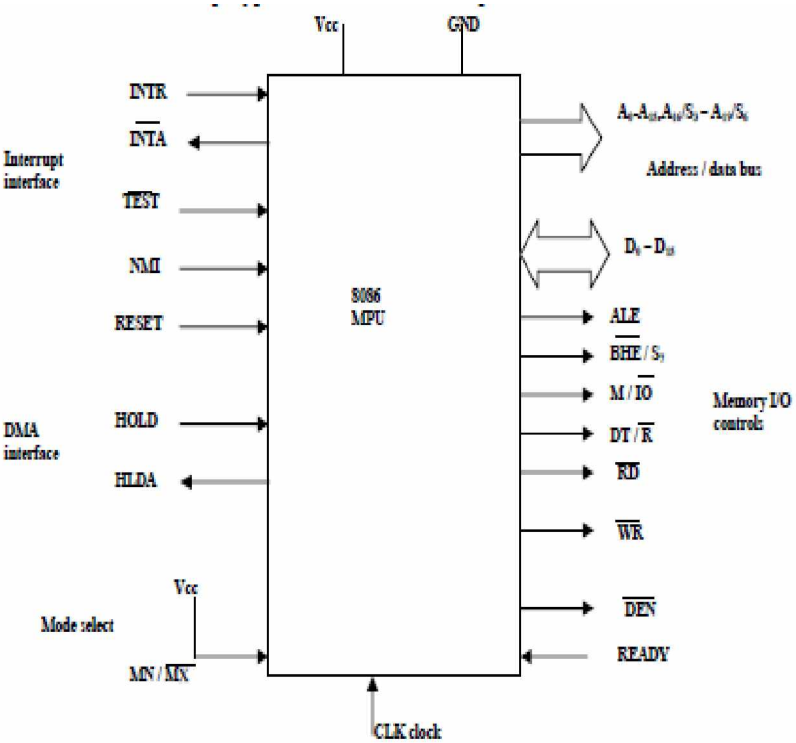
•AEN and IOB are generally grounded. CEN pin is usually tied to +5V. The significance of the MCE/PDEN output depends upon the status of the IOB pin.

•If IOB is grounded, it acts as master cascade enable to control cascade 8259A, else it acts as peripheral data enable used in the multiple bus configurations.

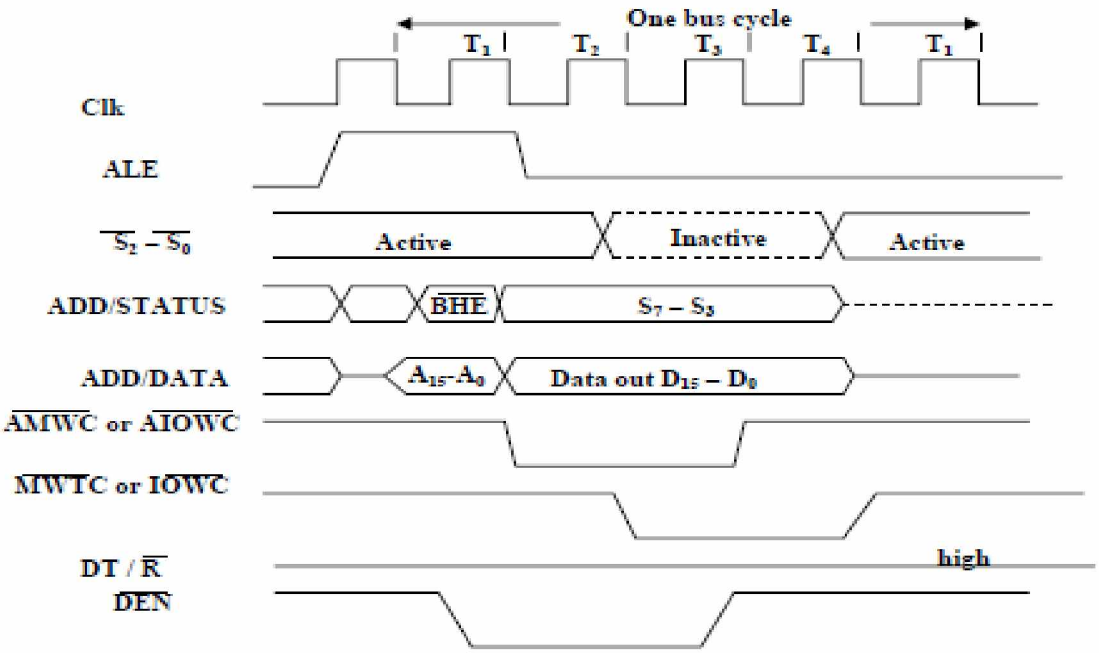
- •INTA pin used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.
- •IORC, IOWC are I/O read command and I/O write command signals respectively. These signals enable an IO interface to read or write the data from or to the address port.
- •The MRDC, MWTC are memory read command and memory write command signals respectively and may be used as memory read or write signals.
- •All these command signals instructs the memory to accept or send data from or to the bus.
- •For both of these write command signals, the advanced signals namely AIOWC and AMWTC are available.
- •Here the only difference between in timing diagram between minimum mode and maximum mode is the status signals used and the available control and advanced command signals.



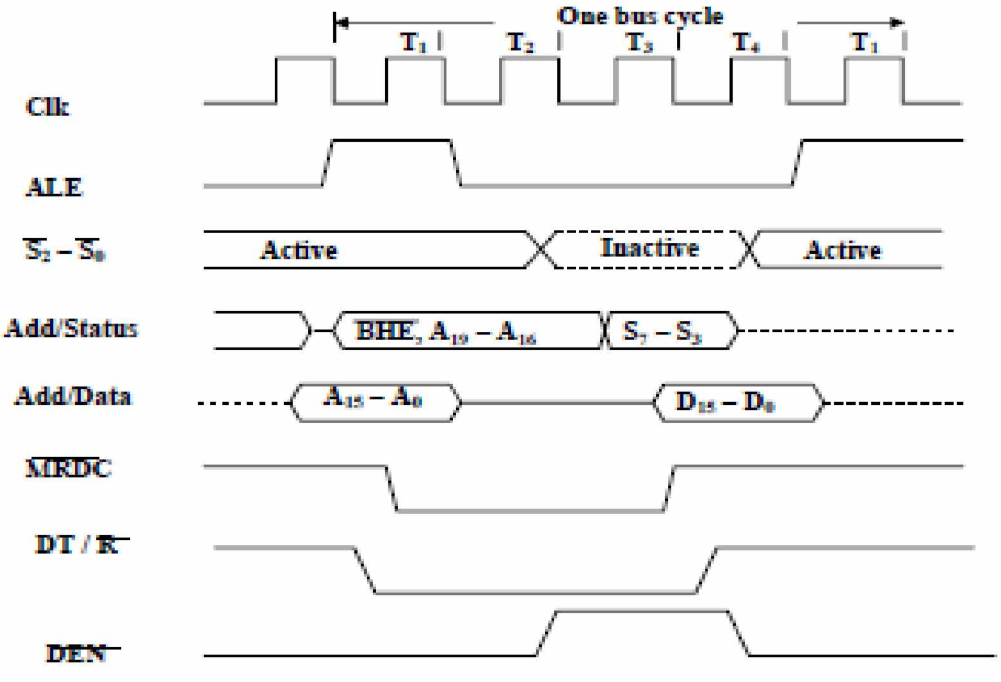
Maximum Mode 8086 System.



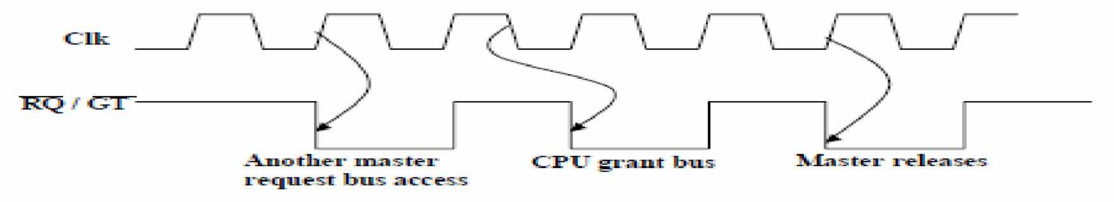
Block Diagram of the Minimum Mode 8086 MPU



Memory Write Timing in Maximum mode.



Memory Read Timing in Maximum Mode



$\overline{RQ}/\overline{GT}$ Timings in Maximum Mode.

- R0, S1, S2 are set at the beginning of bus cycle.8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T1.
- In T2, 8288 will set DEN=1 thus enabling transceivers, and for an input it will activate MRDC or IORC. These signals are activated until T4. For an output, the AMWC or AIOWC is activated from T2 to T4 and MWTC or IOWC is activated from T3 to T4.
- The status bit S0 to S2 remains active until T3 and become passive during T3 and T4**.**
- If reader input is not activated before T3, wait state will be inserted between T3 and T4.
- **Timings for RQ/ GT Signals:**

The request/grant response sequence contains a series of three pulses. The request/grant pins are checked at each rising pulse of clock input.

- When a request is detected and if the condition for HOLD request are satisfied, the processor issues a grant pulse over the RQ/GT pin immediately during T4 (current) or T1 (next) state.
- When the requesting master receives this pulse, it accepts the control of the bus, it sends a release pulse to the processor using RQ/GT pin.

Minimum Mode Interface

- When the Minimum mode operation is selected, the 8086 provides all control signals needed to implement the memory and I/O interface.
- The minimum mode signal can be divided into the following basic groups: address/data bus, status, control, interrupt and DMA.
- **Address/Data Bus**: these lines serve two functions. As an address bus is 20 bits long and consists of signal lines A0 through A19. A19 represents the MSB and A0 LSB. A 20bit address gives the 8086 a 1Mbyte memory address space. More over it has an independent I/O address space which is 64K bytes in length.
- The 16 data bus lines D0 through D15 are actually multiplexed with address lines A0 through A15 respectively. By multiplexed we mean that the bus work as an address bus during first machine cycle and as a data bus during next machine cycles. D15 is the MSB and D0 LSB.

•When acting as a data bus, they carry read/write data for memory, input/output data for I/O devices, and interrupt type codes from an interrupt controller.

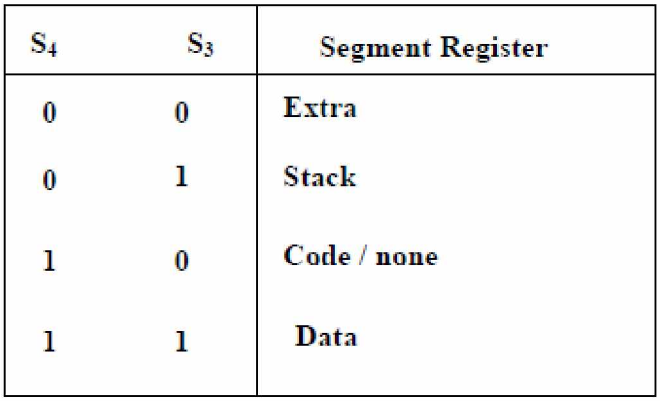
•**Status signal:**

The four most significant address lines A19 through A16 are also multiplexed but in this case with status signals S6 through S3. These status bits are output on the bus at the same time that data are transferred over the other bus lines.

•Bit S4 and S3 together from a 2 bit binary code that identifies which of the 8086 internal segment registers are used to generate the physical address that was output on the address bus during the current bus cycle.

•Code S4S3 = 00 identifies a register known as ***extra segment register*** as the source of the segment address.

•Status line S5 reflects the status of another internal characteristic of the 8086. It is the logic level of the internal enable flag. The last status bit S6 is always at the logic 0 level.



S_4	S_3	Segment Register
0	0	Extra
0	1	Stack
1	0	Code / none
1	1	Data

•**Control Signals**:

The control signals are provided to support the 8086 memory I/O interfaces. They control functions such as when the bus is to carry a valid address in which direction data are to be transferred over the bus, when valid write data are on the bus and when to put read data on the system bus.

•ALE is a pulse to logic 1 that signals external circuitry when a valid address word is on the bus. This address must be latched in external circuitry on the 1-to-0 edge of the pulse at ALE.

•Another control signal that is produced during the bus cycle is BHE bank high enable. Logic 0 on this used as a memory enable signal for the most significant byte half of the data bus D8 through D1. These lines also serves a second function, which is as the S

status line

- Using the M/IO and DT/R lines, the 8086 signals which type of bus cycle is in progress and in which direction data are to be transferred over the bus.
- The logic level of M/IO tells external circuitry whether a memory or I/O transfer is taking place over the bus. Logic 1 at this output signals a memory operation and logic 0 an I/O operation.
- The direction of data transfer over the bus is signaled by the logic level output at DT/R. When this line is logic 1 during the data transfer part of a bus cycle, the bus is in the transmit mode. Therefore, data are either written into memory or output to an I/O device.
- On the other hand, logic 0 at DT/R signals that the bus is in the receive mode. This corresponds to reading data from memory or input of data from an input port.
- The signal read RD and write WR indicates that a read bus cycle or a write bus cycle is in progress. The 8086 switches WR to logic 0 to signal external device that valid write or output data are on the bus.
- On the other hand, RD indicates that the 8086 is performing a read of data of the bus. During read operations, one other control signal is also supplied. This is DEN (data enable) and it signals external devices when they should put data on the bus.
- There is one other control signal that is involved with the memory and I/O interface. This is the READY signal.
- READY signal is used to insert wait states into the bus cycle such that it is extended by a number of clock periods. This signal is provided by an external clock generator device and can be supplied by the memory or I/O sub-system to signal the 8086 when they are ready to permit the data transfer to be completed.
- **Interrupt signals**: The key interrupt interface signals are interrupt request (INTR) and interrupt acknowledge (INTA).
- INTR is an input to the 8086 that can be used by an external device to signal that it need to be serviced.
- Logic 1 at INTR represents an active interrupt request. When an interrupt request has been recognized by the 8086, it indicates this fact to external circuit with pulse to logic 0 at the INTA output.
- The TEST input is also related to the external interrupt interface. Execution of a WAIT instruction causes the 8086 to check the logic level at the TEST input.
- If the logic 1 is found, the MPU suspend operation and goes into the idle state. The 8086 no longer executes instructions, instead it repeatedly checks the logic level of the TEST input waiting for its transition back to logic 0.

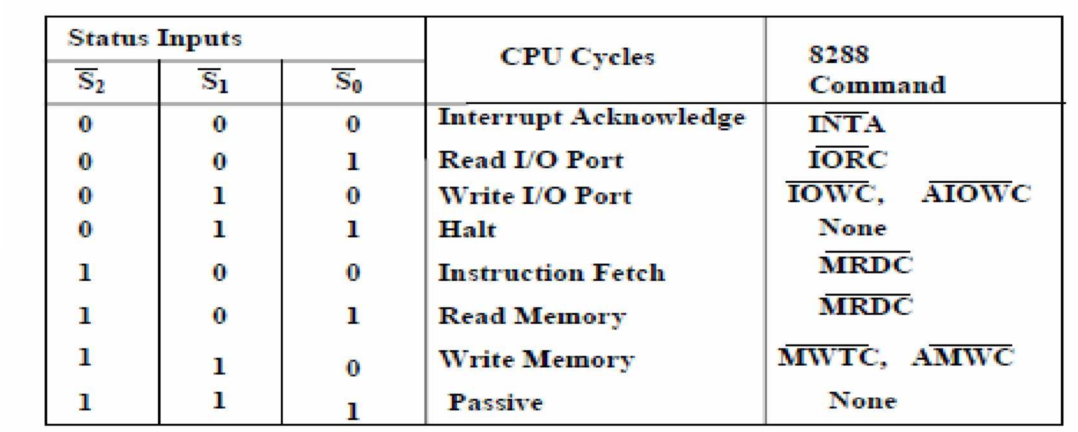
- As TEST switches to 0, execution resume with the next instruction in the program. This feature can be used to synchronize the operation of the 8086 to an event in external hardware.
- There are two more inputs in the interrupt interface: the nonmaskable interrupt NMI and the reset interrupt RESET.
- On the 0-to-1 transition of NMI control is passed to a nonmaskable interrupt service routine. The RESET input is used to provide a hardware reset for the 8086. Switching RESET to logic 0 initializes the internal register of the 8086 and initiates a reset service routine.
- **DMA Interface signals**:The direct memory access DMA interface of the 8086 minimum mode consist of the HOLD and HLDA signals.
- When an external device wants to take control of the system bus, it signals to the 8086 by switching HOLD to the logic 1 level. At the completion of the current bus cycle, the 8086 enters the hold state. In the hold state, signal lines AD0 through AD15, A16/S3 through A19/S6, BHE, M/IO, DT/R, RD, WR, DEN and INTR are all in the high Z state. The 8086 signals external device that it is in this state by switching its HLDA output to logic 1 level.

Maximum Mode Interface

- When the 8086 is set for the maximum-mode configuration, it provides signals for implementing a multiprocessor / coprocessor system environment.
- By multiprocessor environment we mean that one microprocessor exists in the system and that each processor is executing its own program.
- Usually in this type of system environment, there are some system resources that are common to all processors.
- They are called as ***global resources***. There are also other resources that are assigned to specific processors. These are known as ***local or private resources***.
- Coprocessor also means that there is a second processor in the system. In this two processor does not access the bus at the same time.
- One passes the control of the system bus to the other and then may suspend its operation.
- In the maximum-mode 8086 system, facilities are provided for implementing allocation of global resources and passing bus control to other microprocessor or coprocessor.
- **8288 Bus Controller – Bus Command and Control Signals**:

8086 does not directly provide all the signals that are required to control the memory, I/O and interrupt interfaces.

- Specially the WR, M/IO, DT/R, DEN, ALE and INTA, signals are no longer produced by the 8086. Instead it outputs three status signals S0, S1, S2 prior to the initiation of each bus cycle. This 3- bit bus status code identifies which type of bus cycle is to follow.
- S2S1S0 are input to the external bus controller device, the bus controller generates the appropriately timed command and control signals.



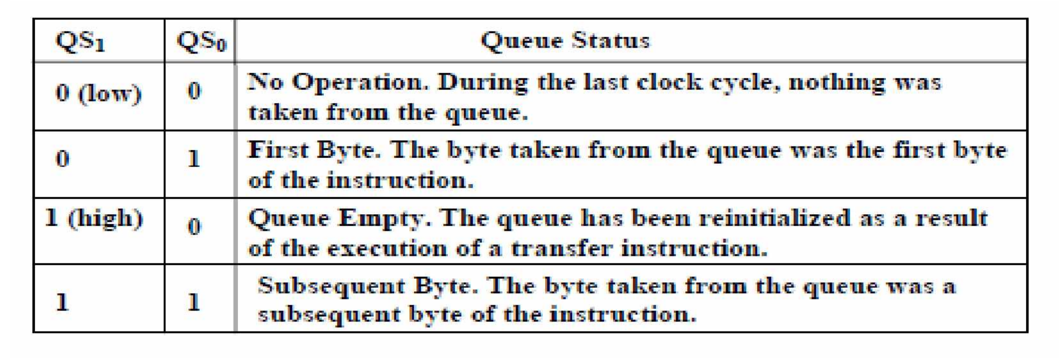
Status Inputs			CPU Cycles	8288 Command
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$		
0	0	0	Interrupt Acknowledge	$\overline{\text{INTA}}$
0	0	1	Read I/O Port	$\overline{\text{IORC}}$
0	1	0	Write I/O Port	$\overline{\text{IOWC}}$, $\overline{\text{AIOWC}}$
0	1	1	Halt	None
1	0	0	Instruction Fetch	$\overline{\text{MRDC}}$
1	0	1	Read Memory	$\overline{\text{MRDC}}$
1	1	0	Write Memory	$\overline{\text{MWTC}}$, $\overline{\text{AMWC}}$
1	1	1	Passive	None

Bus Status Codes

- The 8288 produces one or two of these eight command signals for each bus cycles. For instance, when the 8086 outputs the code S2S1S0 equals 001, it indicates that an ***I/O read cycle*** is to be performed.
- In the code 111 is output by the 8086, it is signaling that no bus activity is to take place.
- The control outputs produced by the 8288 are DEN, DT/R and ALE. These 3 signals provide the same functions as those described for the minimum system mode. This set of bus commands and control signals is compatible with the Multibus and industry standard for interfacing microprocessor systems.
- ***The output of 8289 are bus arbitration signals***:

Bus busy (BUSY), *common bus request* (CBRQ), *bus priority out* (BPRO), *bus priority in* (BPRN), *bus request* (BREQ) and *bus clock* (BCLK).

- They correspond to the bus exchange signals of the Multibus and are used to lock other processor off the system bus during the execution of an instruction by the 8086.
- In this way the processor can be assured of uninterrupted access to common system resources such as ***global memory.***
- **Queue Status Signals**: Two new signals that are produced by the 8086 in the maximum-mode system are queue status outputs QS0 and QS1. Together they form a 2-bit queue status code, QS1QS0.
- Following table shows the four different queue status.



QS_1	QS_0	Queue Status
0 (low)	0	No Operation. During the last clock cycle, nothing was taken from the queue.
0	1	First Byte. The byte taken from the queue was the first byte of the instruction.
1 (high)	0	Queue Empty. The queue has been reinitialized as a result of the execution of a transfer instruction.
1	1	Subsequent Byte. The byte taken from the queue was a subsequent byte of the instruction.

•**Local Bus Control Signal – Request / Grant Signals**: In a maximum mode configuration, the minimum mode HOLD, HLDA interface is also changed. These two are replaced by request/grant lines RQ/ GT0 and RQ/ GT1, respectively. They provide a prioritized bus access mechanism for accessing the local bus.

Internal Registers of 8086

•The 8086 has four groups of the user accessible internal registers. They are the instruction pointer, four data registers, four pointer and index register, four segment registers.

•The 8086 has a total of fourteen 16-bit registers including a 16 bit register called the ***status register***, with 9 of bits implemented for status and control flags.

•Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. To specify where in 1 MB of processor memory these 4 segments are located the processor uses four segment registers:

•**Code segment** (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is autDMAtically updated during far jump, far call and far return instructions.

•**Stack segment** (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.

•**Data segment** (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions.

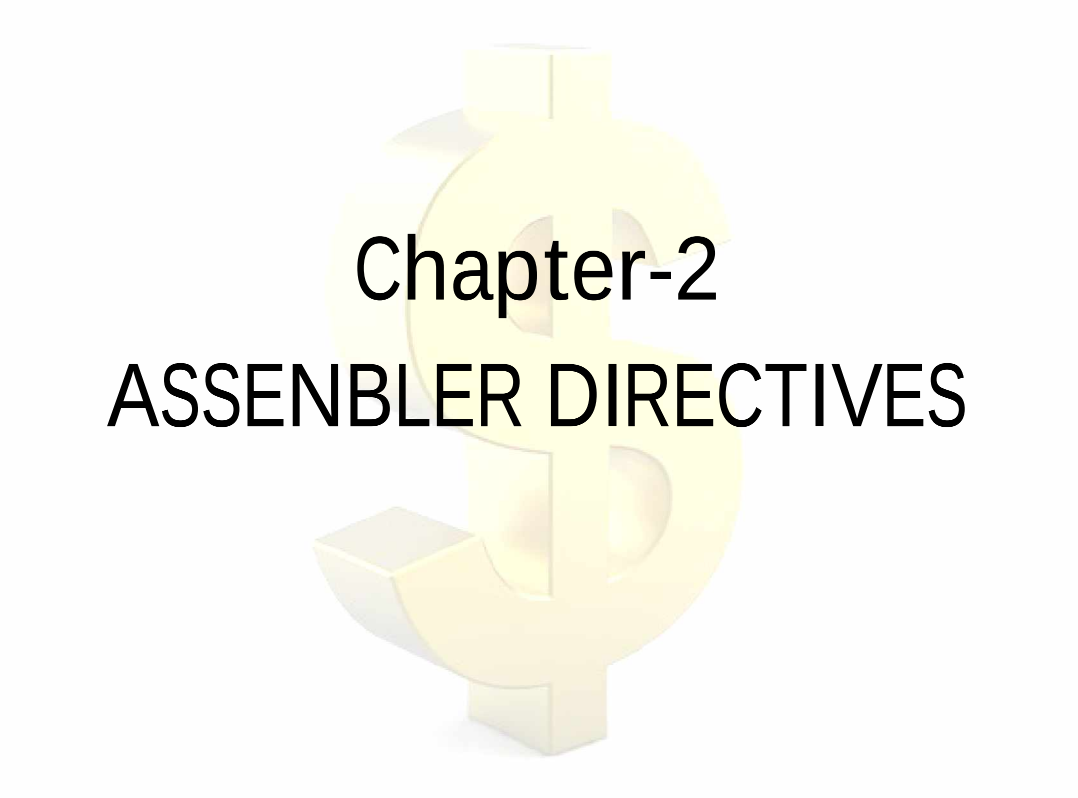
•**Accumulator** register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.

- **Base** register 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. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.
- **Count** register 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. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation,.
- **Data** 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. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.
- **The following registers are both general and index registers:**
- **Stack Pointer** (SP) is a 16-bit register pointing to program stack.
- **Base Pointer** (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.
- **Source Index** (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.
- **Destination Index** (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Other registers:

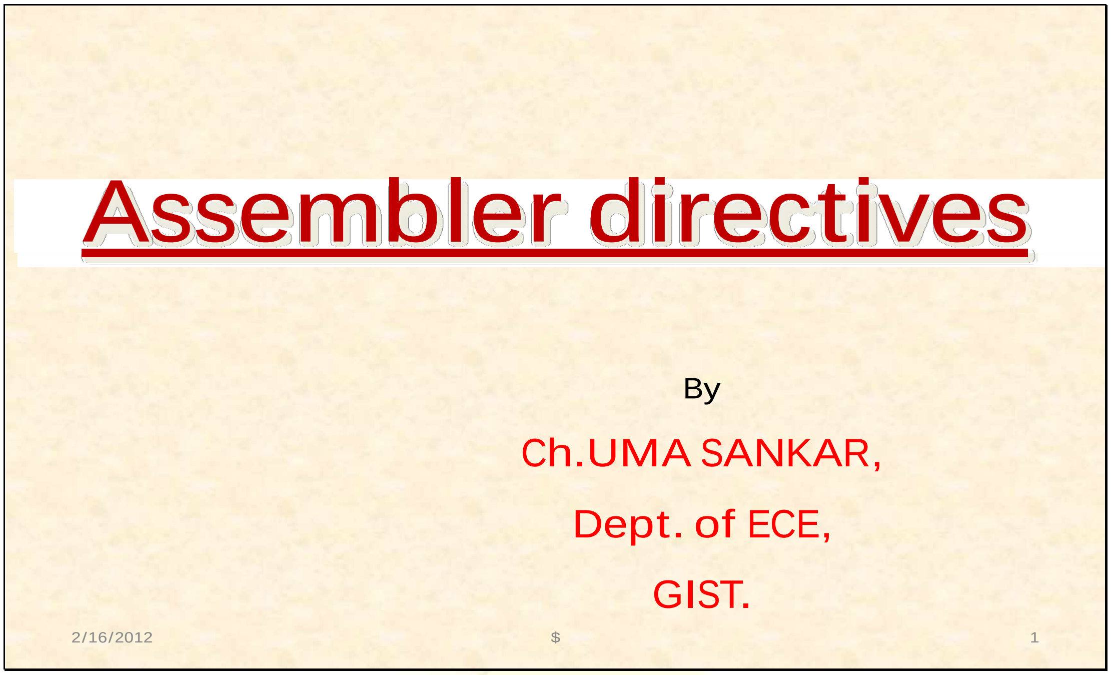
- **Instruction Pointer** (IP) is a 16-bit register.

- **Flags** is a 16-bit register containing 9 one bit flags.
- **Overflow Flag** (OF) - set if the result is too large positive number, or is too small negative number to fit into destination operand.
- **Direction Flag** (DF) - if set then string manipulation instructions will auto-decrement index registers. If cleared then the index registers will be auto-incremented.
- **Interrupt-enable Flag** (IF) - setting this bit enables maskable interrupts.
- **Single-step Flag** (TF) - if set then single-step interrupt will occur after the next instruction.
- **Sign Flag** (SF) - set if the most significant bit of the result is set.
- **Zero Flag** (ZF) - set if the result is zero.
- **Auxiliary carry Flag** (AF) - set if there was a carry from or borrow to bits 0-3 in the AL register.
- **Parity Flag** (PF) - set if parity (the number of "1" bits) in the low-order byte of the result is even.
- **Carry Flag** (CF) - set if there was a carry from or borrow to the most significant bit during last result calculation.



Chapter-2

ASSENBLER DIRECTIVES



Assembler directives

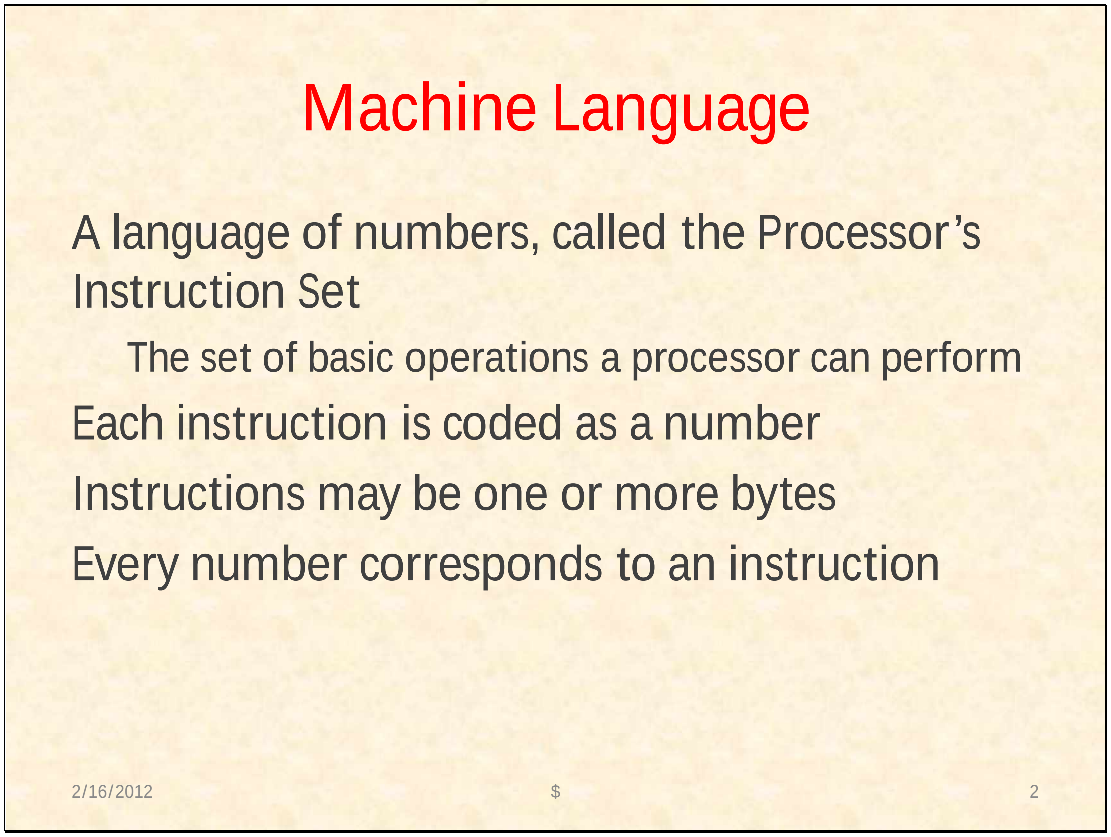
By

Ch.UMA SANKAR,

Dept. of ECE,

GIST.

2/16/2012 $ 1



Machine Language

A language of numbers, called the Processor's Instruction Set

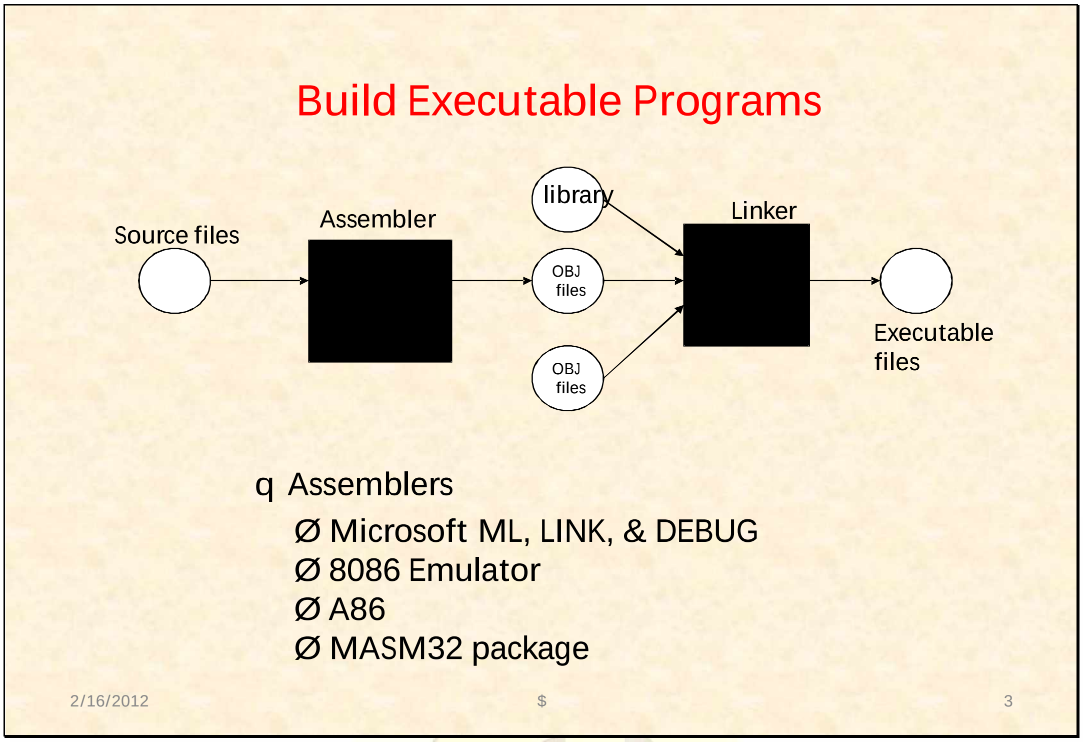
The set of basic operations a processor can perform

Each instruction is coded as a number

Instructions may be one or more bytes

Every number corresponds to an instruction

2/16/2012 $ 2



Build Executable Programs

Source files

Assembler

library

OBJ files

OBJ files

Linker

Executable files

q Assemblers

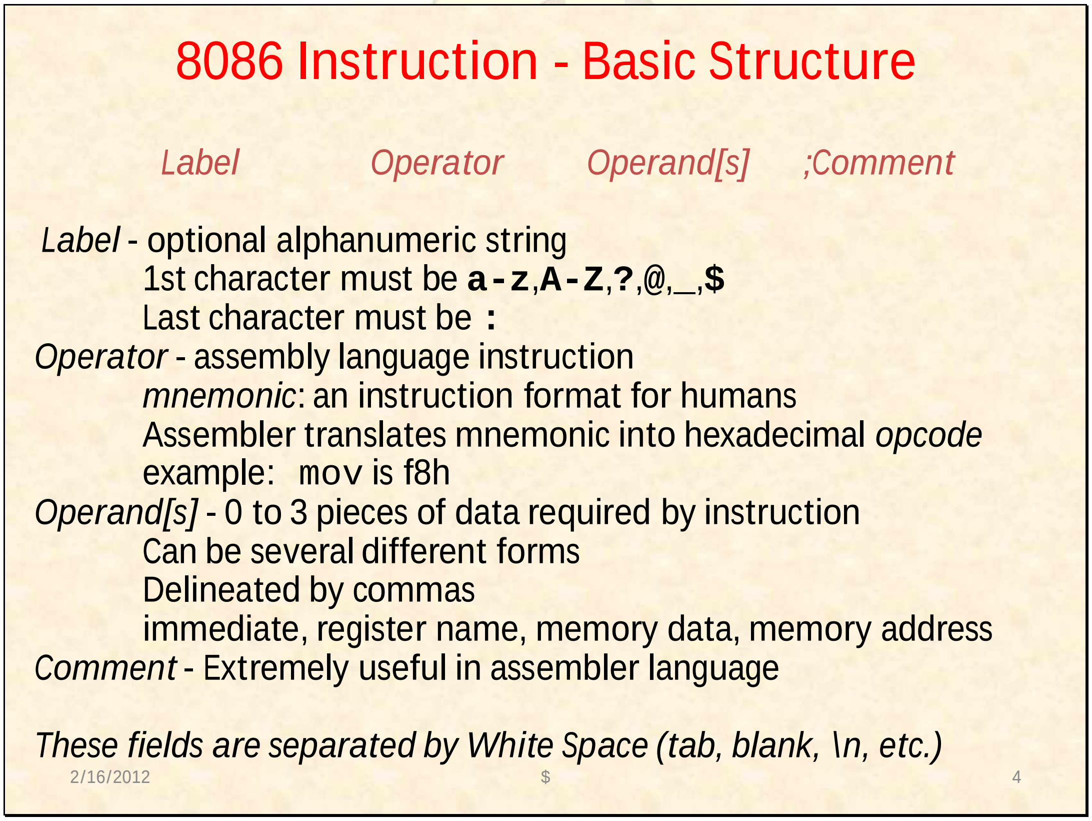
Ø Microsoft ML, LINK, & DEBUG

Ø 8086 Emulator

Ø A86

Ø MASM32 package

2/16/2012 $ 3



8086 Instruction - Basic Structure

Label *Operator* *Operand[s]* *;Comment*

Label - optional alphanumeric string
1st character must be **a-z**,**A-Z**,**?**,@,_,**$**
Last character must be **:**

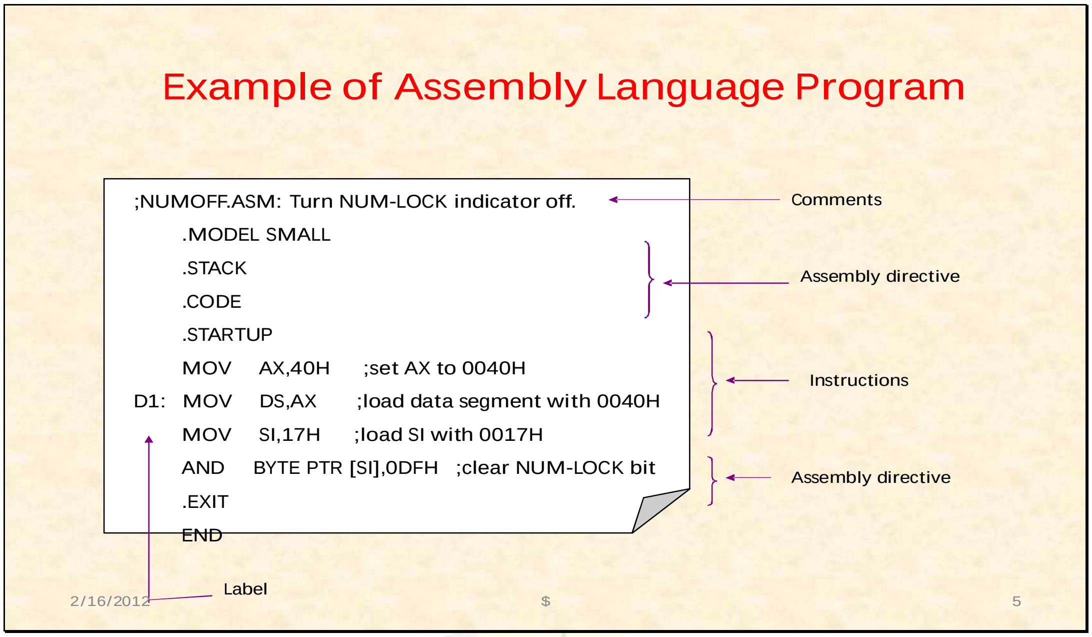
Operator - assembly language instruction
mnemonic: an instruction format for humans
Assembler translates mnemonic into hexadecimal *opcode*
example: `mov` is f8h

Operand[s] - 0 to 3 pieces of data required by instruction
Can be several different forms
Delineated by commas
immediate, register name, memory data, memory address

Comment - Extremely useful in assembler language

These fields are separated by White Space (tab, blank, \n, etc.)

2/16/2012 $ 4

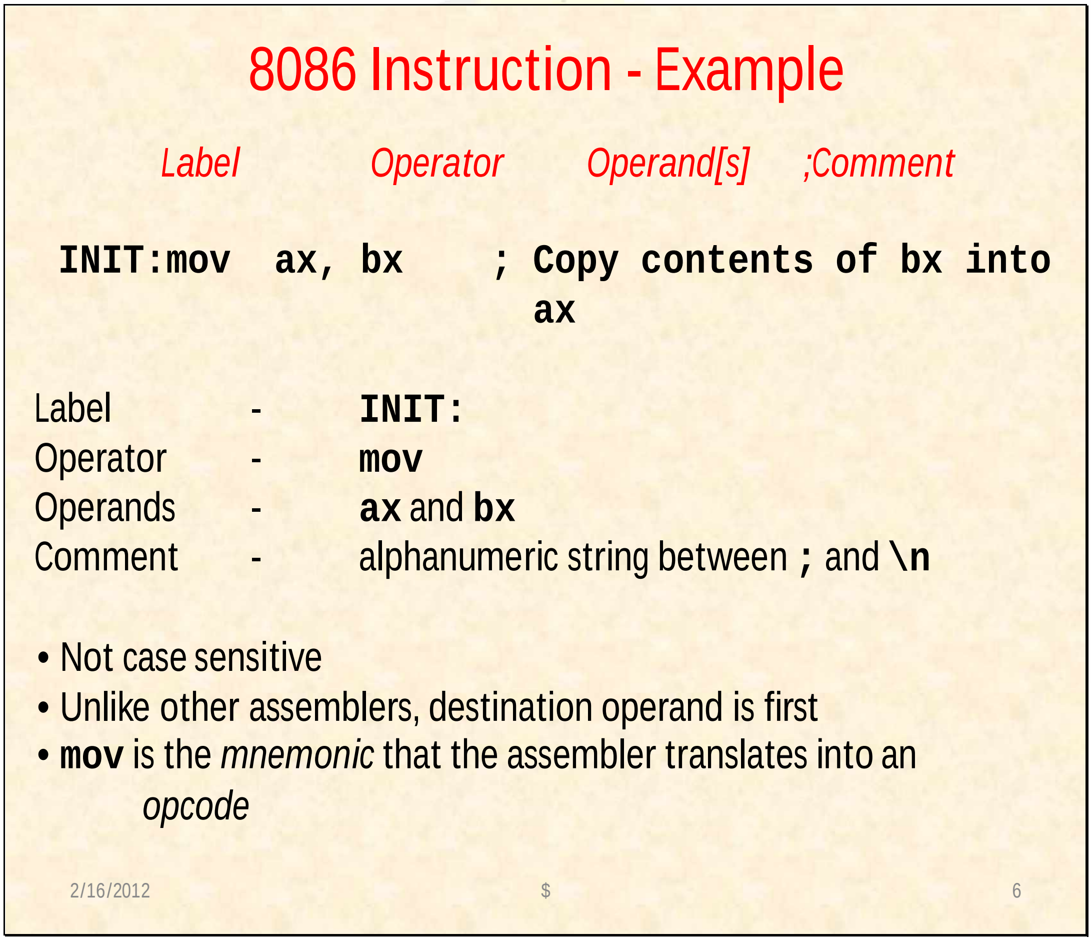


Example of Assembly Language Program

```
;NUMOFF.ASM: Turn NUM-LOCK indicator off.
       .MODEL SMALL
       .STACK
       .CODE
       .STARTUP
       MOV    AX,40H      ;set AX to 0040H
D1:    MOV    DS,AX       ;load data segment with 0040H
       MOV    SI,17H      ;load SI with 0017H
       AND    BYTE PTR [SI],0DFH   ;clear NUM-LOCK bit
       .EXIT
       END
```

Comments
Assembly directive
Instructions
Assembly directive
Label

2/16/2012 $ 5



8086 Instruction - Example

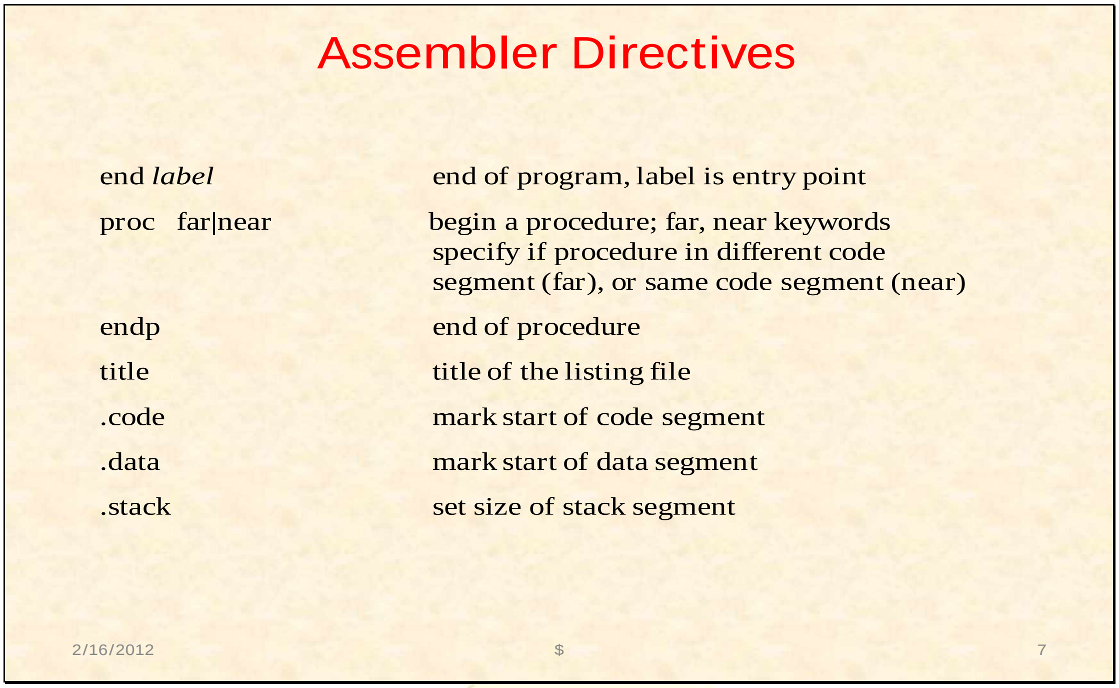
Label *Operator* *Operand[s]* *;Comment*

```
INIT:mov  ax, bx    ; Copy contents of bx into
                      ax
```

Label	-	**`INIT:`**
Operator	-	**`mov`**
Operands	-	**`ax`** and **`bx`**
Comment	-	alphanumeric string between **`;`** and **`\n`**

- Not case sensitive
- Unlike other assemblers, destination operand is first
- **`mov`** is the *mnemonic* that the assembler translates into an *opcode*

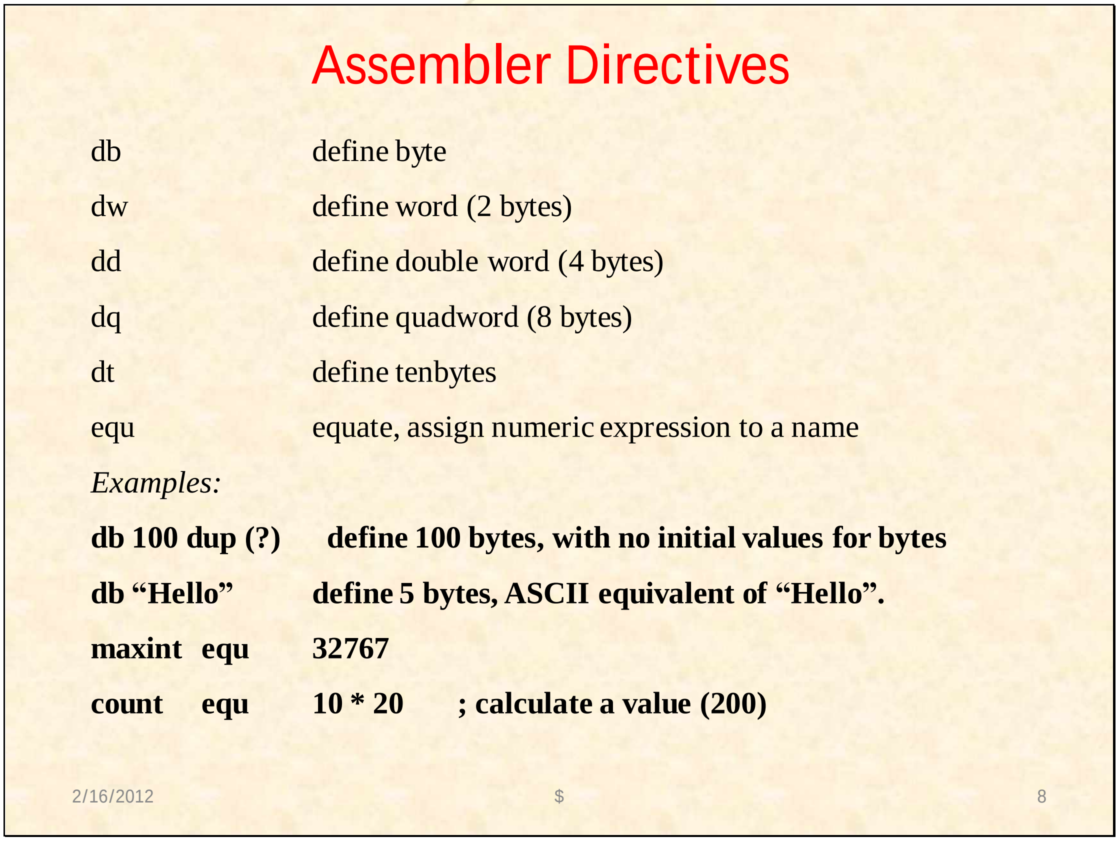
2/16/2012 $ 6



Assembler Directives

end *label*	end of program, label is entry point
proc far|near	begin a procedure; far, near keywords specify if procedure in different code segment (far), or same code segment (near)
endp	end of procedure
title	title of the listing file
.code	mark start of code segment
.data	mark start of data segment
.stack	set size of stack segment

2/16/2012 $ 7



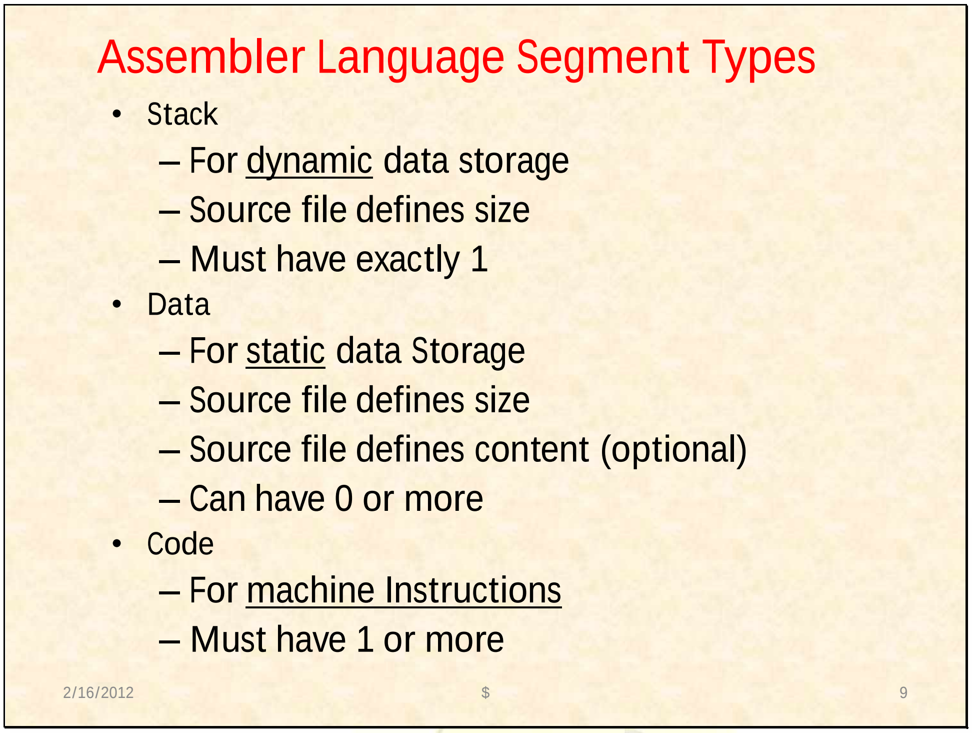
Assembler Directives

db	define byte
dw	define word (2 bytes)
dd	define double word (4 bytes)
dq	define quadword (8 bytes)
dt	define tenbytes
equ	equate, assign numeric expression to a name

Examples:

db 100 dup (?)	**define 100 bytes, with no initial values for bytes**
db "Hello"	**define 5 bytes, ASCII equivalent of "Hello".**
maxint equ	**32767**
count equ	**10 * 20 ; calculate a value (200)**

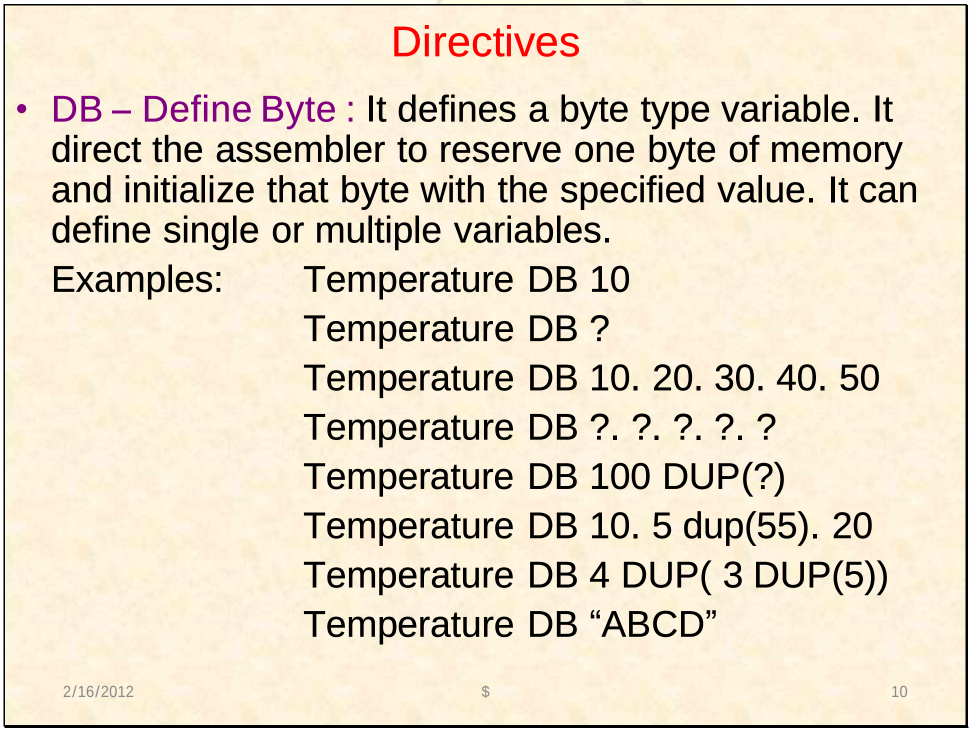
2/16/2012 $ 8



Assembler Language Segment Types

- Stack
  - – For dynamic data storage
  - – Source file defines size
  - – Must have exactly 1
- Data
  - – For static data Storage
  - – Source file defines size
  - – Source file defines content (optional)
  - – Can have 0 or more
- Code
  - – For machine Instructions
  - – Must have 1 or more

2/16/2012 $ 9



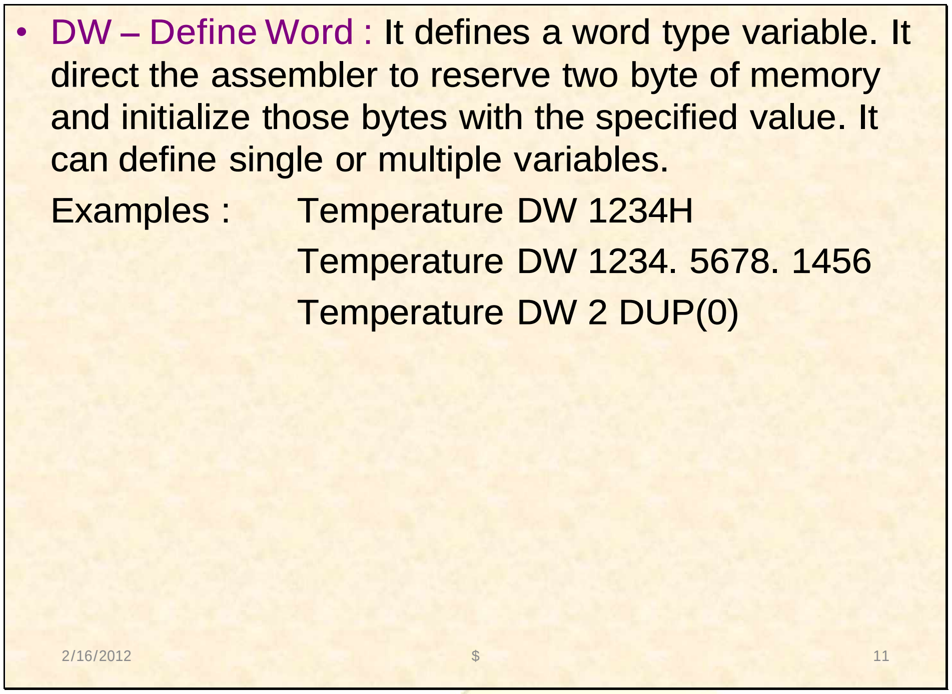
Directives

- DB – Define Byte : It defines a byte type variable. It direct the assembler to reserve one byte of memory and initialize that byte with the specified value. It can define single or multiple variables.

Examples:

```
Temperature DB 10
Temperature DB ?
Temperature DB 10. 20. 30. 40. 50
Temperature DB ?. ?. ?. ?. ?
Temperature DB 100 DUP(?)
Temperature DB 10. 5 dup(55). 20
Temperature DB 4 DUP( 3 DUP(5))
Temperature DB “ABCD”
```

2/16/2012 $ 10



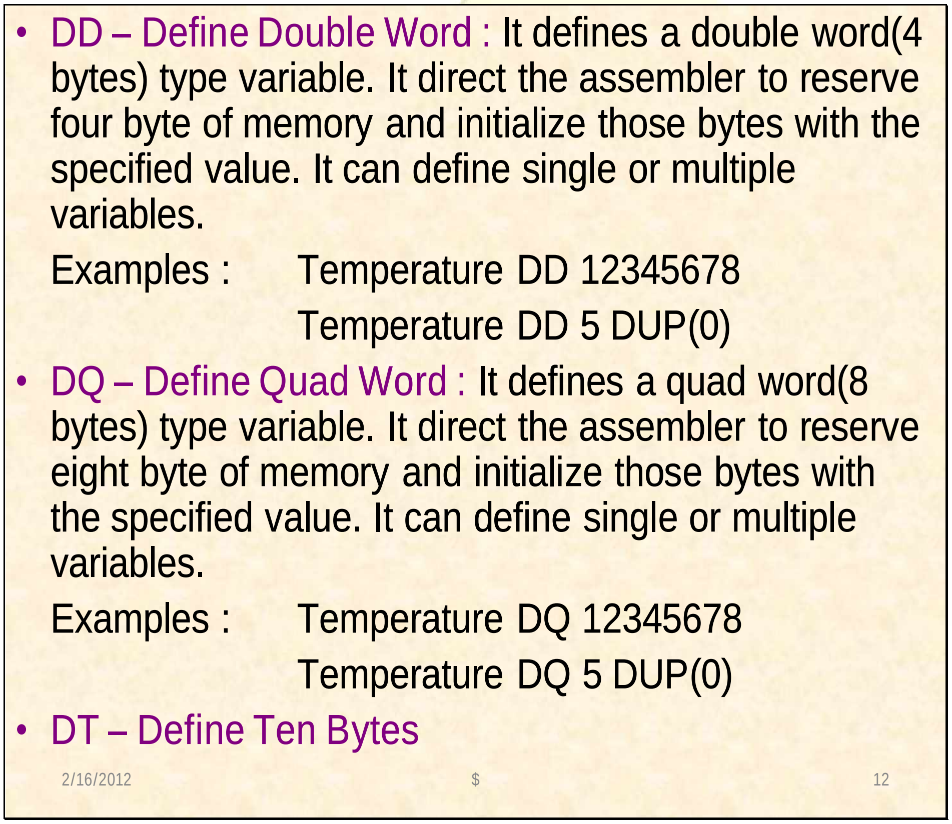
- DW – Define Word : It defines a word type variable. It direct the assembler to reserve two byte of memory and initialize those bytes with the specified value. It can define single or multiple variables.

  Examples : Temperature DW 1234H

  Temperature DW 1234. 5678. 1456

  Temperature DW 2 DUP(0)

2/16/2012 $ 11



- DD – Define Double Word : It defines a double word(4 bytes) type variable. It direct the assembler to reserve four byte of memory and initialize those bytes with the specified value. It can define single or multiple variables.

  Examples : Temperature DD 12345678

  Temperature DD 5 DUP(0)

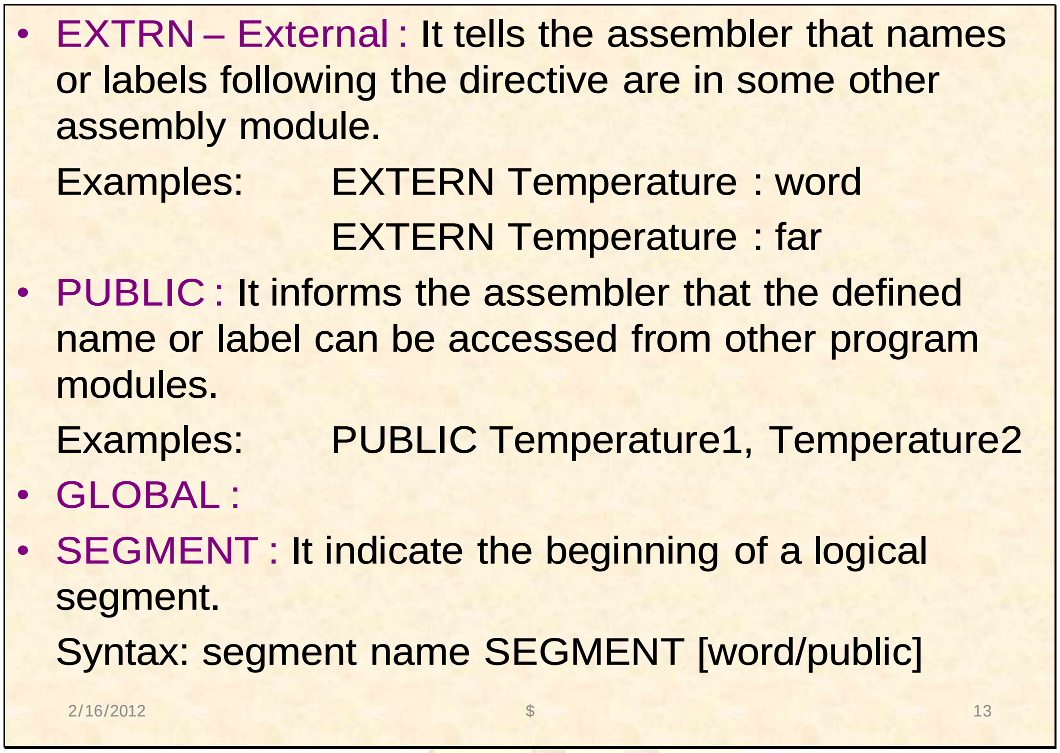
- DQ – Define Quad Word : It defines a quad word(8 bytes) type variable. It direct the assembler to reserve eight byte of memory and initialize those bytes with the specified value. It can define single or multiple variables.

  Examples : Temperature DQ 12345678

  Temperature DQ 5 DUP(0)

- DT – Define Ten Bytes

2/16/2012 $ 12



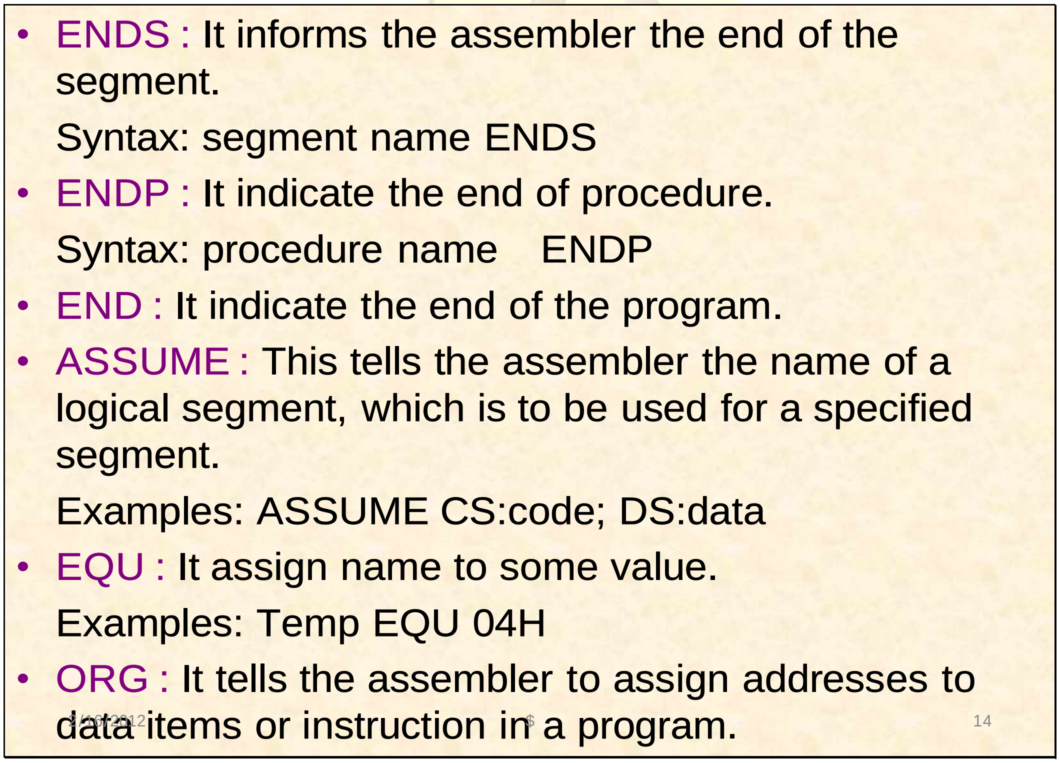
- EXTRN – External : It tells the assembler that names or labels following the directive are in some other assembly module.

  Examples: EXTERN Temperature : word

  EXTERN Temperature : far
- PUBLIC : It informs the assembler that the defined name or label can be accessed from other program modules.

  Examples: PUBLIC Temperature1, Temperature2
- GLOBAL :
- SEGMENT : It indicate the beginning of a logical segment.

  Syntax: segment name SEGMENT [word/public]



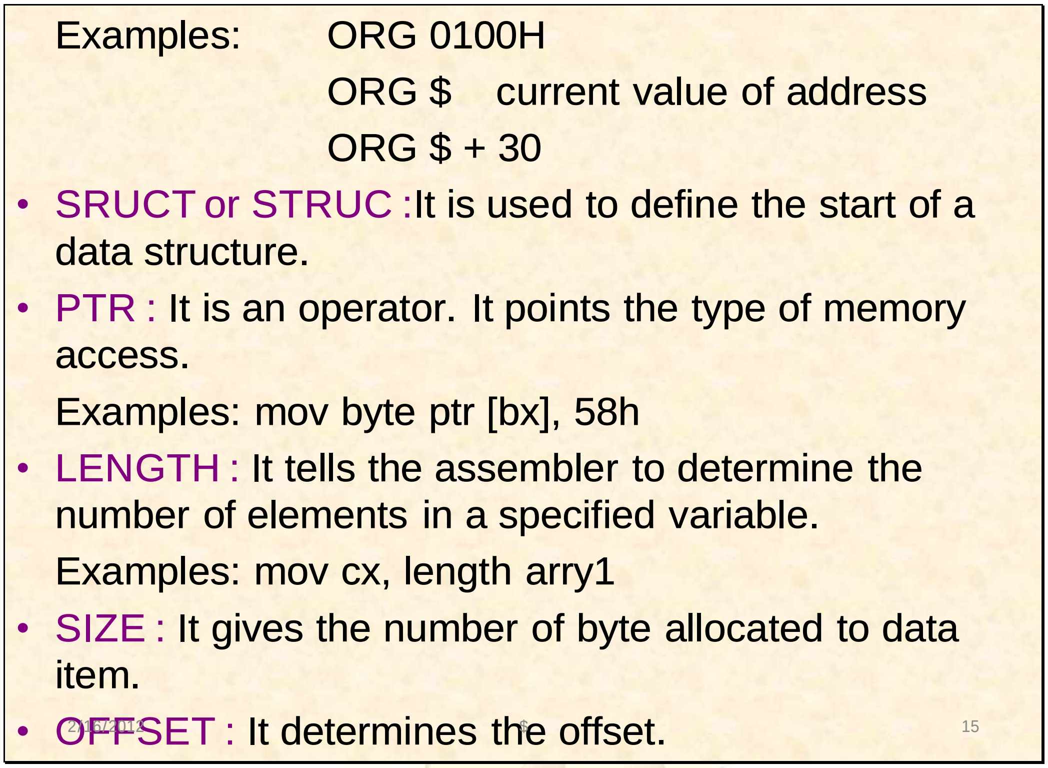
- ENDS : It informs the assembler the end of the segment.

  Syntax: segment name ENDS
- ENDP : It indicate the end of procedure.

  Syntax: procedure name ENDP
- END : It indicate the end of the program.
- ASSUME : This tells the assembler the name of a logical segment, which is to be used for a specified segment.

  Examples: ASSUME CS:code; DS:data
- EQU : It assign name to some value.

  Examples: Temp EQU 04H
- ORG : It tells the assembler to assign addresses to data items or instruction in a program.



Examples: ORG 0100H

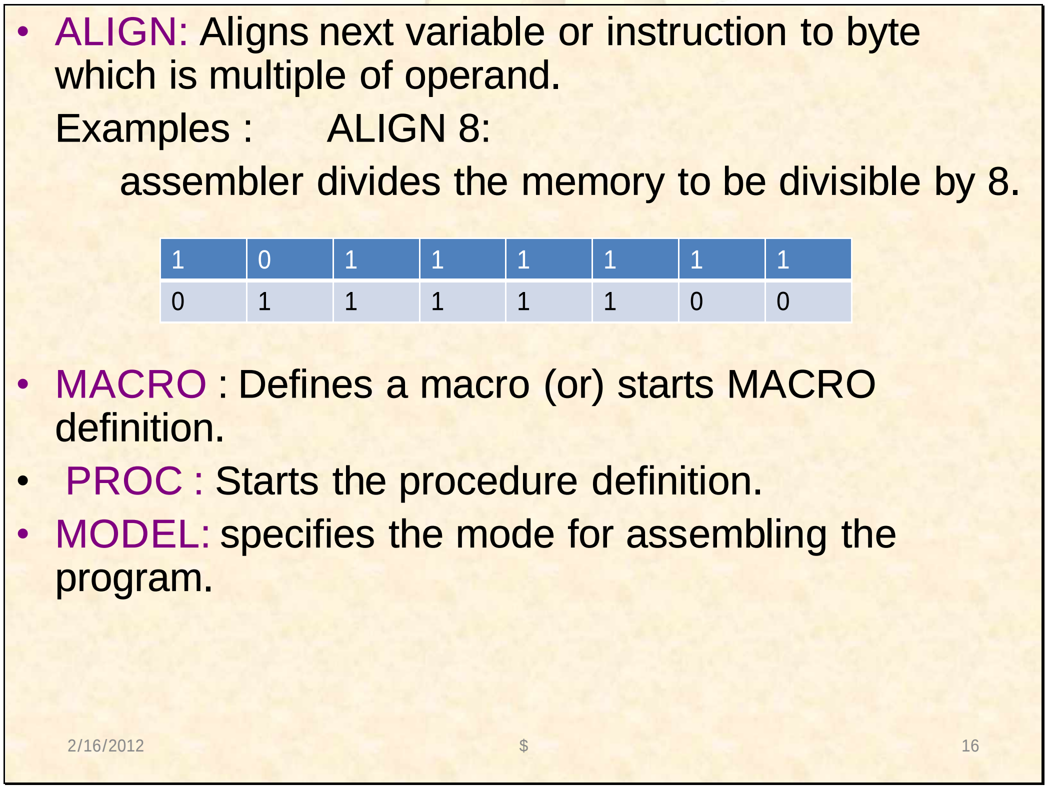
ORG $ current value of address

ORG $ + 30

- SRUCT or STRUC :It is used to define the start of a data structure.
- PTR : It is an operator. It points the type of memory access.

  Examples: mov byte ptr [bx], 58h
- LENGTH : It tells the assembler to determine the number of elements in a specified variable.

  Examples: mov cx, length arry1
- SIZE : It gives the number of byte allocated to data item.
- OFFSET : It determines the offset.



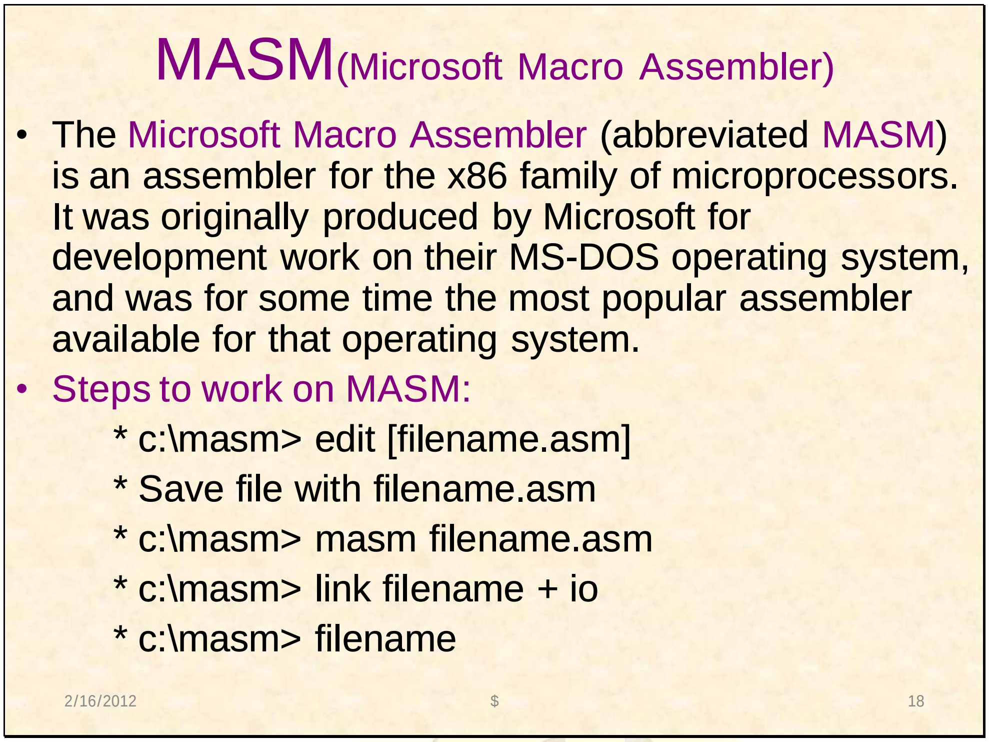
- ALIGN: Aligns next variable or instruction to byte which is multiple of operand.

  Examples : ALIGN 8:

  assembler divides the memory to be divisible by 8.

1	0	1	1	1	1	1	1
0	1	1	1	1	1	0	0

- MACRO : Defines a macro (or) starts MACRO definition.
- PROC : Starts the procedure definition.
- MODEL: specifies the mode for assembling the program.

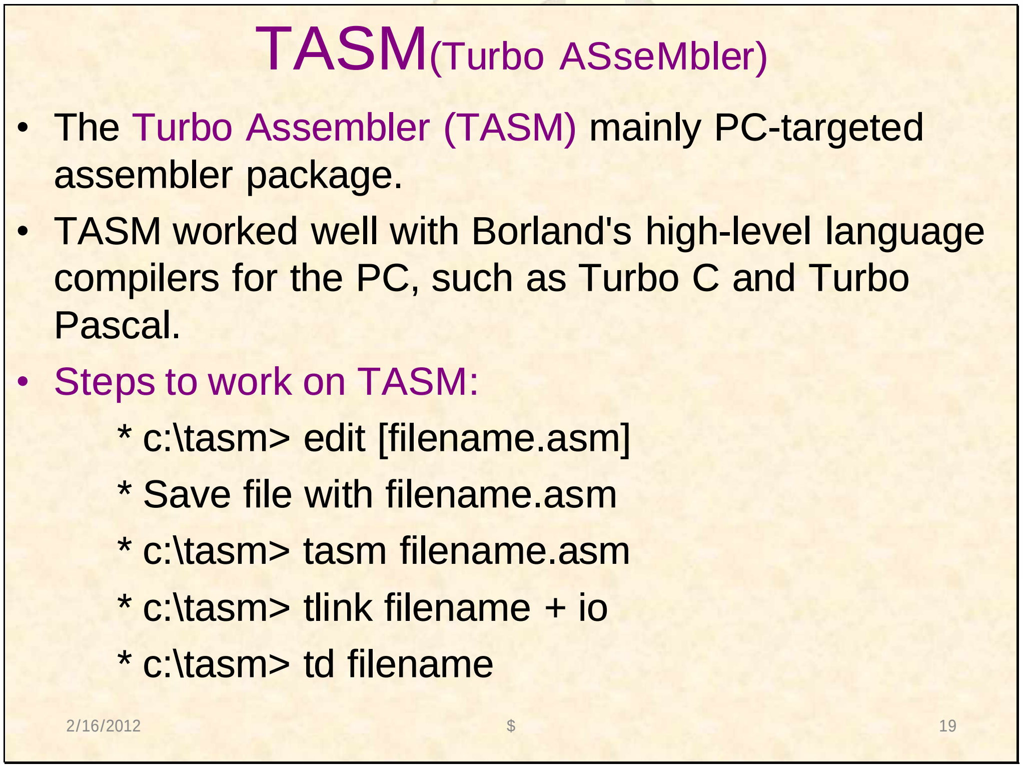


MASM(Microsoft Macro Assembler)

- The Microsoft Macro Assembler (abbreviated MASM) is an assembler for the x86 family of microprocessors. It was originally produced by Microsoft for development work on their MS-DOS operating system, and was for some time the most popular assembler available for that operating system.
- Steps to work on MASM:

```
* c:\masm> edit [filename.asm]
* Save file with filename.asm
* c:\masm> masm filename.asm
* c:\masm> link filename + io
* c:\masm> filename
```

2/16/2012 $ 18



TASM(Turbo ASseMbler)

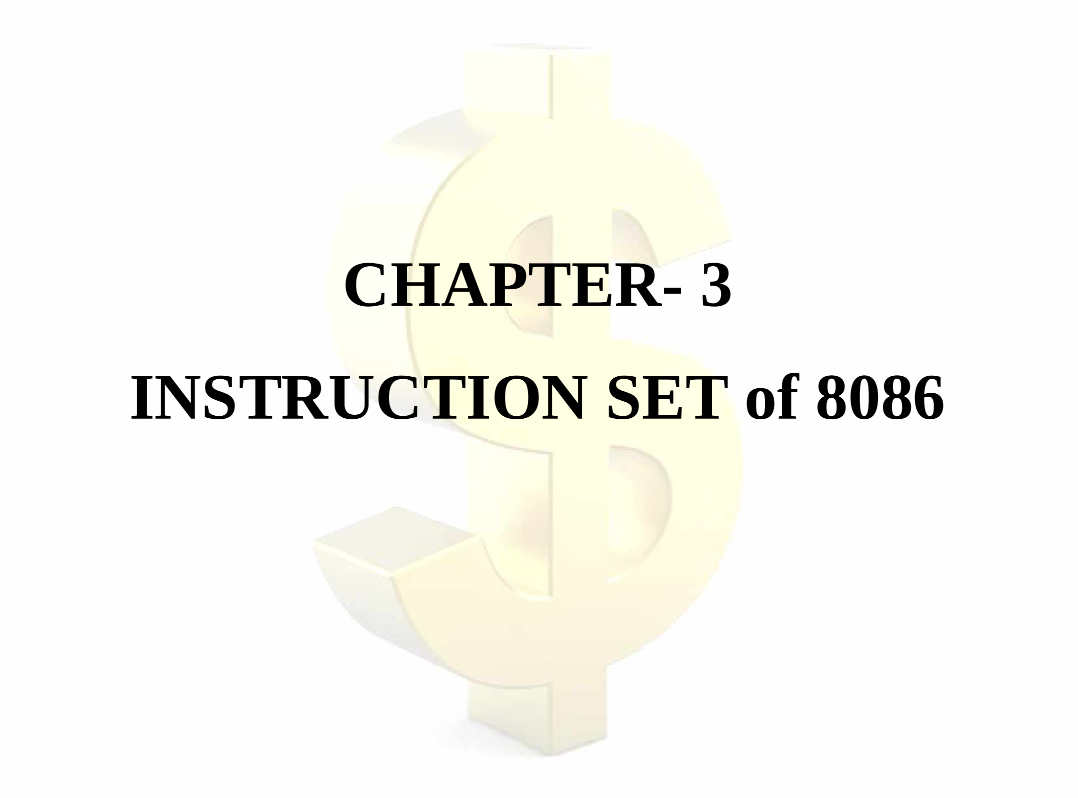
- The Turbo Assembler (TASM) mainly PC-targeted assembler package.
- TASM worked well with Borland's high-level language compilers for the PC, such as Turbo C and Turbo Pascal.
- Steps to work on TASM:

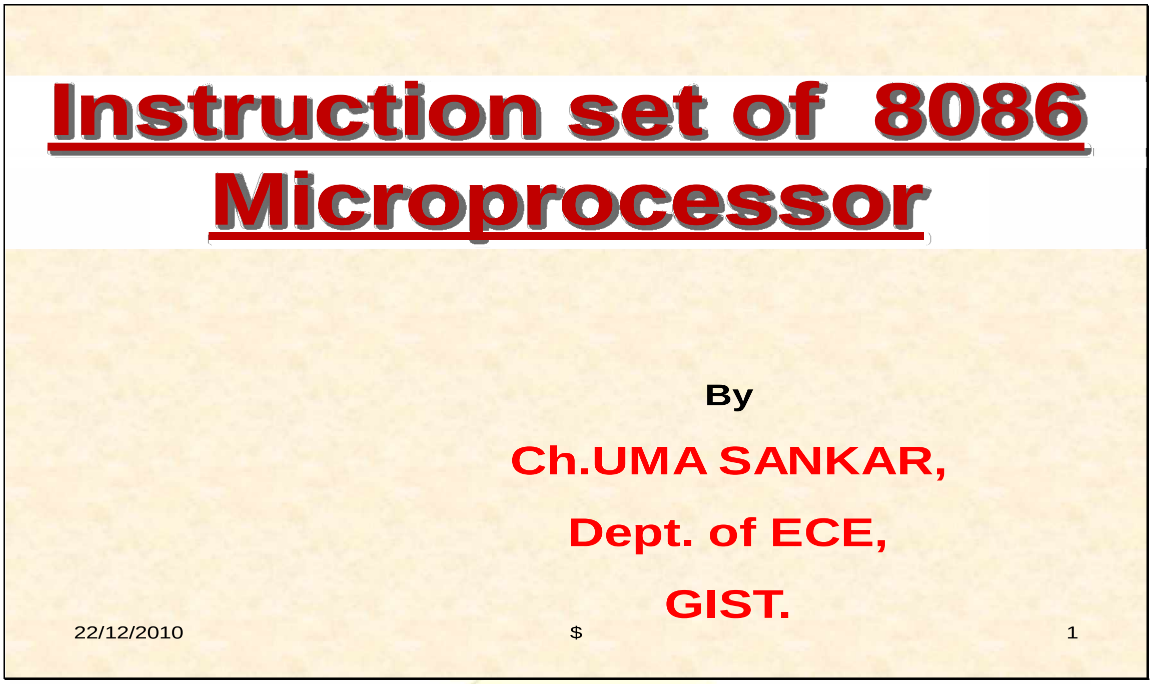
```
* c:\tasm> edit [filename.asm]
* Save file with filename.asm
* c:\tasm> tasm filename.asm
* c:\tasm> tlink filename + io
* c:\tasm> td filename
```

2/16/2012 $ 19

CHAPTER- 3

INSTRUCTION SET of 8086





Instruction set of 8086 Microprocessor

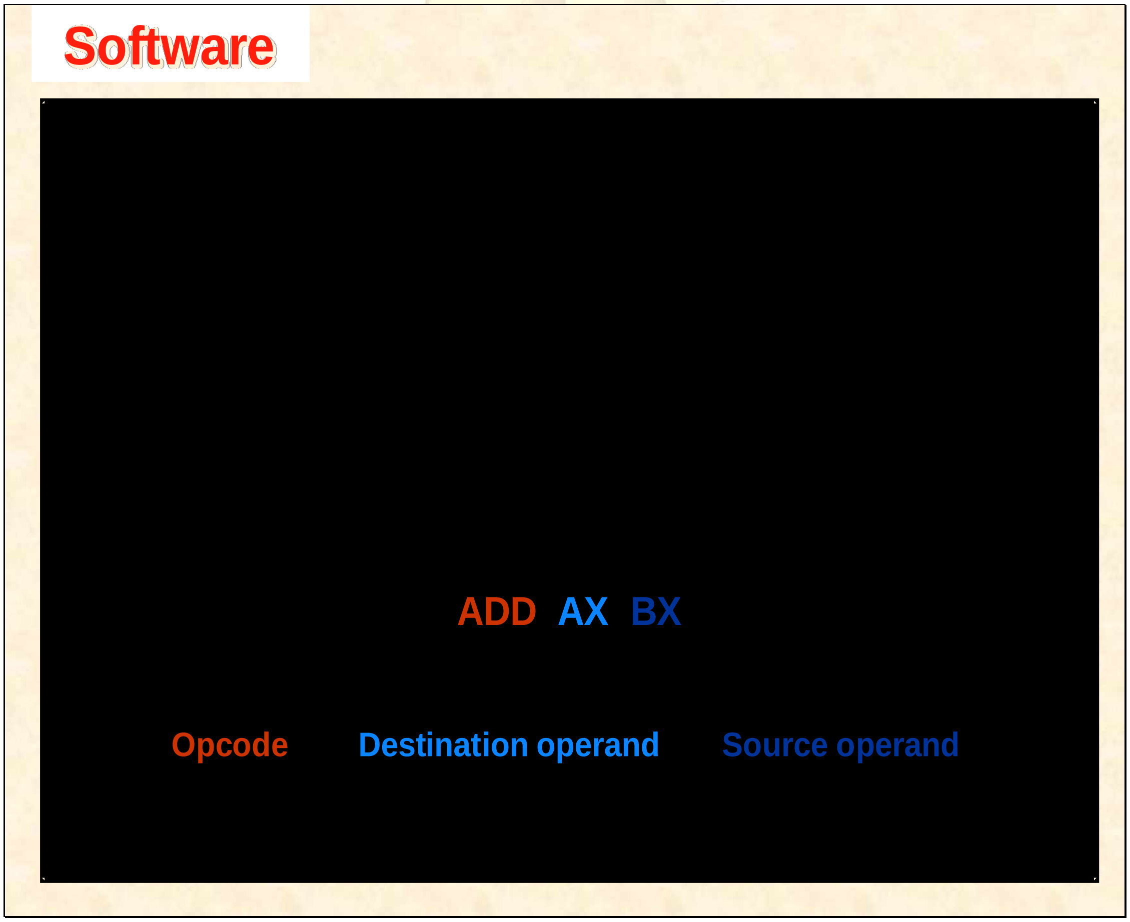
By

Ch.UMA SANKAR,

Dept. of ECE,

GIST.

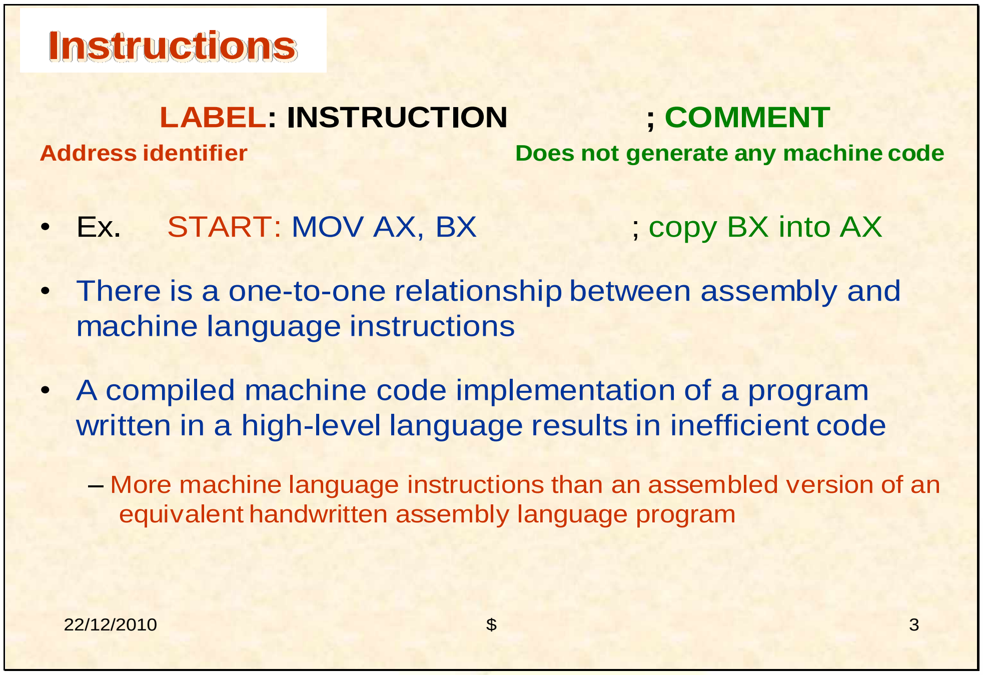
22/12/2010 $ 1



Software

ADD AX BX

Opcode Destination operand Source operand



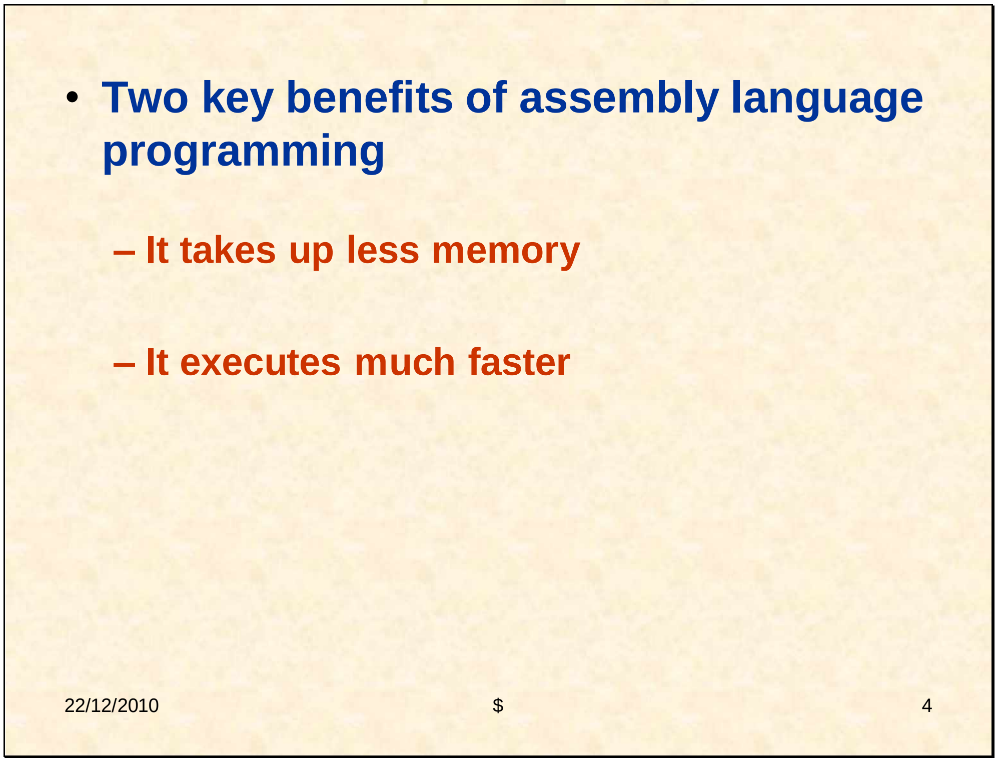
Instructions

LABEL: INSTRUCTION ; COMMENT

Address identifier **Does not generate any machine code**

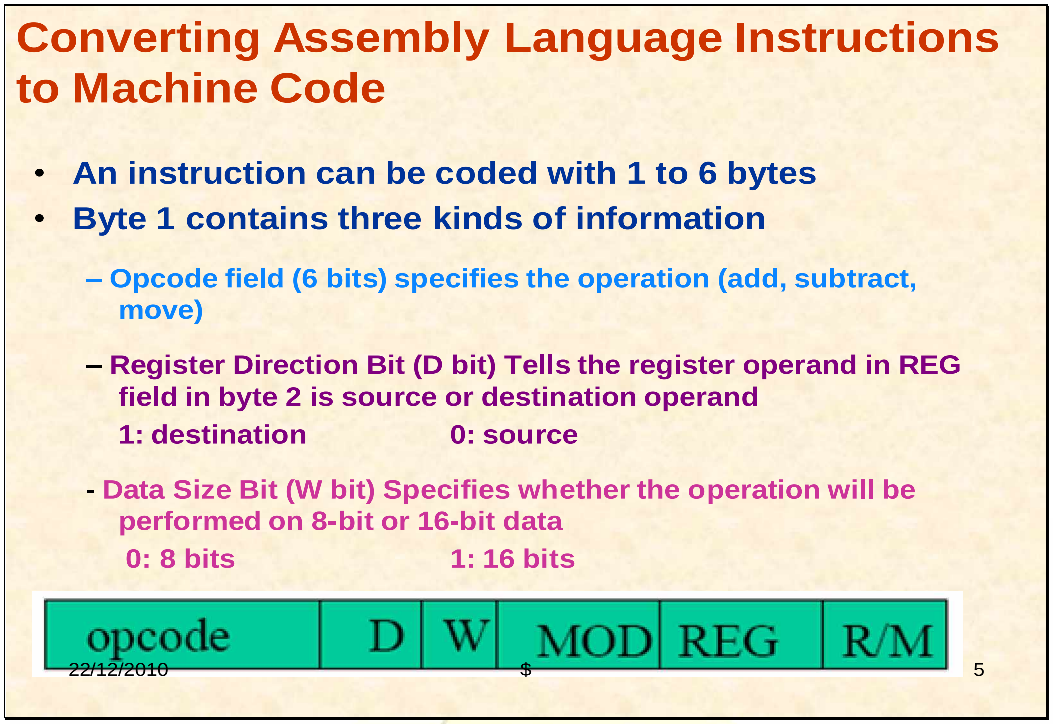
- Ex. START: MOV AX, BX ; copy BX into AX
- There is a one-to-one relationship between assembly and machine language instructions
- A compiled machine code implementation of a program written in a high-level language results in inefficient code
  - More machine language instructions than an assembled version of an equivalent handwritten assembly language program

22/12/2010 $ 3



- **Two key benefits of assembly language programming**
  - **It takes up less memory**
  - **It executes much faster**

22/12/2010 $ 4



Converting Assembly Language Instructions to Machine Code

- An instruction can be coded with 1 to 6 bytes
- Byte 1 contains three kinds of information

– Opcode field (6 bits) specifies the operation (add, subtract, move)

– Register Direction Bit (D bit) Tells the register operand in REG field in byte 2 is source or destination operand

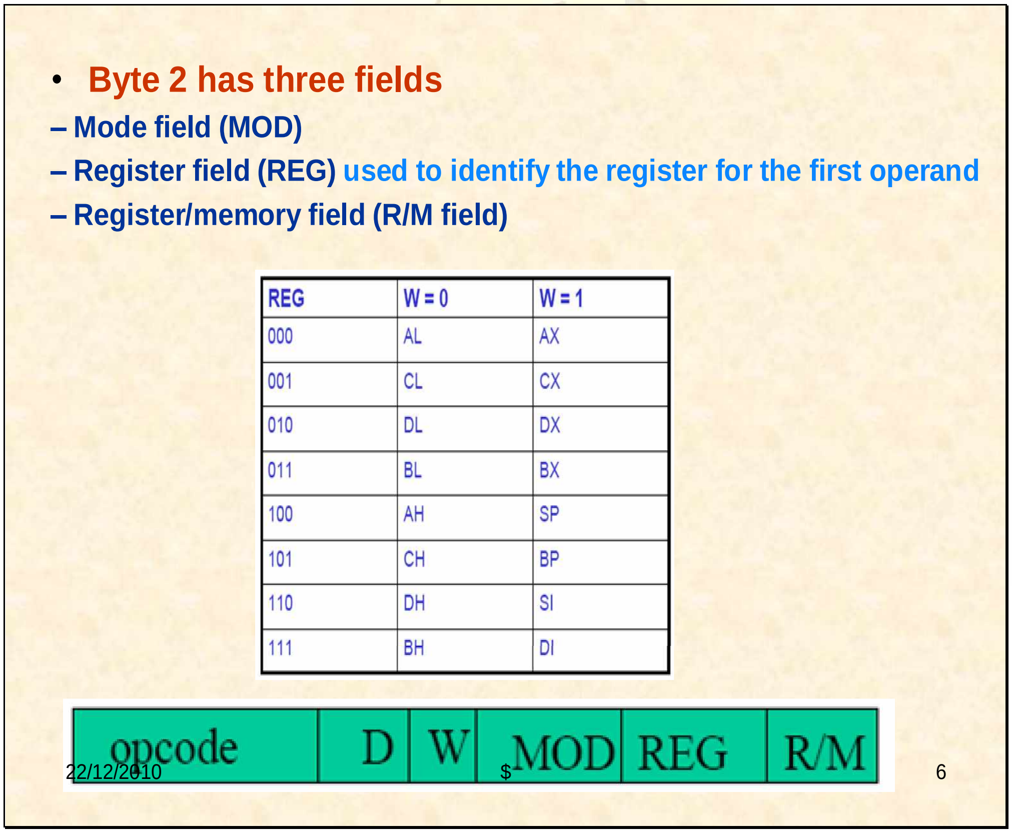
1: destination 0: source

- Data Size Bit (W bit) Specifies whether the operation will be performed on 8-bit or 16-bit data

0: 8 bits 1: 16 bits

opcode	D	W	MOD	REG	R/M

22/12/2010 $ 5



- Byte 2 has three fields

– Mode field (MOD)

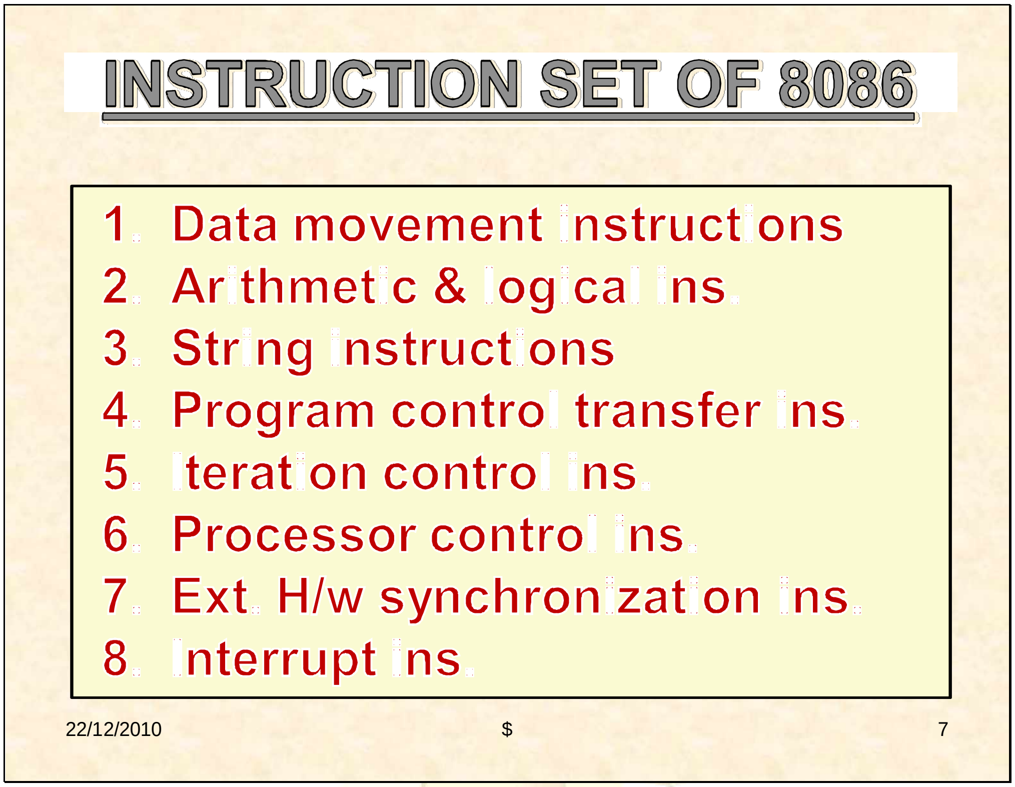
– Register field (REG) used to identify the register for the first operand

– Register/memory field (R/M field)

REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

opcode	D	W	MOD	REG	R/M

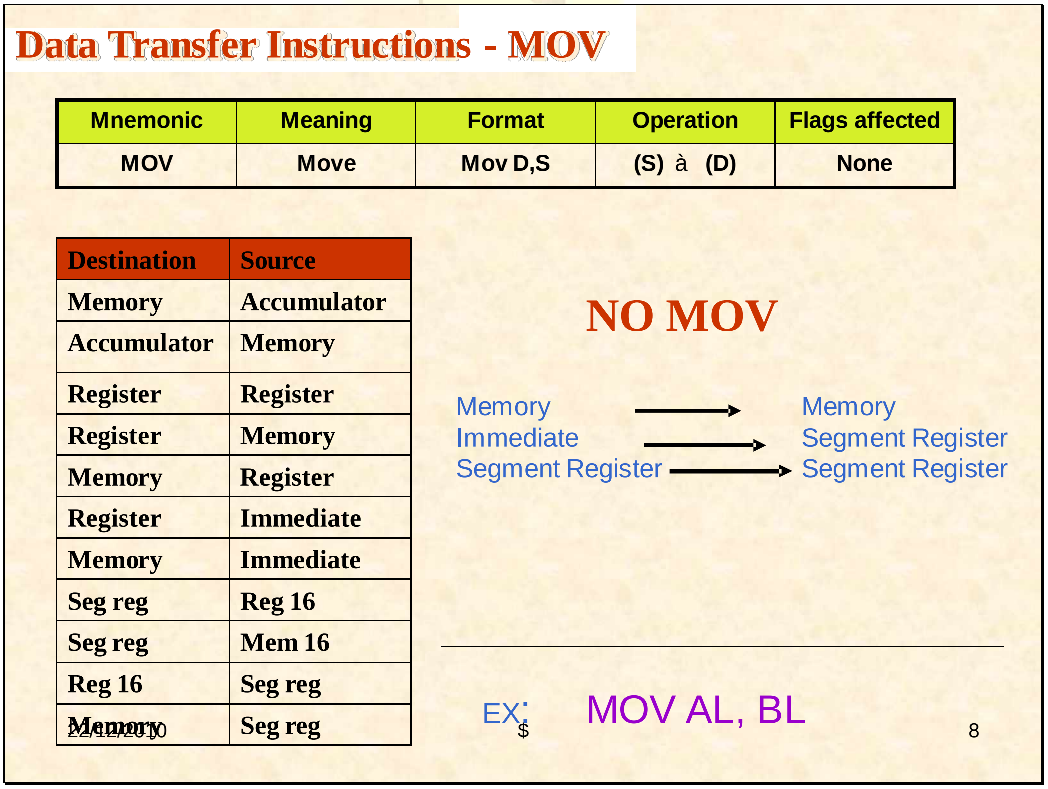
22/12/2010 $ 6



INSTRUCTION SET OF 8086

1. Data movement instructions
2. Arithmetic & logical ins.
3. String instructions
4. Program control transfer ins.
5. Iteration control ins.
6. Processor control ins.
7. Ext. H/w synchronization ins.
8. Interrupt ins.

22/12/2010 $ 7



Data Transfer Instructions - MOV

Mnemonic	Meaning	Format	Operation	Flags affected
MOV	Move	Mov D,S	(S) à (D)	None

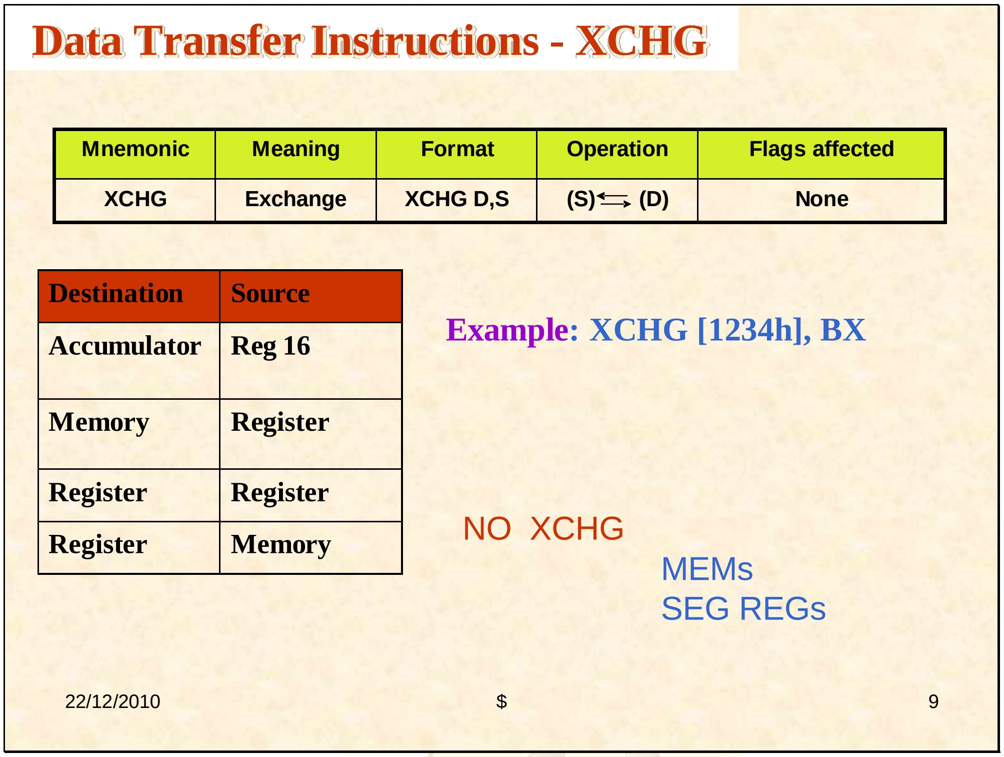
Destination	Source
Memory	Accumulator
Accumulator	Memory
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Seg reg	Reg 16
Seg reg	Mem 16
Reg 16	Seg reg
Memory	Seg reg

NO MOV

Memory → Memory
Immediate → Segment Register
Segment Register → Segment Register

EX: MOV AL, BL

22/12/2010 $ 8



Data Transfer Instructions - XCHG

Mnemonic	Meaning	Format	Operation	Flags affected
XCHG	Exchange	XCHG D,S	(S) ⇆ (D)	None

Destination	Source
Accumulator	Reg 16
Memory	Register
Register	Register
Register	Memory

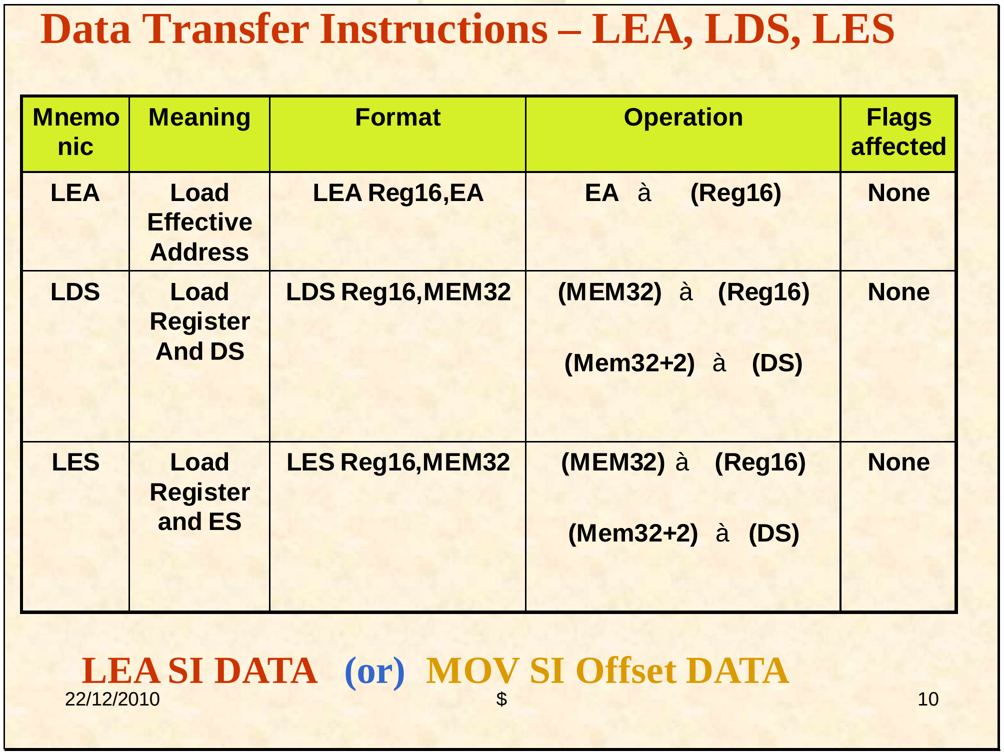
Example: XCHG [1234h], BX

NO XCHG

MEMs

SEG REGs

22/12/2010 $ 9

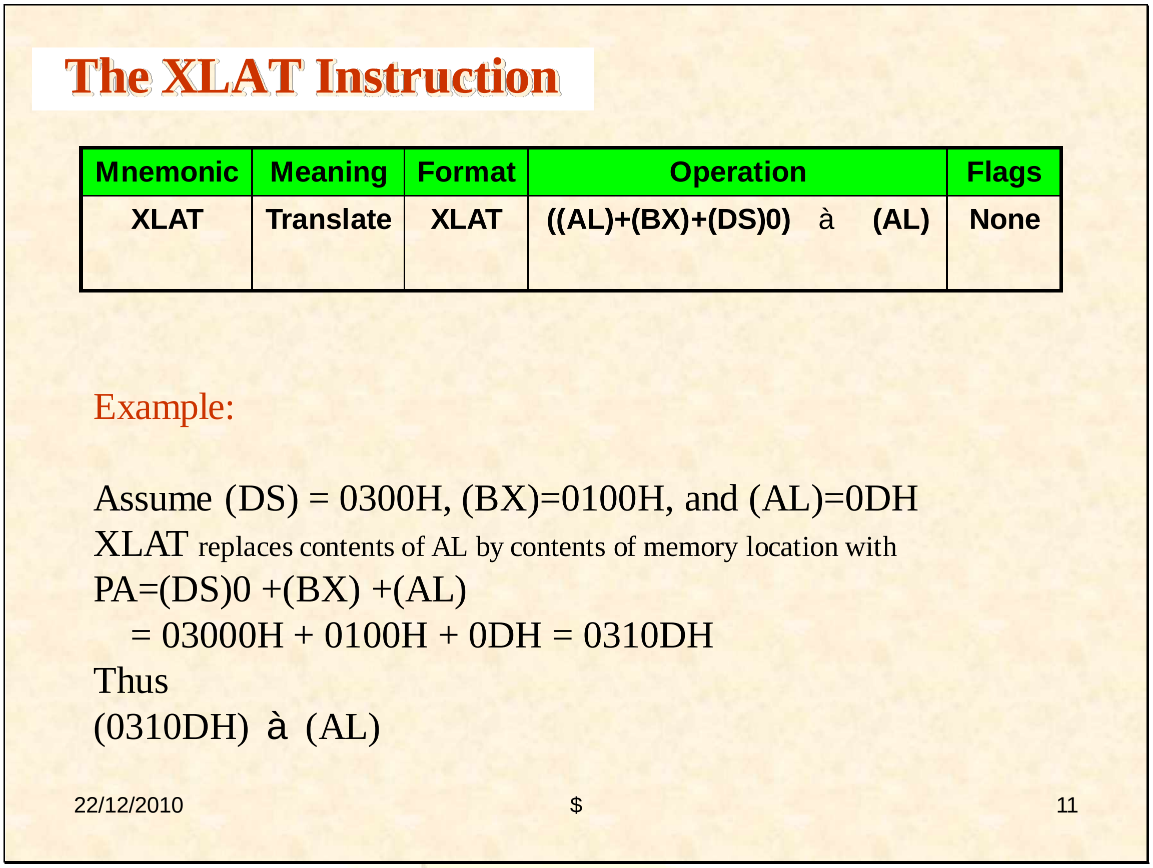


Data Transfer Instructions – LEA, LDS, LES

Mnemonic	Meaning	Format	Operation	Flags affected
LEA	Load Effective Address	LEA Reg16,EA	EA à (Reg16)	None
LDS	Load Register And DS	LDS Reg16,MEM32	(MEM32) à (Reg16) (Mem32+2) à (DS)	None
LES	Load Register and ES	LES Reg16,MEM32	(MEM32) à (Reg16) (Mem32+2) à (DS)	None

LEA SI DATA (or) MOV SI Offset DATA

22/12/2010 $ 10



The XLAT Instruction

Mnemonic	Meaning	Format	Operation	Flags
XLAT	Translate	XLAT	((AL)+(BX)+(DS)0) à (AL)	None

Example:

Assume (DS) = 0300H, (BX)=0100H, and (AL)=0DH

XLAT replaces contents of AL by contents of memory location with

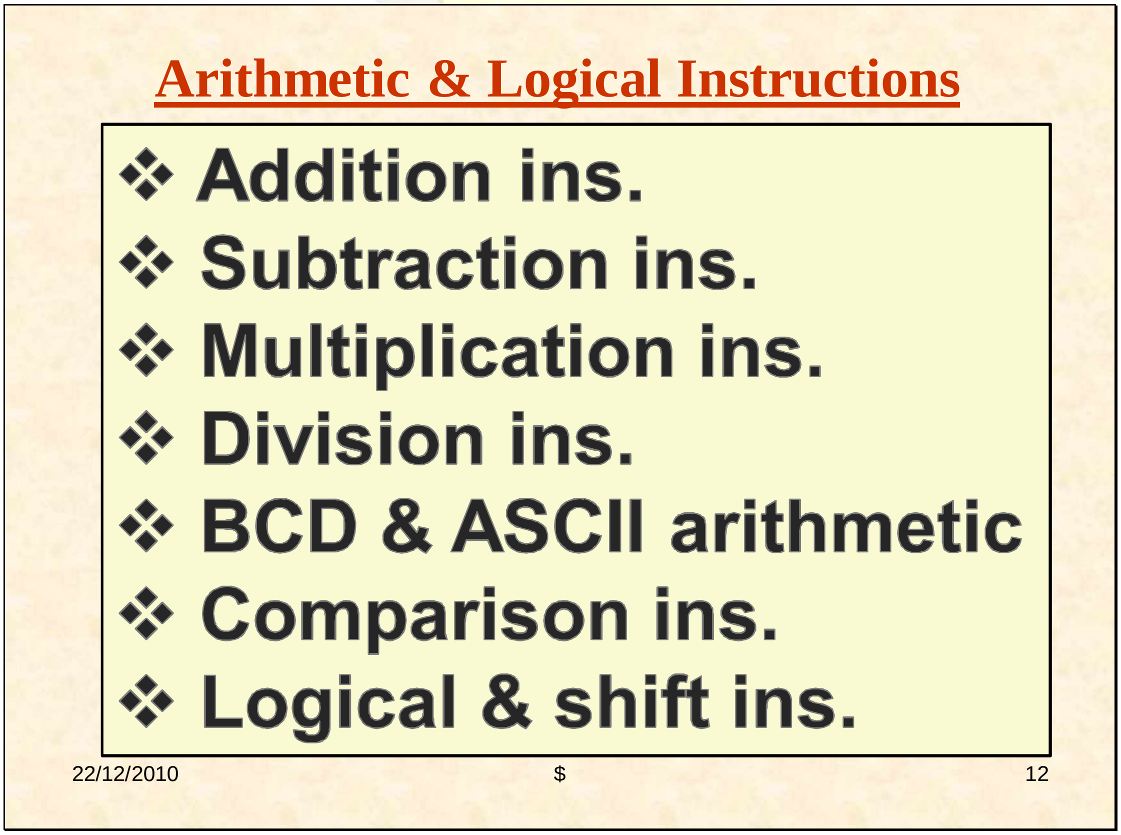
PA=(DS)0 +(BX) +(AL)

= 03000H + 0100H + 0DH = 0310DH

Thus

(0310DH) à (AL)

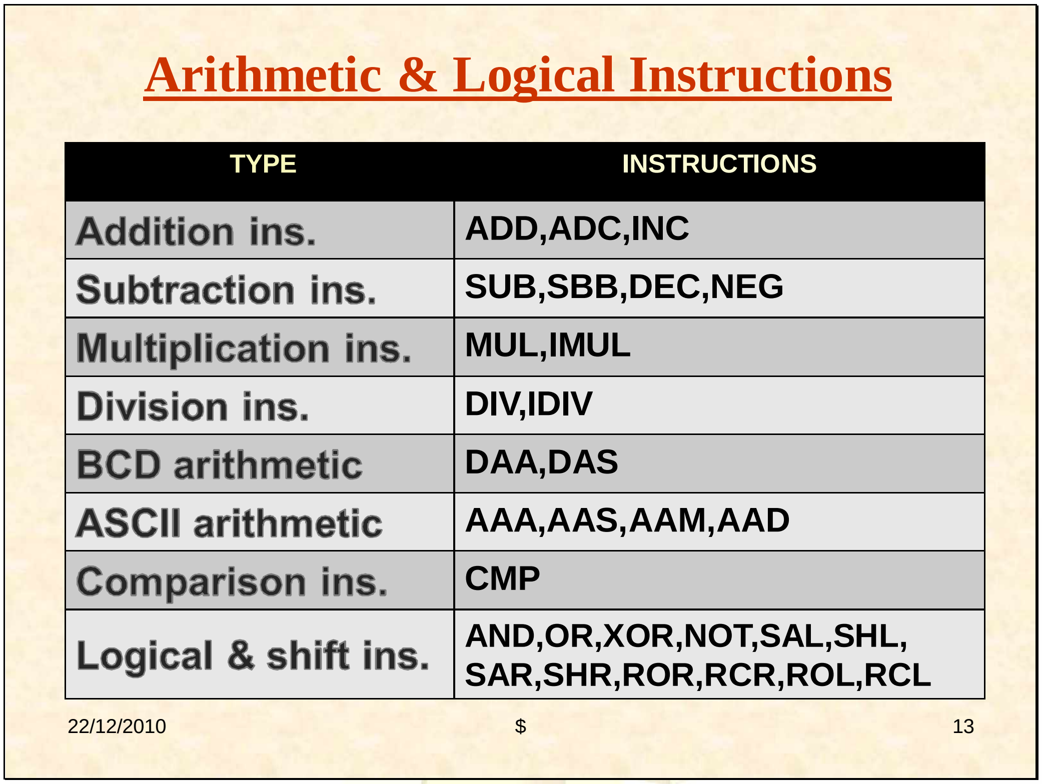
22/12/2010 $ 11



Arithmetic & Logical Instructions

- Addition ins.
- Subtraction ins.
- Multiplication ins.
- Division ins.
- BCD & ASCII arithmetic
- Comparison ins.
- Logical & shift ins.

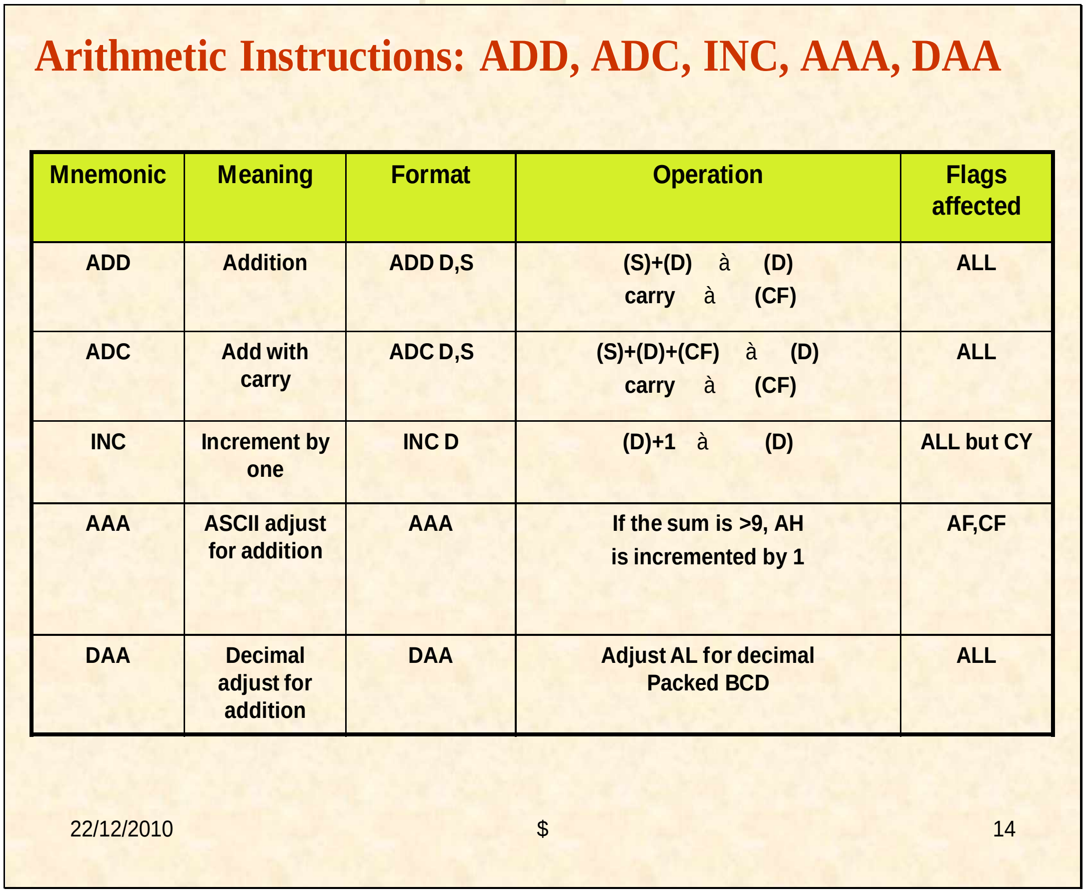
22/12/2010 $ 12



Arithmetic & Logical Instructions

TYPE	INSTRUCTIONS
Addition ins.	ADD,ADC,INC
Subtraction ins.	SUB,SBB,DEC,NEG
Multiplication ins.	MUL,IMUL
Division ins.	DIV,IDIV
BCD arithmetic	DAA,DAS
ASCII arithmetic	AAA,AAS,AAM,AAD
Comparison ins.	CMP
Logical & shift ins.	AND,OR,XOR,NOT,SAL,SHL, SAR,SHR,ROR,RCR,ROL,RCL

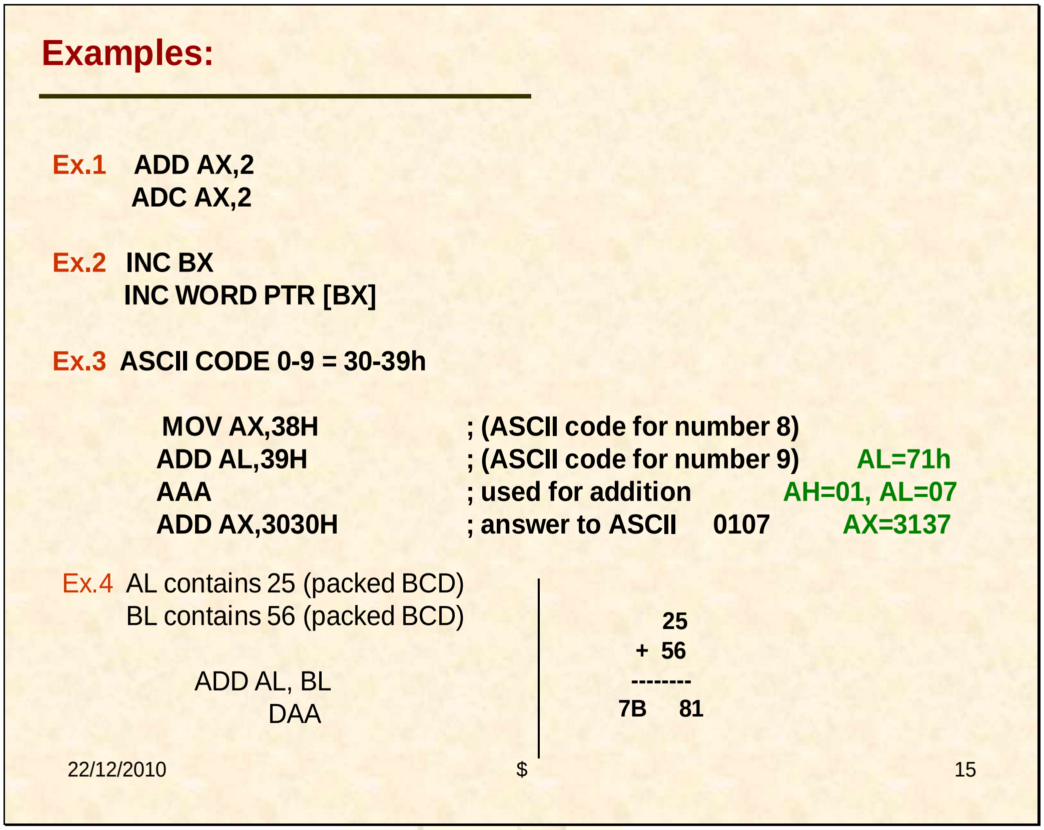
22/12/2010 $ 13



Arithmetic Instructions: ADD, ADC, INC, AAA, DAA

Mnemonic	Meaning	Format	Operation	Flags affected
ADD	Addition	ADD D,S	(S)+(D) à (D) carry à (CF)	ALL
ADC	Add with carry	ADC D,S	(S)+(D)+(CF) à (D) carry à (CF)	ALL
INC	Increment by one	INC D	(D)+1 à (D)	ALL but CY
AAA	ASCII adjust for addition	AAA	If the sum is >9, AH is incremented by 1	AF,CF
DAA	Decimal adjust for addition	DAA	Adjust AL for decimal Packed BCD	ALL

22/12/2010 $ 14



Examples:

Ex.1 ADD AX,2
ADC AX,2

Ex.2 INC BX
INC WORD PTR [BX]

Ex.3 ASCII CODE 0-9 = 30-39h

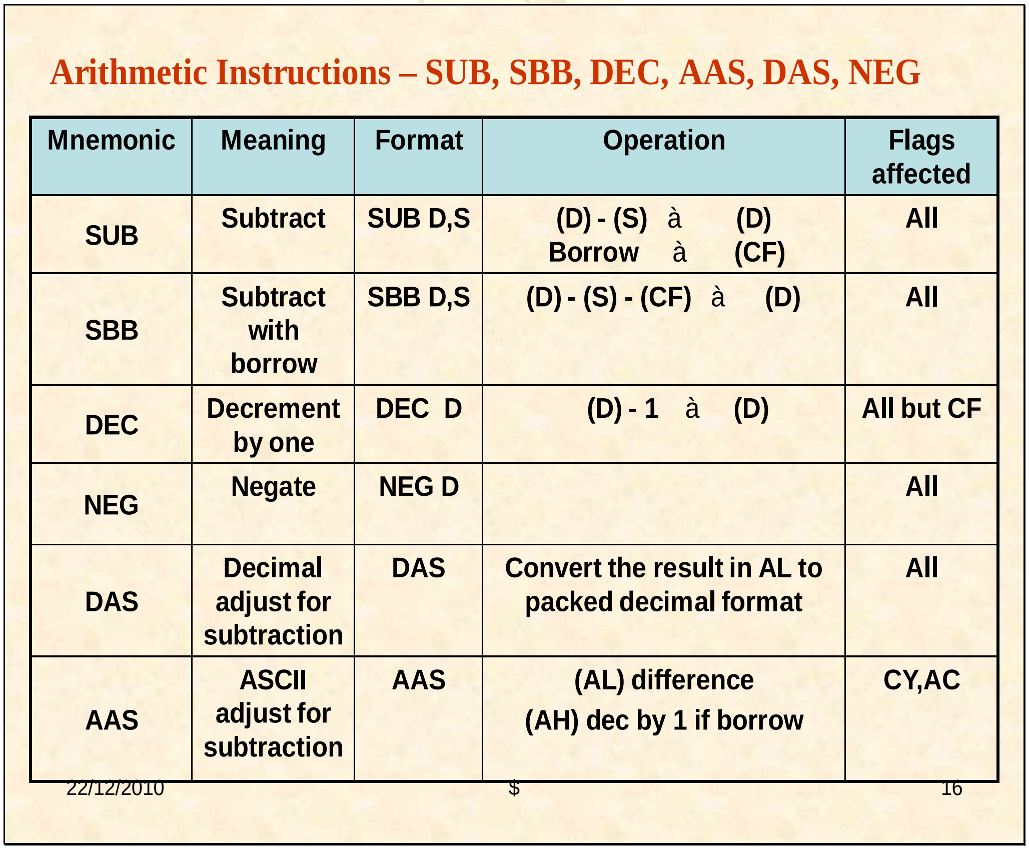
```
MOV AX,38H        ; (ASCII code for number 8)
ADD AL,39H        ; (ASCII code for number 9)        AL=71h
AAA               ; used for addition            AH=01, AL=07
ADD AX,3030H      ; answer to ASCII     0107         AX=3137
```

Ex.4 AL contains 25 (packed BCD)
BL contains 56 (packed BCD)

```
ADD AL, BL
    DAA
```

```
   25
 + 56
--------
7B    81
```

22/12/2010 $ 15



Arithmetic Instructions – SUB, SBB, DEC, AAS, DAS, NEG

Mnemonic	Meaning	Format	Operation	Flags affected
SUB	Subtract	SUB D,S	(D) - (S) à (D) Borrow à (CF)	All
SBB	Subtract with borrow	SBB D,S	(D) - (S) - (CF) à (D)	All
DEC	Decrement by one	DEC D	(D) - 1 à (D)	All but CF
NEG	Negate	NEG D		All
DAS	Decimal adjust for subtraction	DAS	Convert the result in AL to packed decimal format	All
AAS	ASCII adjust for subtraction	AAS	(AL) difference (AH) dec by 1 if borrow	CY,AC

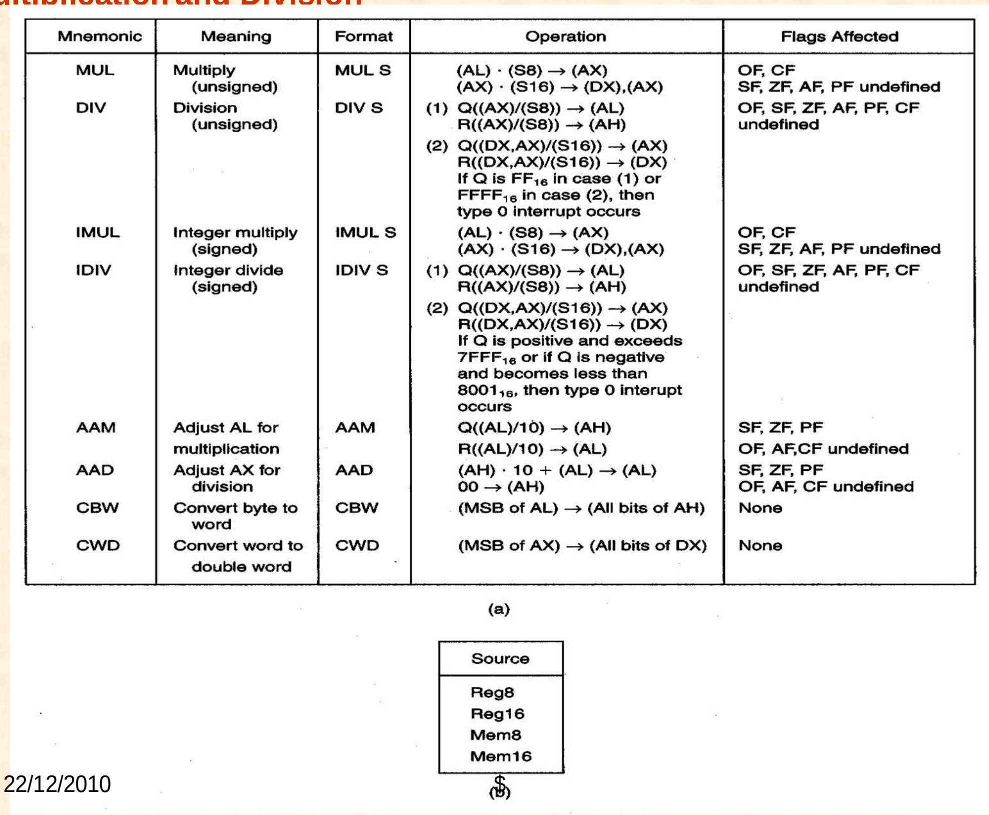
22/12/2010 $ 16

Examples: DAS

AL=5BH
adjust as AL=55H

AX=FF09 ten’s complement of -1 (Borrow one from AH)
AL=39

Multiplication and Division



Mnemonic	Meaning	Format	Operation	Flags Affected
MUL	Multiply (unsigned)	MUL S	(AL) · (S8) → (AX) (AX) · (S16) → (DX),(AX)	OF, CF SF, ZF, AF, PF undefined
DIV	Division (unsigned)	DIV S	(1) Q((AX)/(S8)) → (AL) R((AX)/(S8)) → (AH) (2) Q((DX,AX)/(S16)) → (AX) R((DX,AX)/(S16)) → (DX) If Q is FF_{16} in case (1) or $FFFF_{16}$ in case (2), then type 0 interrupt occurs	OF, SF, ZF, AF, PF, CF undefined
IMUL	Integer multiply (signed)	IMUL S	(AL) · (S8) → (AX) (AX) · (S16) → (DX),(AX)	OF, CF SF, ZF, AF, PF undefined
IDIV	Integer divide (signed)	IDIV S	(1) Q((AX)/(S8)) → (AL) R((AX)/(S8)) → (AH) (2) Q((DX,AX)/(S16)) → (AX) R((DX,AX)/(S16)) → (DX) If Q is positive and exceeds $7FFF_{16}$ or if Q is negative and becomes less than 8001_{16}, then type 0 interupt occurs	OF, SF, ZF, AF, PF, CF undefined

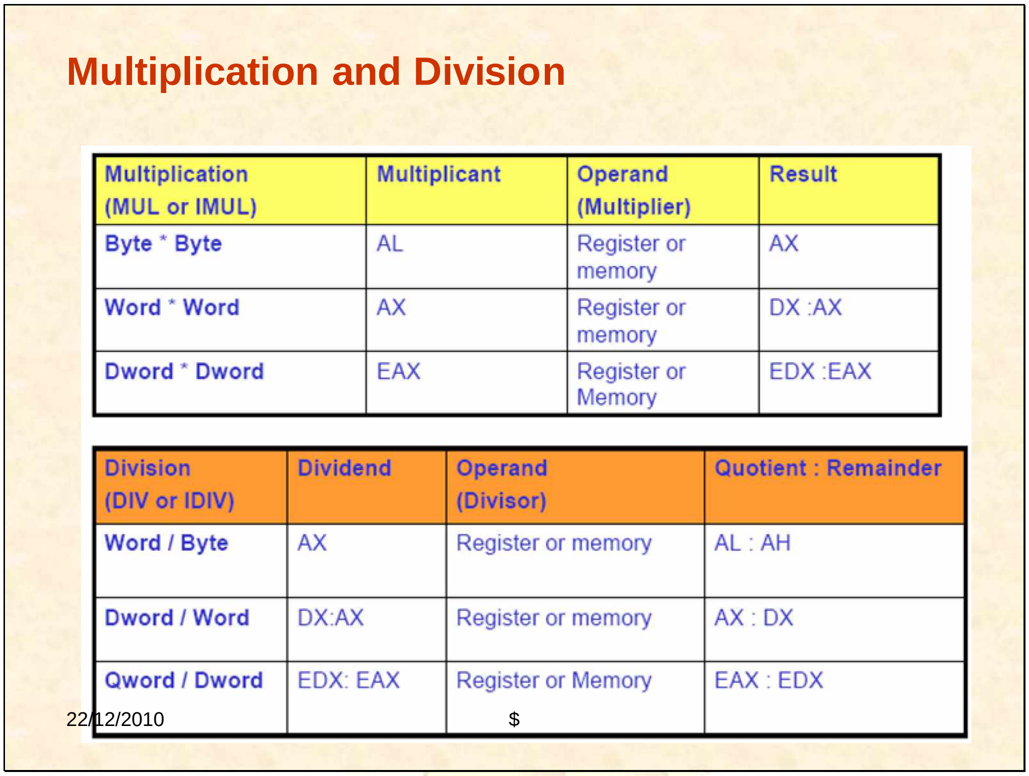
AAM	Adjust AL for multiplication	AAM	Q((AL)/10) → (AH)
			R((AL)/10) → (AL)
				SF, ZF, PF
				OF, AF,CF undefined

AAD	Adjust AX for division	AAD	(AH) · 10 + (AL) → (AL) 00 → (AH)	SF, ZF, PF OF, AF, CF undefined
CBW	Convert byte to word	CBW	(MSB of AL) → (All bits of AH)	None
CWD	Convert word to double word	CWD	(MSB of AX) → (All bits of DX)	None

(a)

Source
Reg8 Reg16 Mem8 Mem16

(b)

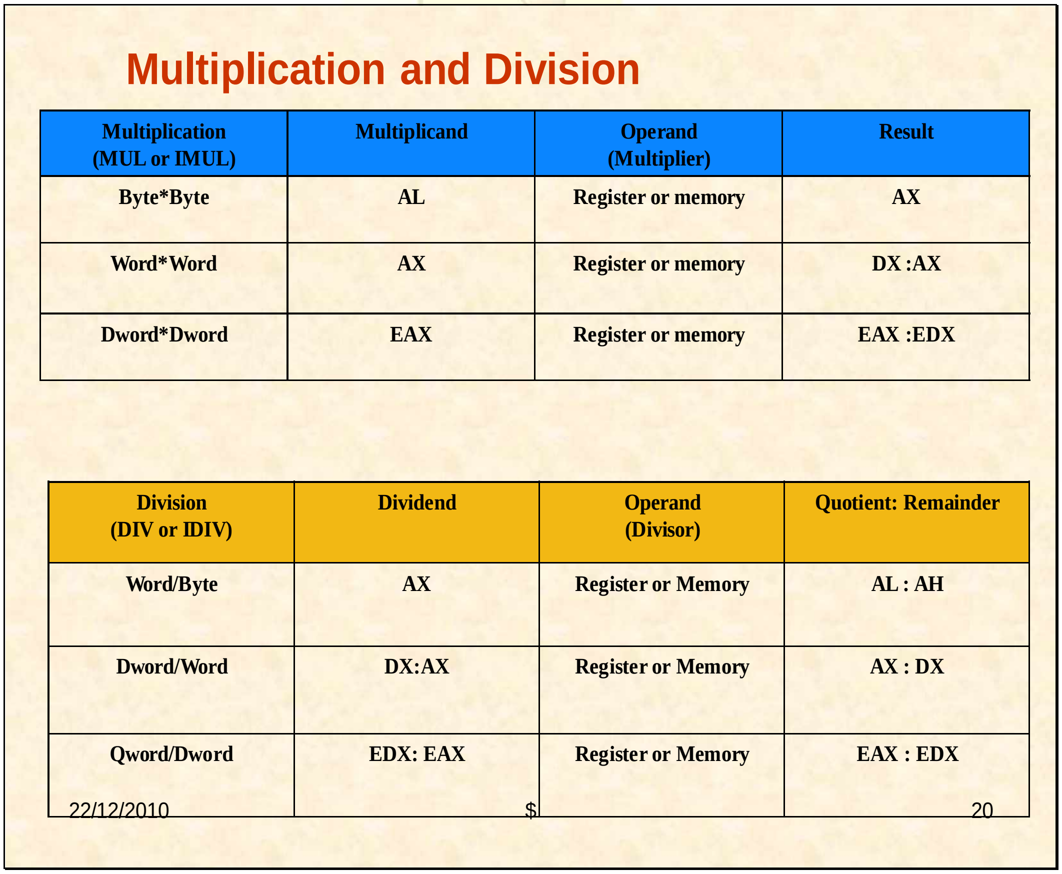


Multiplication and Division

Multiplication (MUL or IMUL)	Multiplicant	Operand (Multiplier)	Result
Byte * Byte	AL	Register or memory	AX
Word * Word	AX	Register or memory	DX :AX
Dword * Dword	EAX	Register or Memory	EDX :EAX

Division (DIV or IDIV)	Dividend	Operand (Divisor)	Quotient : Remainder
Word / Byte	AX	Register or memory	AL : AH
Dword / Word	DX:AX	Register or memory	AX : DX
Qword / Dword	EDX: EAX	Register or Memory	EAX : EDX

22/12/2010 $

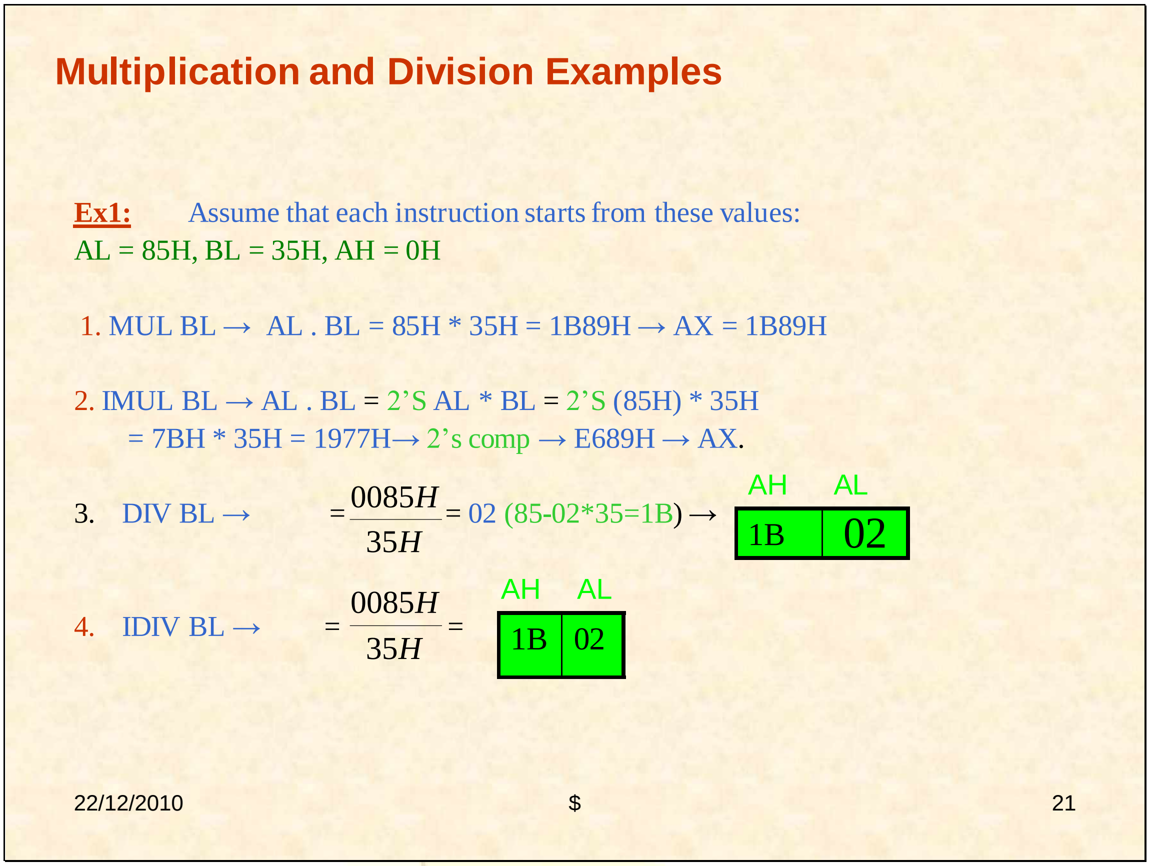


Multiplication and Division

Multiplication (MUL or IMUL)	Multiplicand	Operand (Multiplier)	Result
Byte*Byte	AL	Register or memory	AX
Word*Word	AX	Register or memory	DX :AX
Dword*Dword	EAX	Register or memory	EAX :EDX

Division (DIV or IDIV)	Dividend	Operand (Divisor)	Quotient: Remainder
Word/Byte	AX	Register or Memory	AL : AH
Dword/Word	DX:AX	Register or Memory	AX : DX
Qword/Dword	EDX: EAX	Register or Memory	EAX : EDX

22/12/2010 $ 20



Multiplication and Division Examples

Ex1: Assume that each instruction starts from these values:
AL = 85H, BL = 35H, AH = 0H

1. MUL BL → AL . BL = 85H * 35H = 1B89H → AX = 1B89H

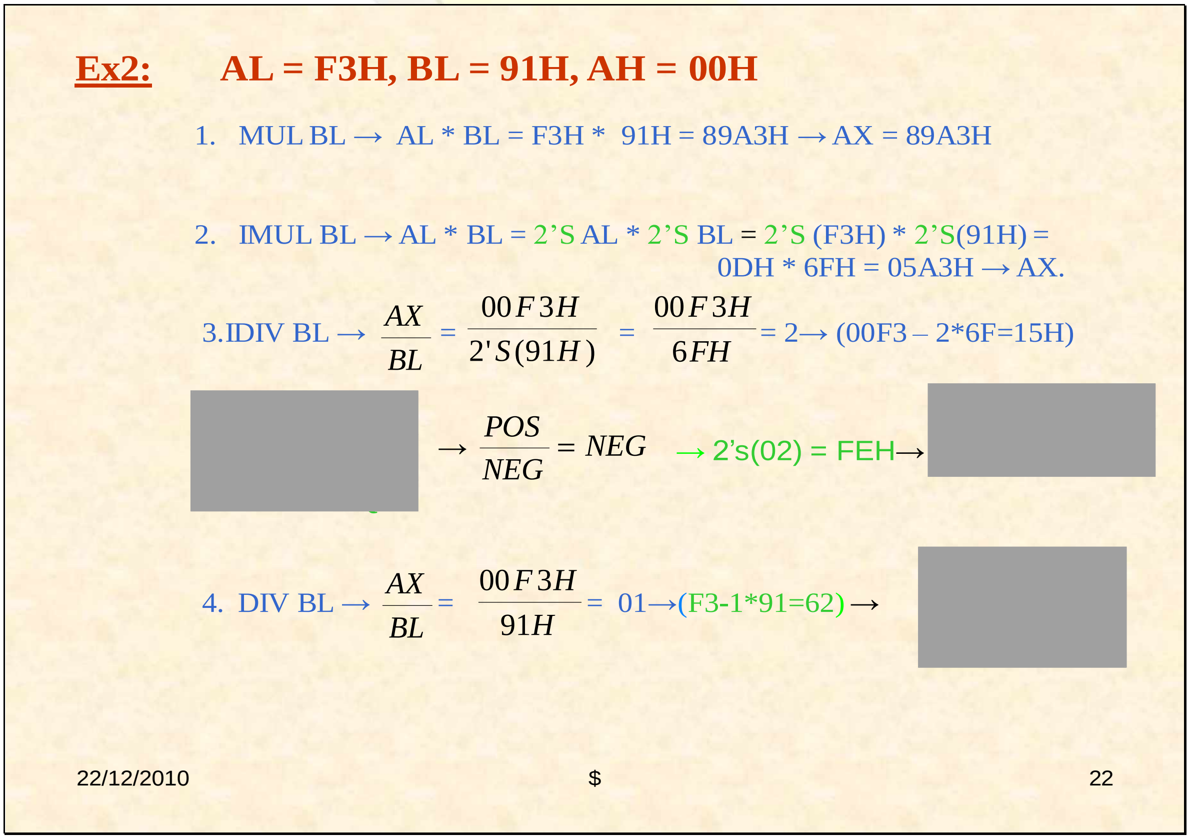
2. IMUL BL → AL . BL = 2'S AL * BL = 2'S (85H) * 35H
= 7BH * 35H = 1977H→ 2's comp → E689H → AX.

3. DIV BL → $=\frac{0085H}{35H}=$ 02 (85-02*35=1B) →

AH	AL
1B	02

4. IDIV BL → $=\frac{0085H}{35H}=$

AH	AL
1B	02



Ex2: AL = F3H, BL = 91H, AH = 00H

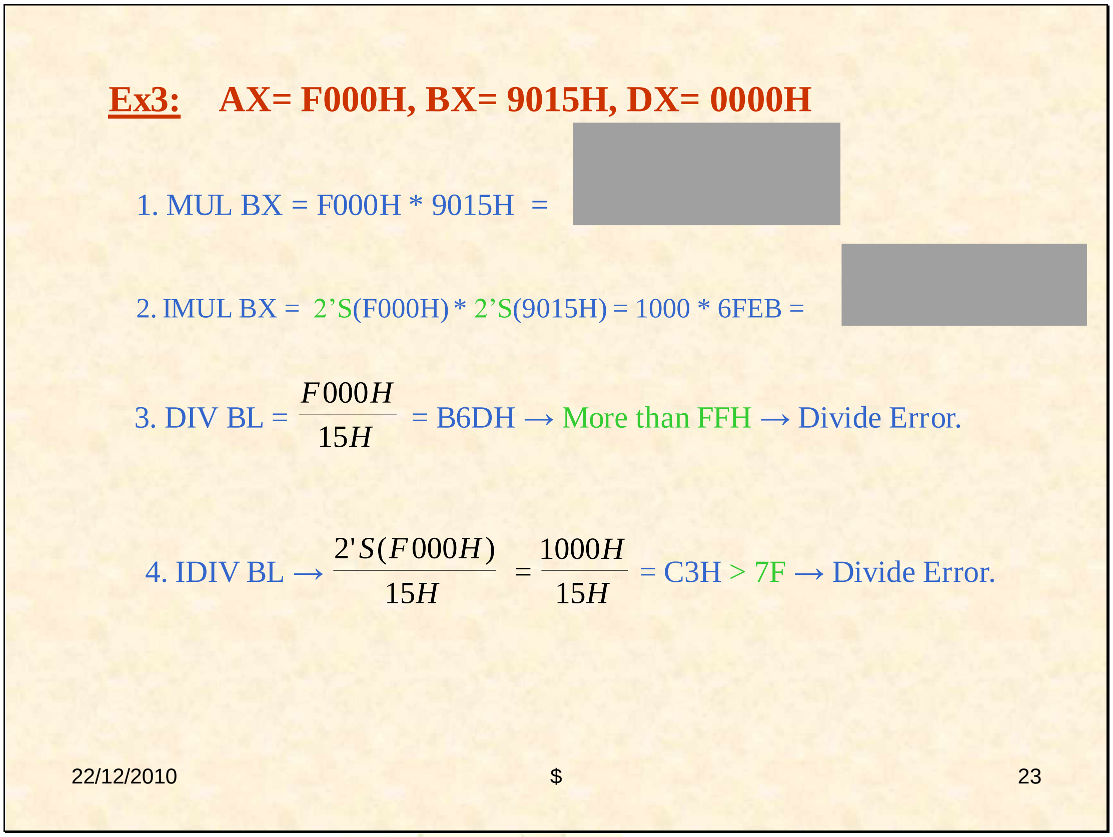
1. MUL BL → AL * BL = F3H * 91H = 89A3H → AX = 89A3H

2. IMUL BL → AL * BL = 2'S AL * 2'S BL = 2'S (F3H) * 2'S(91H) =
0DH * 6FH = 05A3H → AX.

3.IDIV BL → $\frac{AX}{BL}=\frac{00F3H}{2'S(91H)}=\frac{00F3H}{6FH}$ = 2→ (00F3 – 2*6F=15H)

→ $\frac{POS}{NEG}=NEG$ → 2's(02) = FEH→

4. DIV BL → $\frac{AX}{BL}=\frac{00F3H}{91H}$ = 01→(F3-1*91=62) →



Ex3: **AX= F000H, BX= 9015H, DX= 0000H**

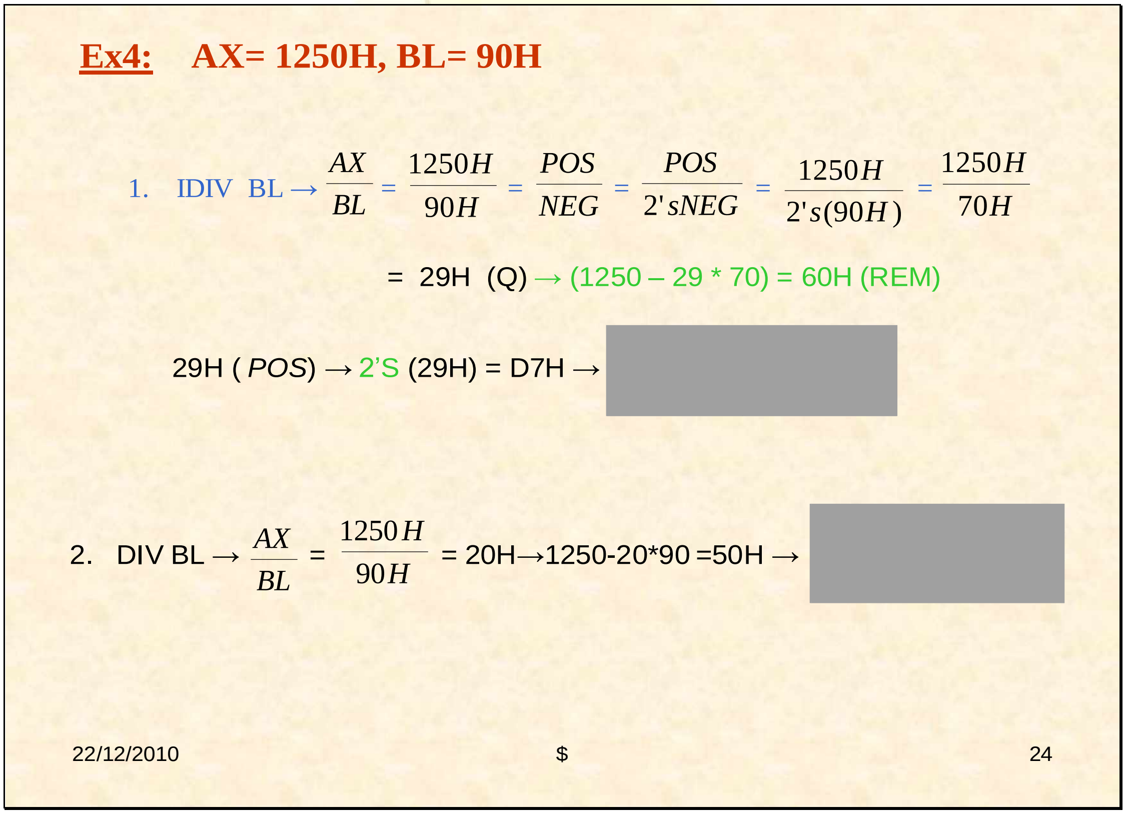
1. MUL BX = F000H * 9015H = [redacted]

2. IMUL BX = 2’S(F000H) * 2’S(9015H) = 1000 * 6FEB = [redacted]

3. DIV BL = $\frac{F000H}{15H}$ = B6DH → More than FFH → Divide Error.

4. IDIV BL → $\frac{2'S(F000H)}{15H} = \frac{1000H}{15H}$ = C3H > 7F → Divide Error.

22/12/2010 $ 23



Ex4: **AX= 1250H, BL= 90H**

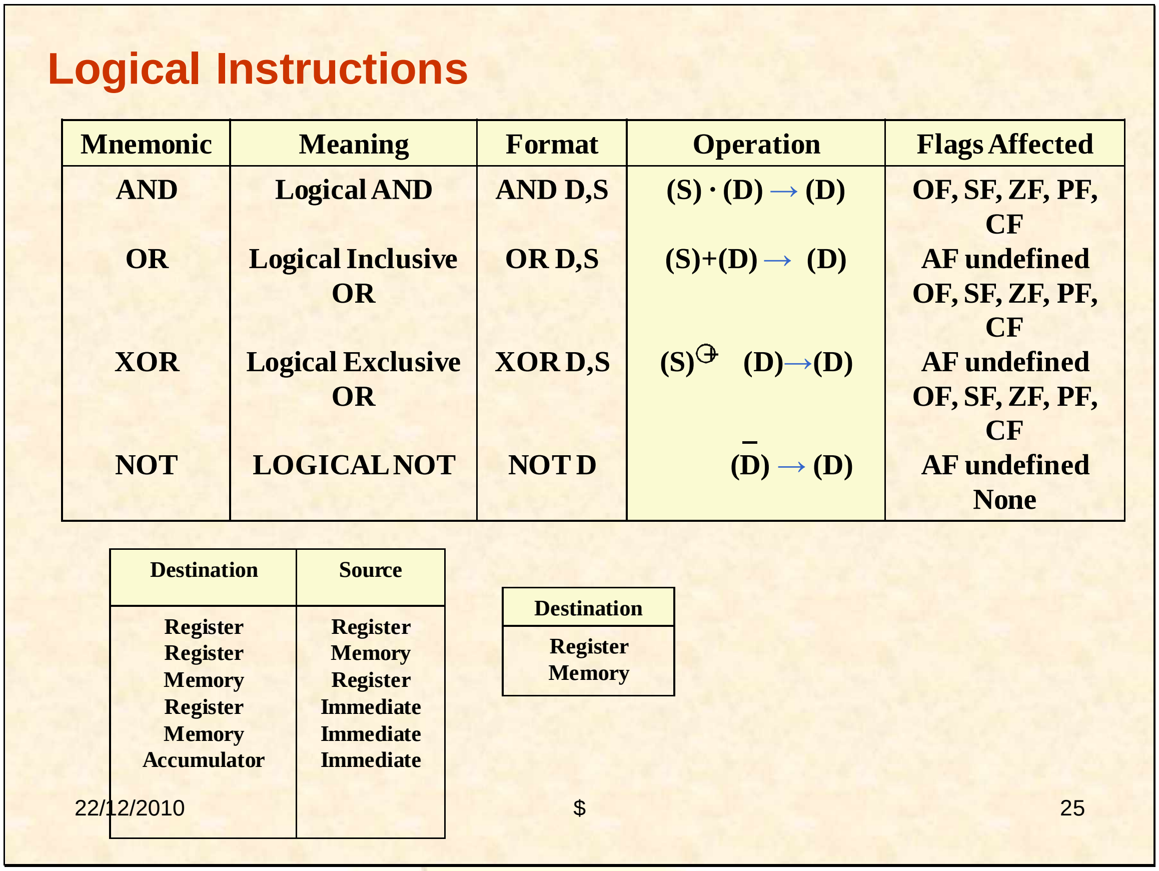
1. IDIV BL → $\frac{AX}{BL} = \frac{1250H}{90H} = \frac{POS}{NEG} = \frac{POS}{2'sNEG} = \frac{1250H}{2's(90H)} = \frac{1250H}{70H}$

= 29H (Q) → (1250 – 29 * 70) = 60H (REM)

29H (*POS*) → 2’S (29H) = D7H → [redacted]

2. DIV BL → $\frac{AX}{BL} = \frac{1250H}{90H}$ = 20H→1250-20*90 =50H → [redacted]

22/12/2010 $ 24



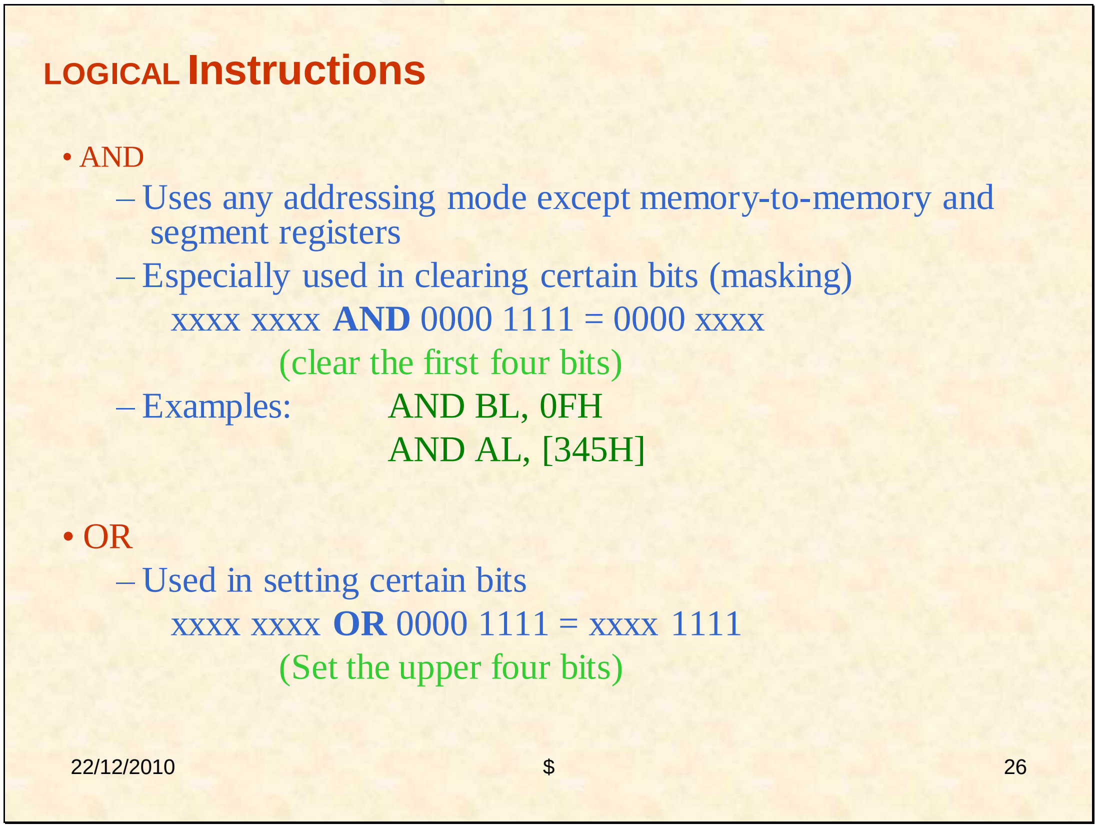
Logical Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
AND	Logical AND	AND D,S	(S) · (D) → (D)	OF, SF, ZF, PF, CF AF undefined
OR	Logical Inclusive OR	OR D,S	(S)+(D) → (D)	OF, SF, ZF, PF, CF AF undefined
XOR	Logical Exclusive OR	XOR D,S	(S)⊕ (D)→(D)	OF, SF, ZF, PF, CF AF undefined
NOT	LOGICAL NOT	NOT D	$\overline{(D)}$ → (D)	None

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

Destination
Register
Memory

22/12/2010 $ 25



LOGICAL Instructions

- AND
  - Uses any addressing mode except memory-to-memory and segment registers
  - Especially used in clearing certain bits (masking)

    xxxx xxxx **AND** 0000 1111 = 0000 xxxx

    (clear the first four bits)
  - Examples: AND BL, 0FH

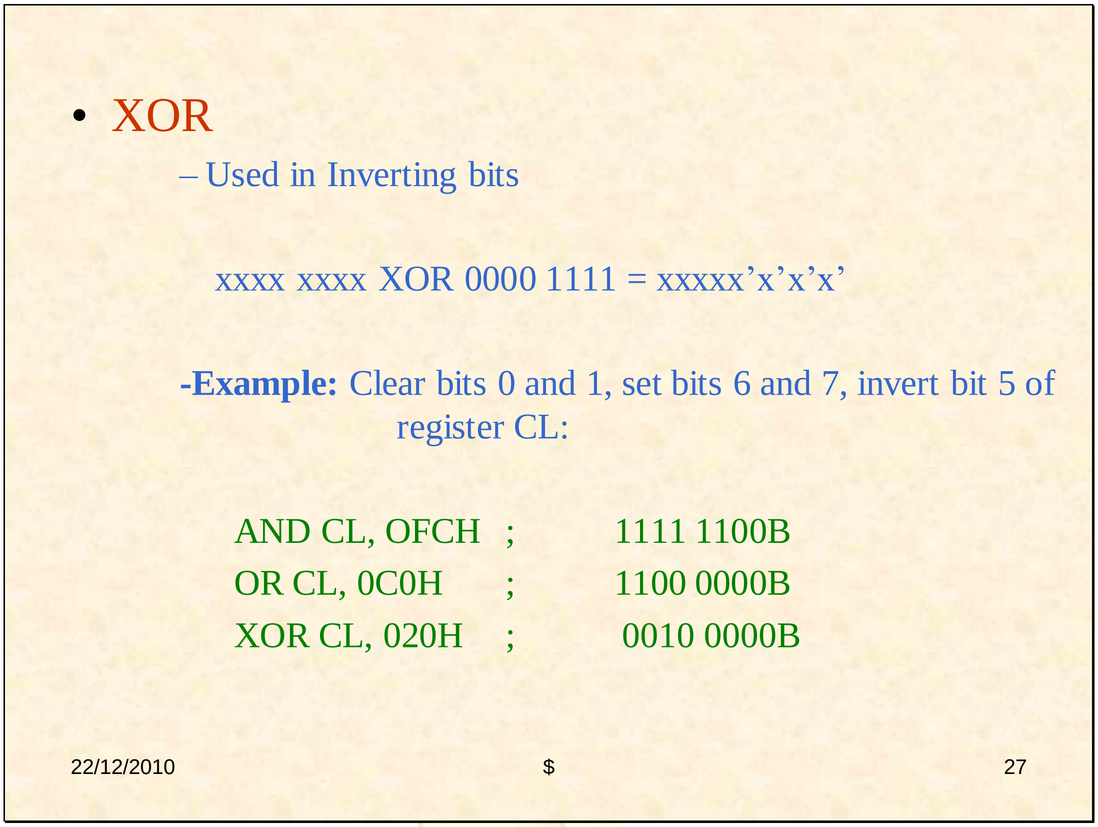
    AND AL, [345H]

- OR
  - Used in setting certain bits

    xxxx xxxx **OR** 0000 1111 = xxxx 1111

    (Set the upper four bits)

22/12/2010 $ 26



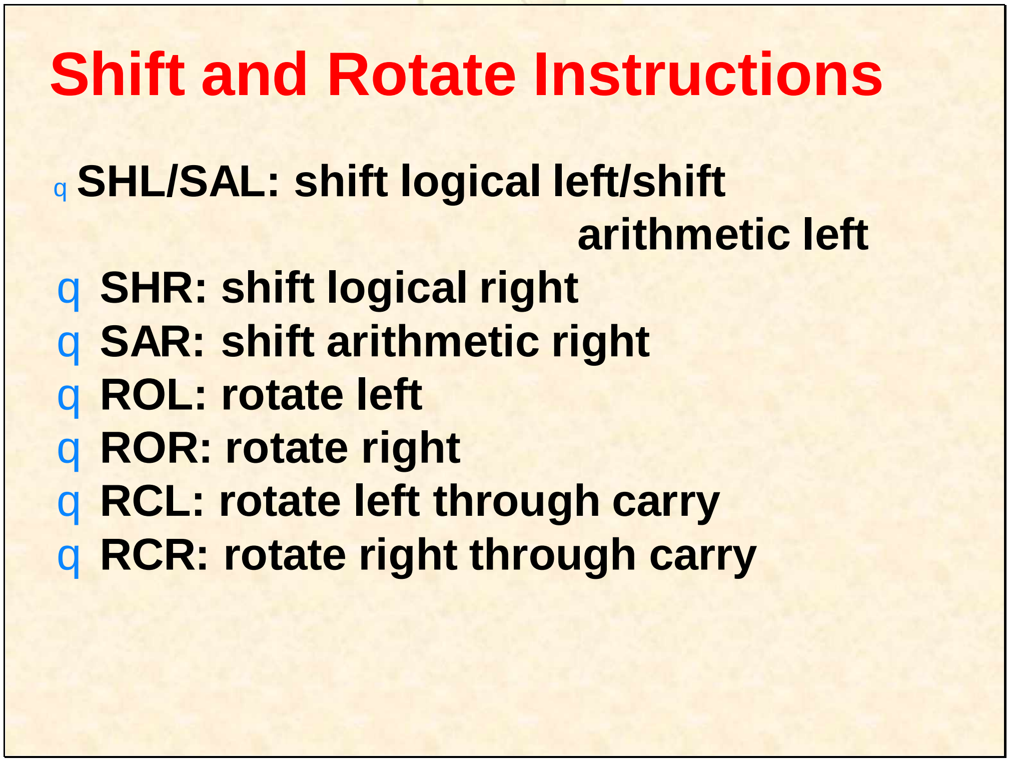
- XOR
  – Used in Inverting bits

xxxx xxxx XOR 0000 1111 = xxxxx’x’x’x’

-Example: Clear bits 0 and 1, set bits 6 and 7, invert bit 5 of register CL:

```
AND CL, OFCH  ;        1111 1100B
OR CL, 0C0H   ;        1100 0000B
XOR CL, 020H  ;         0010 0000B
```

22/12/2010 $ 27



Shift and Rotate Instructions

q **SHL/SAL: shift logical left/shift arithmetic left**

q **SHR: shift logical right**

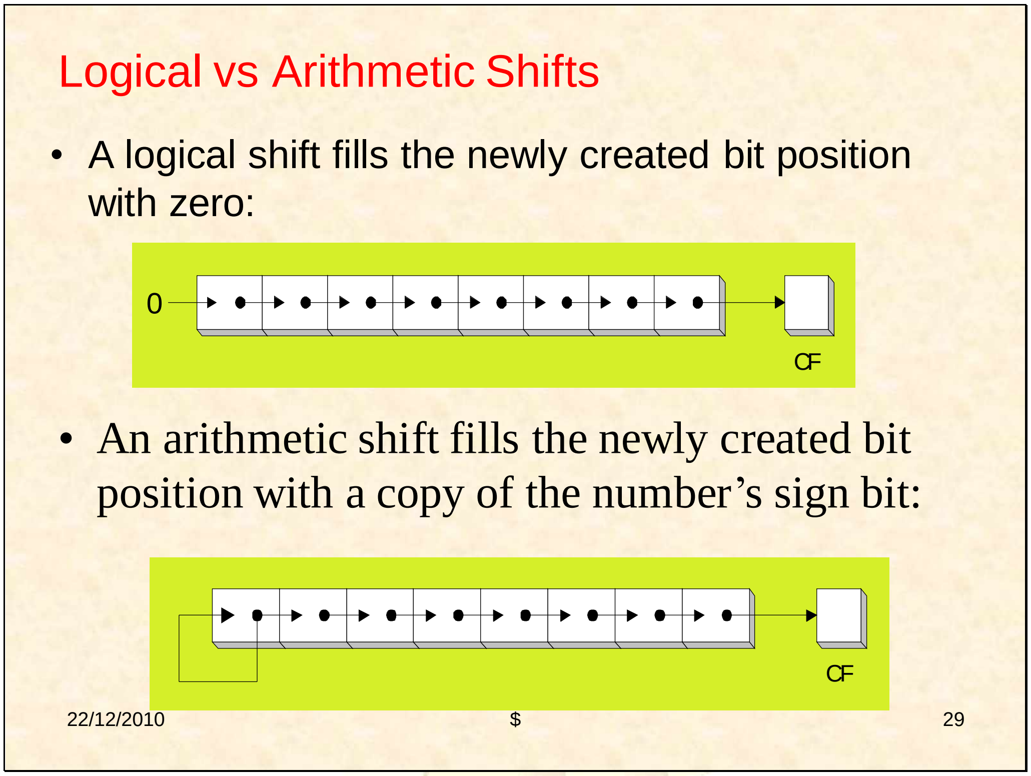
q **SAR: shift arithmetic right**

q **ROL: rotate left**

q **ROR: rotate right**

q **RCL: rotate left through carry**

q **RCR: rotate right through carry**

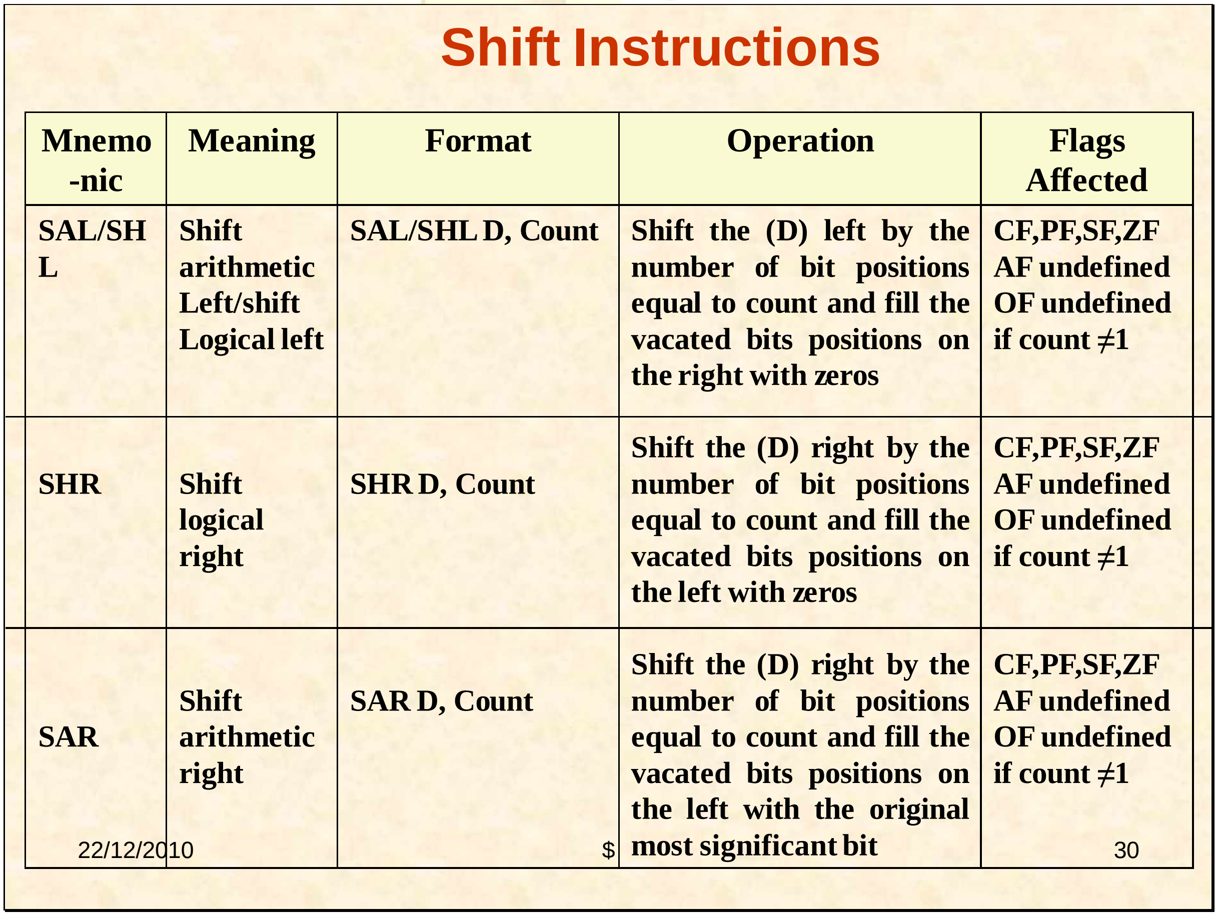


Logical vs Arithmetic Shifts

- A logical shift fills the newly created bit position with zero:

- An arithmetic shift fills the newly created bit position with a copy of the number’s sign bit:

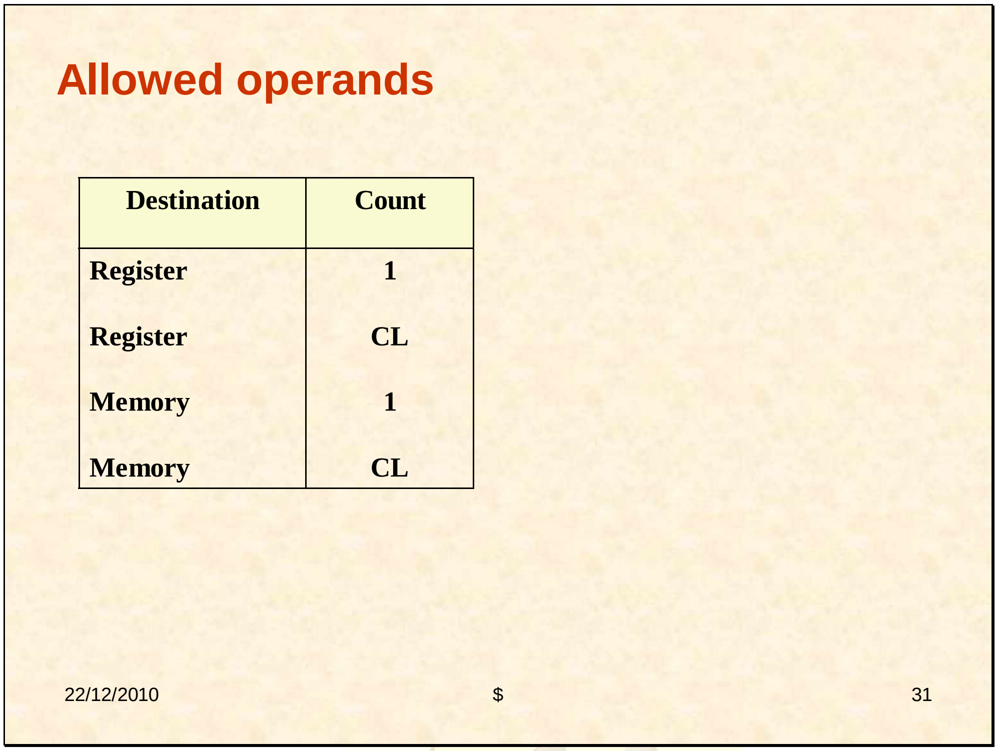
22/12/2010 $ 29



Shift Instructions

Mnemo -nic	Meaning	Format	Operation	Flags Affected
SAL/SHL	Shift arithmetic Left/shift Logical left	SAL/SHL D, Count	Shift the (D) left by the number of bit positions equal to count and fill the vacated bits positions on the right with zeros	CF,PF,SF,ZF AF undefined OF undefined if count ≠1
SHR	Shift logical right	SHR D, Count	Shift the (D) right by the number of bit positions equal to count and fill the vacated bits positions on the left with zeros	CF,PF,SF,ZF AF undefined OF undefined if count ≠1
SAR	Shift arithmetic right	SAR D, Count	Shift the (D) right by the number of bit positions equal to count and fill the vacated bits positions on the left with the original most significant bit	CF,PF,SF,ZF AF undefined OF undefined if count ≠1

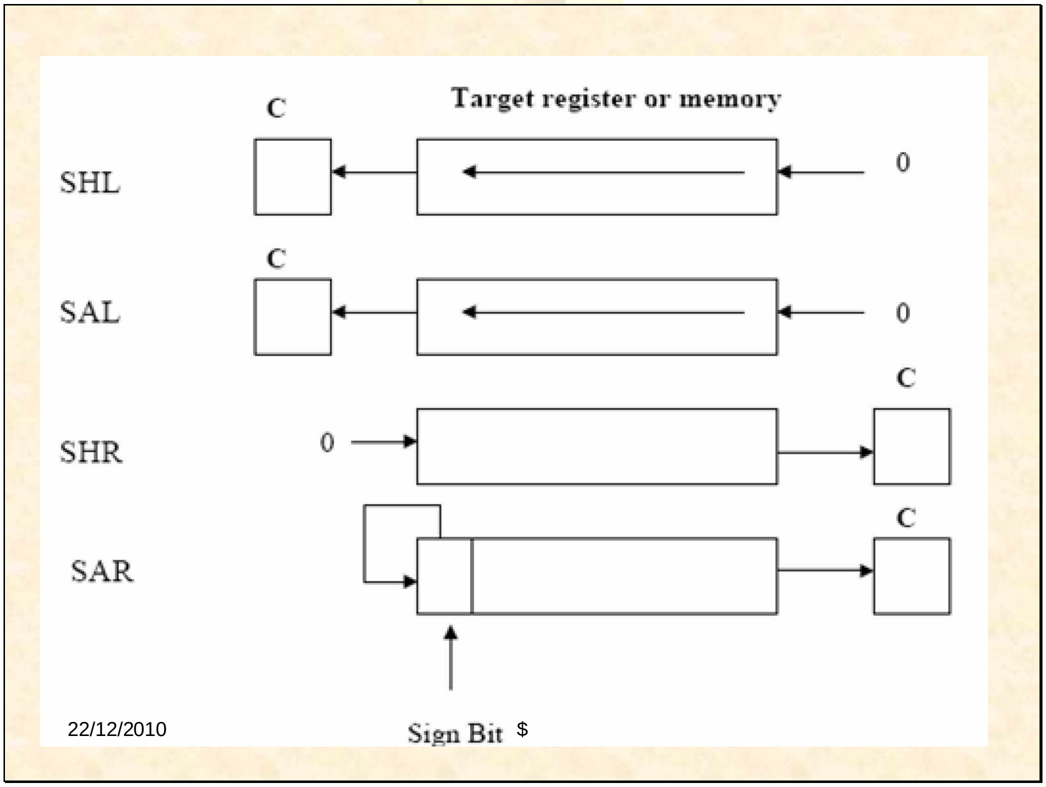
22/12/2010 $ 30



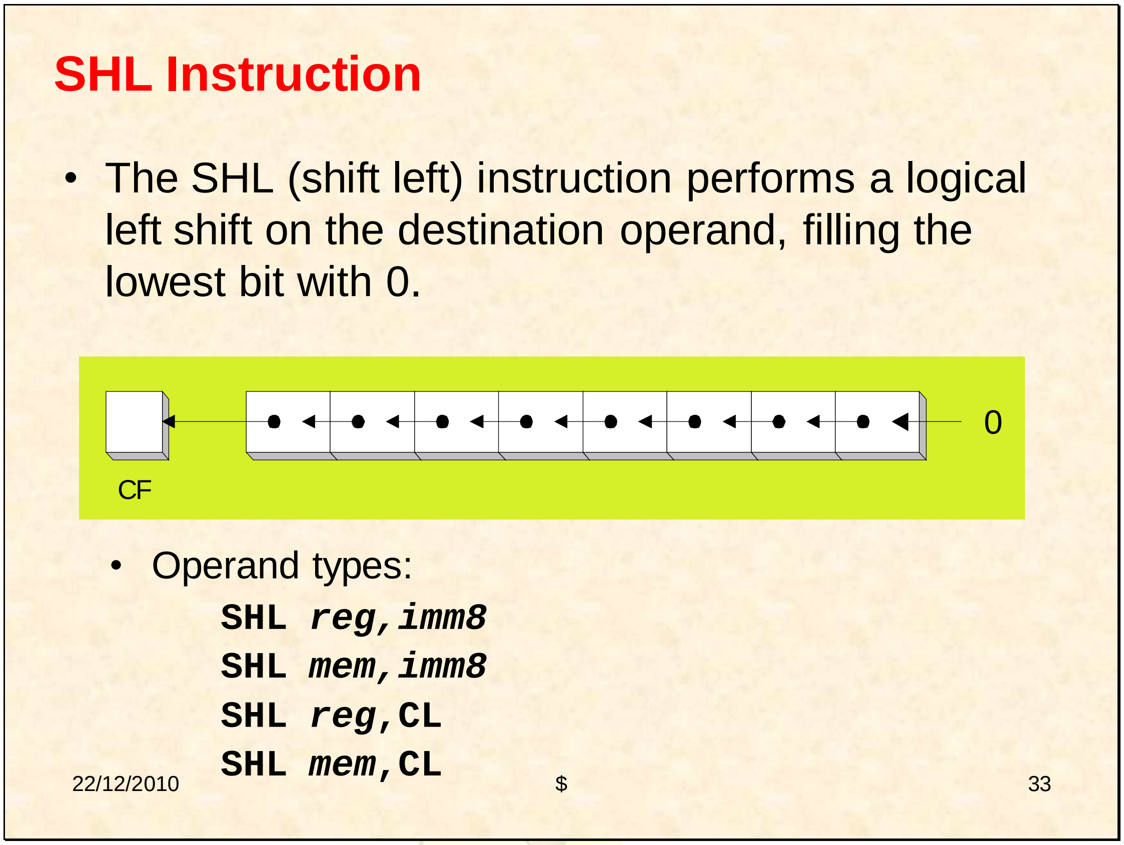
Allowed operands

Destination	Count
Register	1
Register	CL
Memory	1
Memory	CL

22/12/2010 $ 31



22/12/2010 $



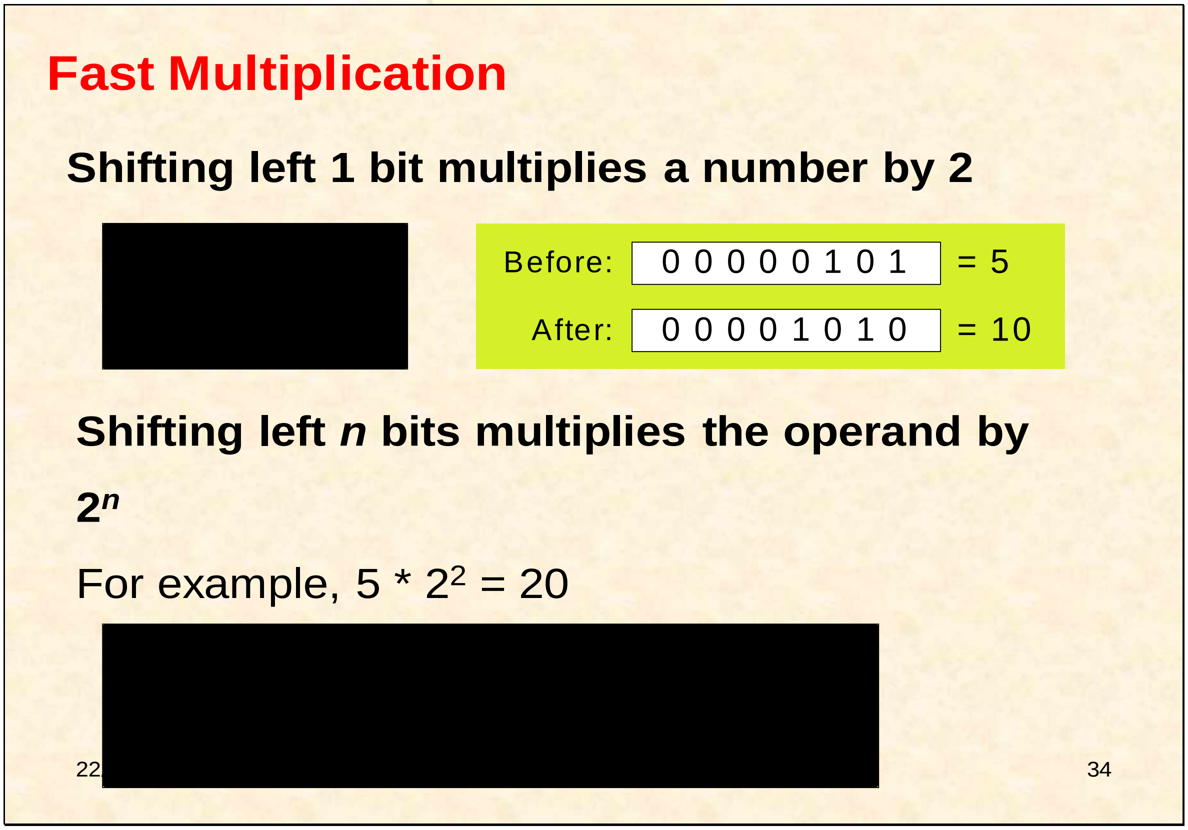
SHL Instruction

- The SHL (shift left) instruction performs a logical left shift on the destination operand, filling the lowest bit with 0.

- Operand types:

```
SHL reg,imm8
SHL mem,imm8
SHL reg,CL
SHL mem,CL
```

22/12/2010 $ 33



Fast Multiplication

Shifting left 1 bit multiplies a number by 2

[redacted]

Before: 0 0 0 0 0 1 0 1 = 5

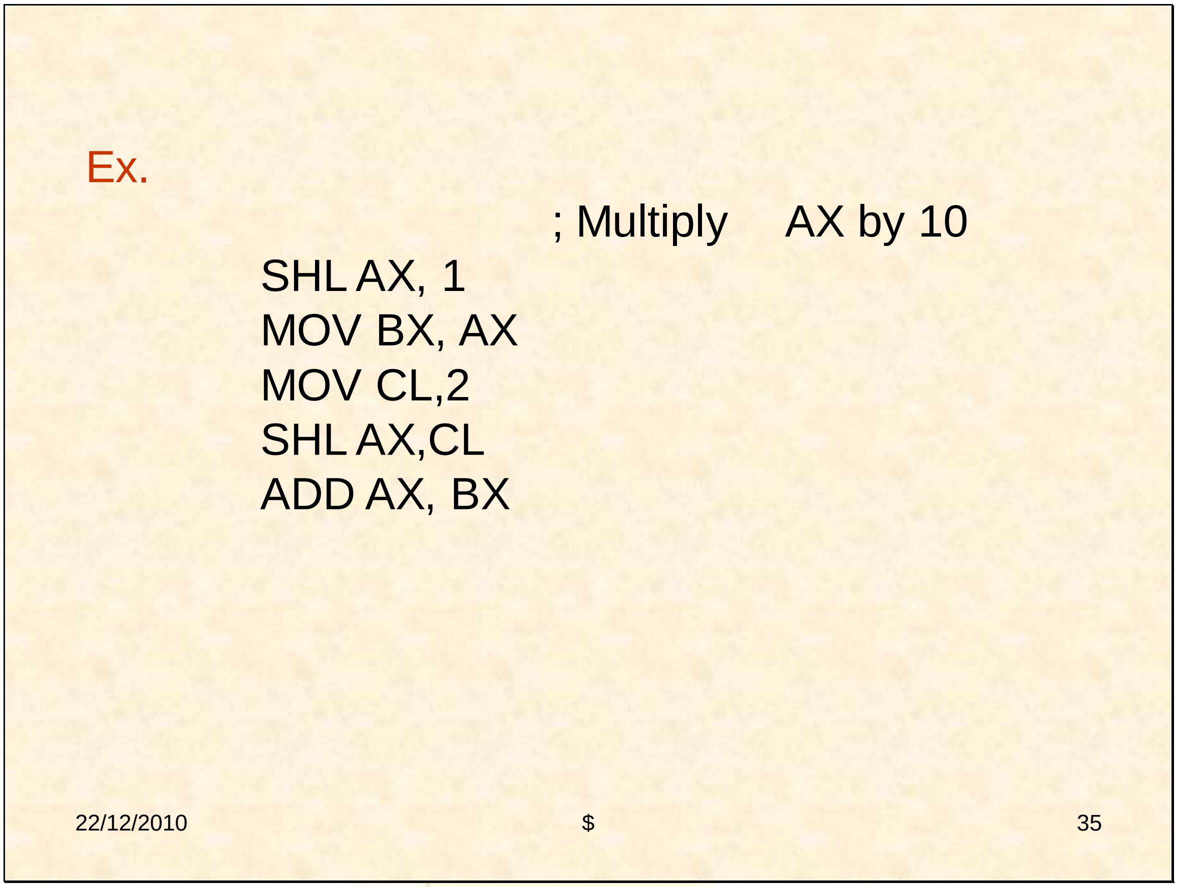
After: 0 0 0 0 1 0 1 0 = 10

Shifting left *n* bits multiplies the operand by 2^n

For example, $5 * 2^2 = 20$

[redacted]

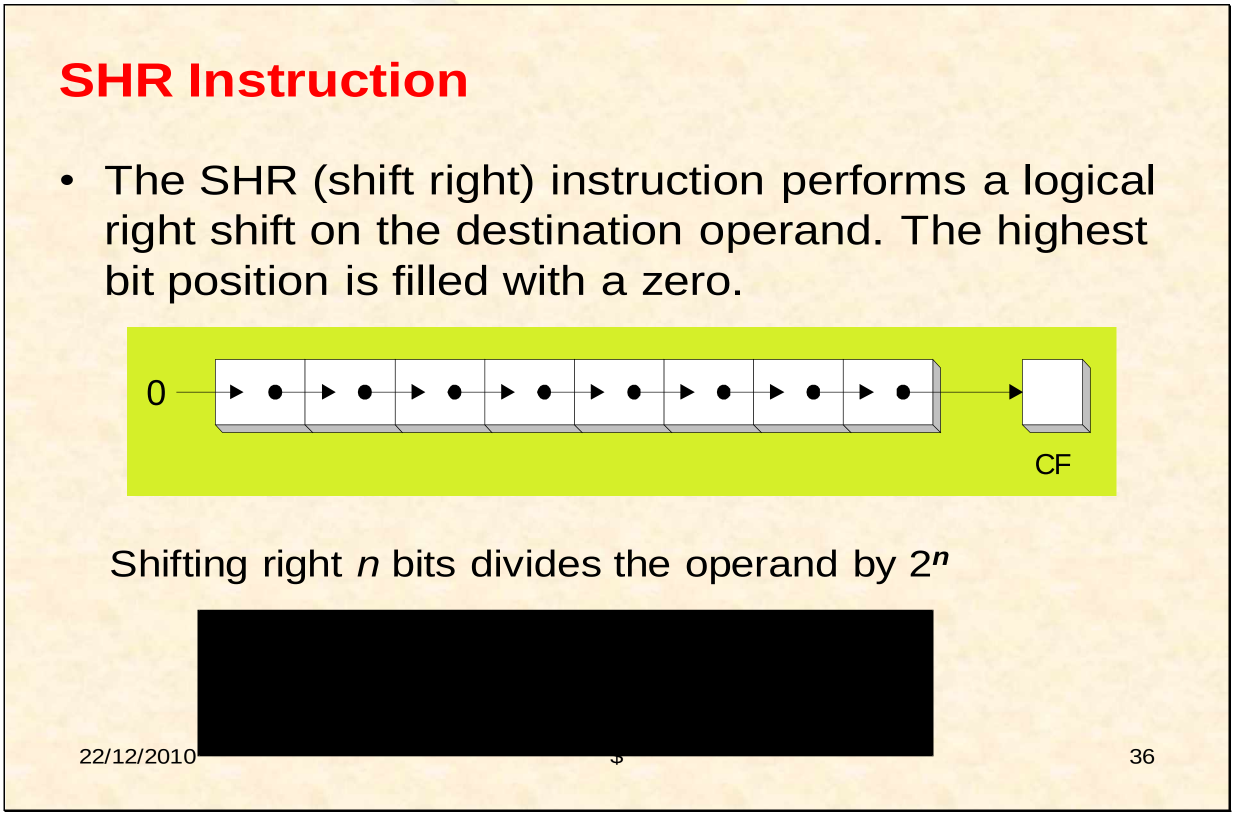
22 34



Ex.

```
                        ; Multiply    AX by 10
        SHL AX, 1
        MOV BX, AX
        MOV CL,2
        SHL AX,CL
        ADD AX, BX
```

22/12/2010 $ 35



SHR Instruction

- The SHR (shift right) instruction performs a logical right shift on the destination operand. The highest bit position is filled with a zero.

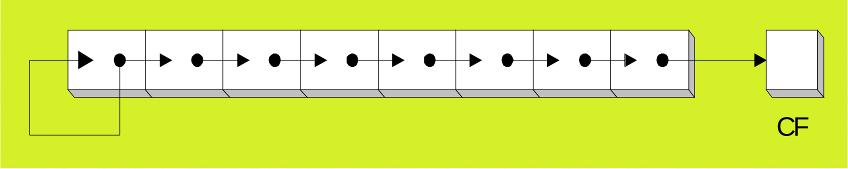
Shifting right *n* bits divides the operand by 2^{n}

[redacted]

22/12/2010 $ 36

SAR Instruction

- **SAR (shift arithmetic right) performs a right arithmetic shift on the destination operand.**



An arithmetic shift preserves the number's sign.

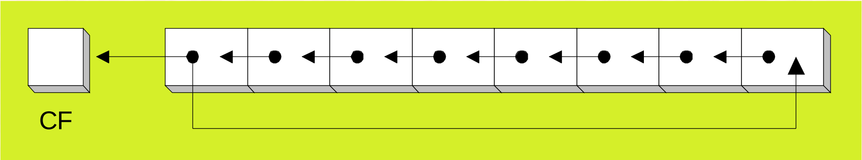
[redacted]

Rotate Instructions

Mnem -onic	Meaning	Format	Operation	Flags Affected
ROL	Rotate Left	ROL D,Count	Rotate the (D) left by the number of bit positions equal to Count. Each bit shifted out from the left most bit goes back into the rightmost bit position.	CF OF undefined if count ≠ 1
ROR	Rotate Right	ROR D,Count	Rotate the (D) right by the number of bit positions equal to Count. Each bit shifted out from the rightmost bit goes back into the leftmost bit position.	CF OF undefined if count ≠ 1
RCL	Rotate Left through Carry	RCL D,Count	Same as ROL except carry is attached to (D) for rotation.	CF OF undefined if count ≠ 1
RCR	Rotate right through Carry	RCR D,Count	Same as ROR except carry is attached to (D) for rotation.	CF OF undefined if count ≠ 1

ROL Instruction

- ROL (rotate) shifts each bit to the left
- The highest bit is copied into both the Carry flag and into the lowest bit
- No bits are lost

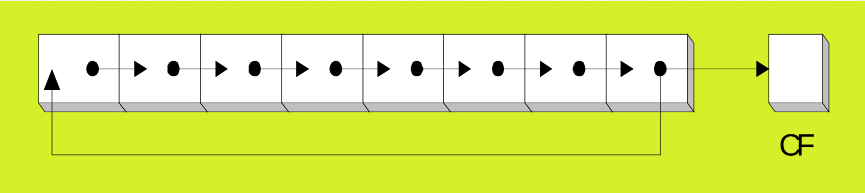


[redacted]

22/12/ 39

ROR Instruction

- ROR (rotate right) shifts each bit to the right
- The lowest bit is copied into both the Carry flag and into the highest bit
- No bits are lost

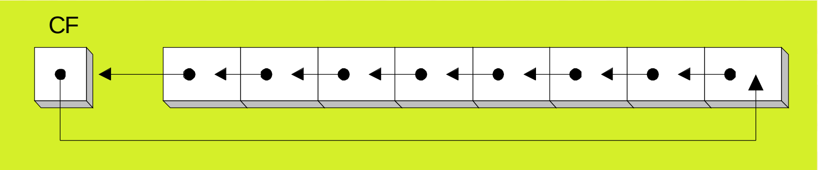


[redacted]

22/12/2010 $ 40

RCL Instruction

- RCL (rotate carry left) shifts each bit to the left
- Copies the Carry flag to the least significant bit
- Copies the most significant bit to the Carry flag

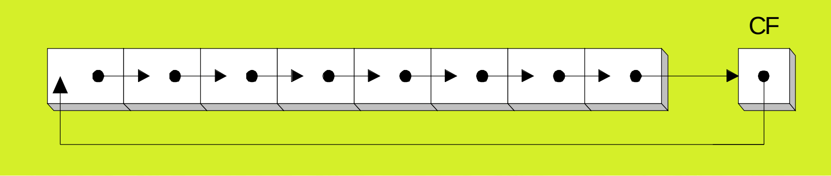


[redacted]

22/12/2010 $ 41

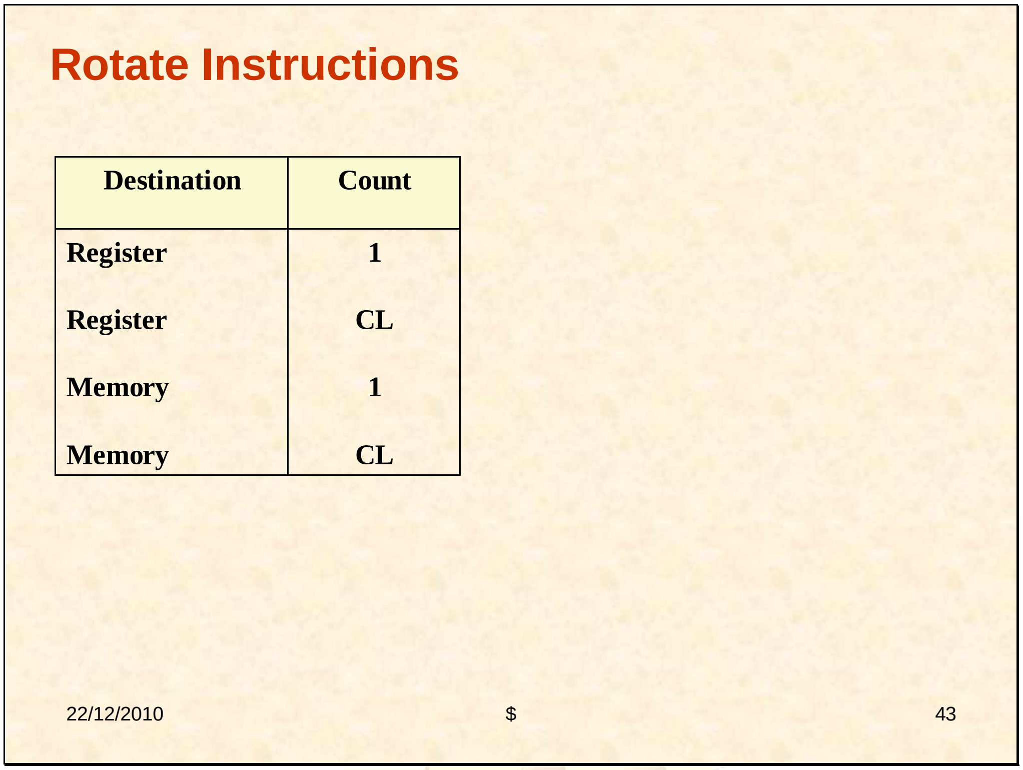
RCR Instruction

- RCR (rotate carry right) shifts each bit to the right
- Copies the Carry flag to the most significant bit
- Copies the least significant bit to the Carry flag



[redacted]

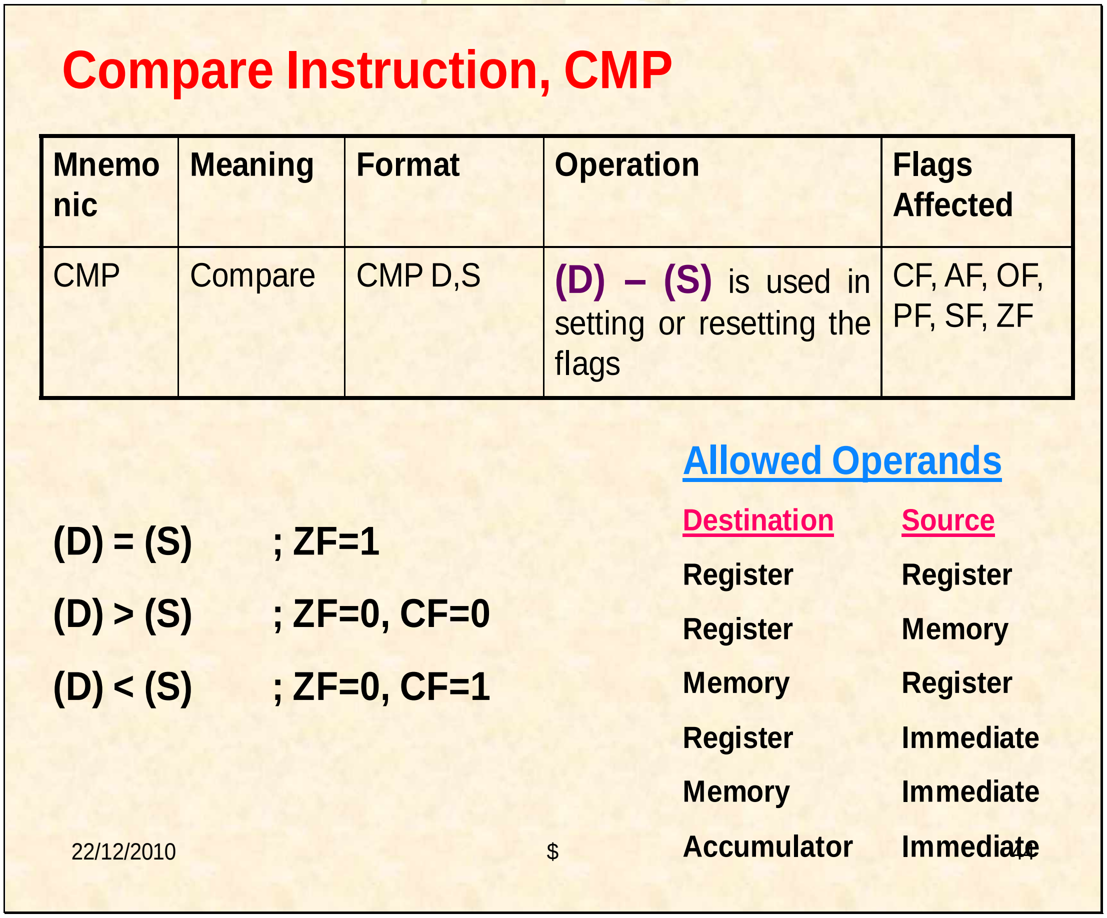
22/12/2010 $ 42



Rotate Instructions

Destination	Count
Register	1
Register	CL
Memory	1
Memory	CL

22/12/2010 $ 43



Compare Instruction, CMP

Mnemonic	Meaning	Format	Operation	Flags Affected
CMP	Compare	CMP D,S	(D) – (S) is used in setting or resetting the flags	CF, AF, OF, PF, SF, ZF

(D) = (S) ; ZF=1

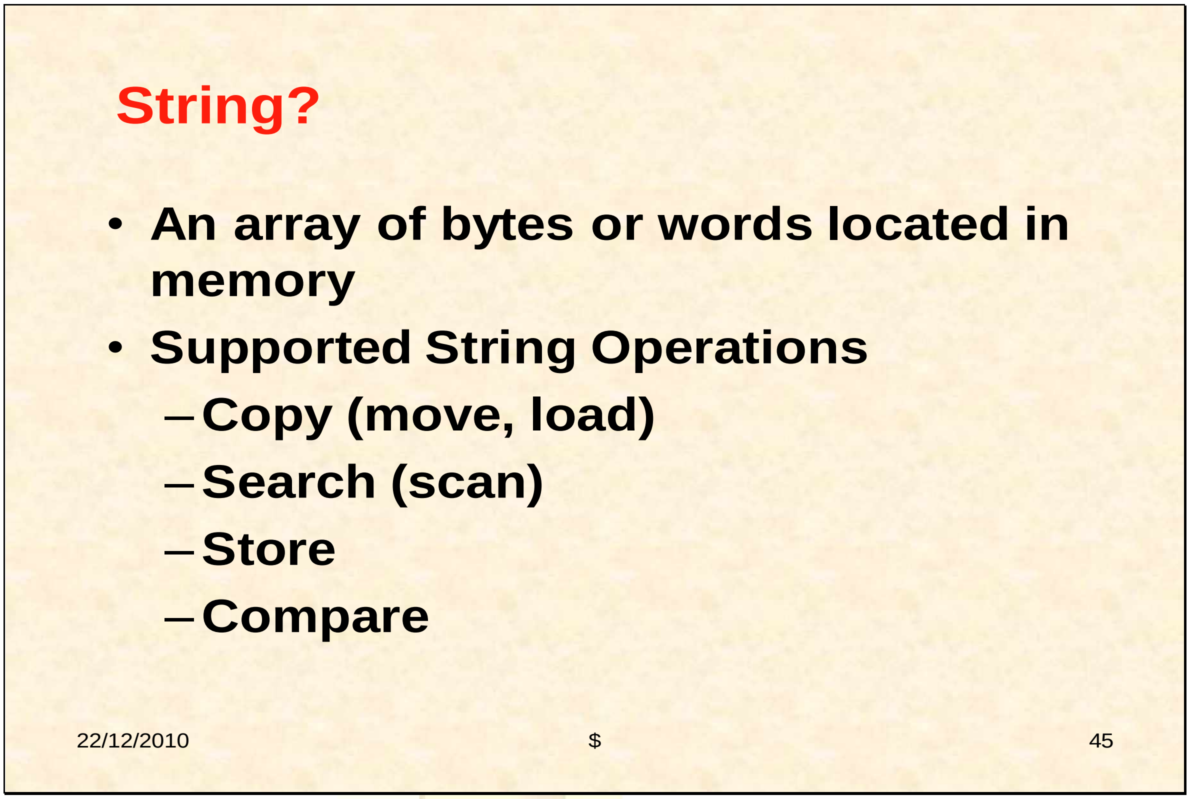
(D) > (S) ; ZF=0, CF=0

(D) < (S) ; ZF=0, CF=1

Allowed Operands

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

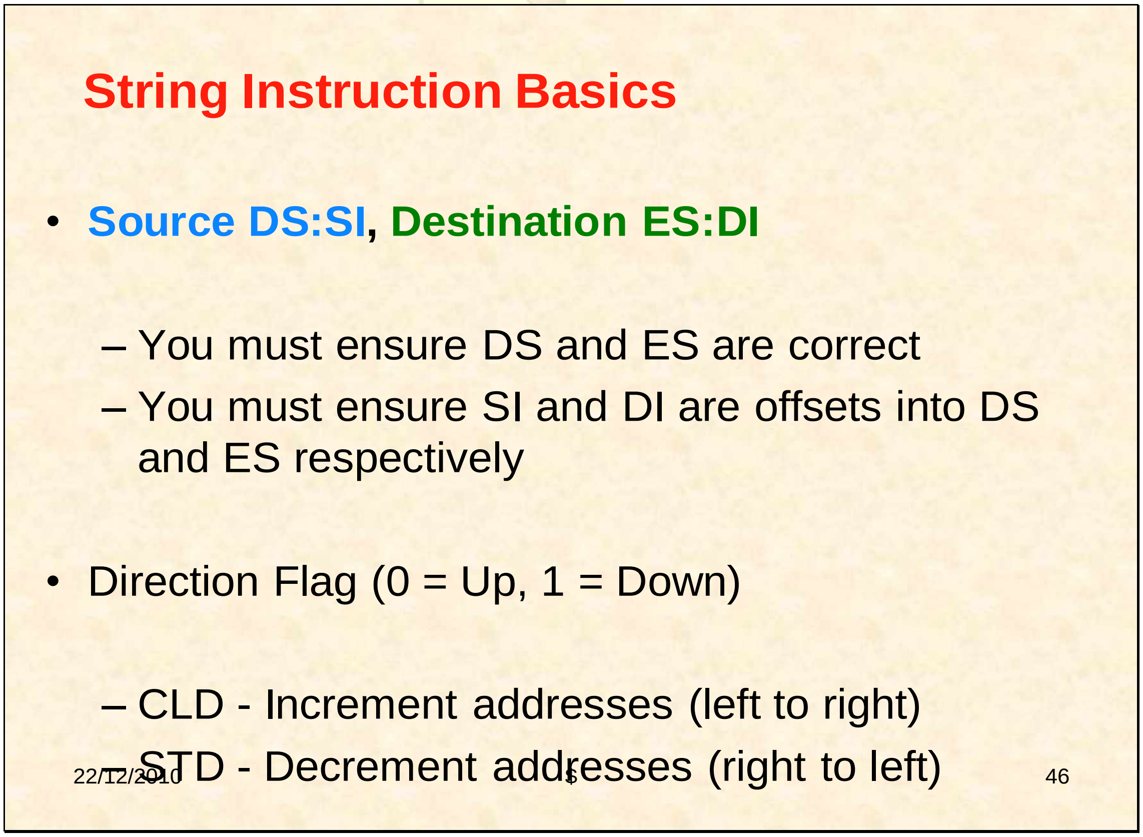
22/12/2010 $ 44



String?

- An array of bytes or words located in memory
- Supported String Operations
  - Copy (move, load)
  - Search (scan)
  - Store
  - Compare

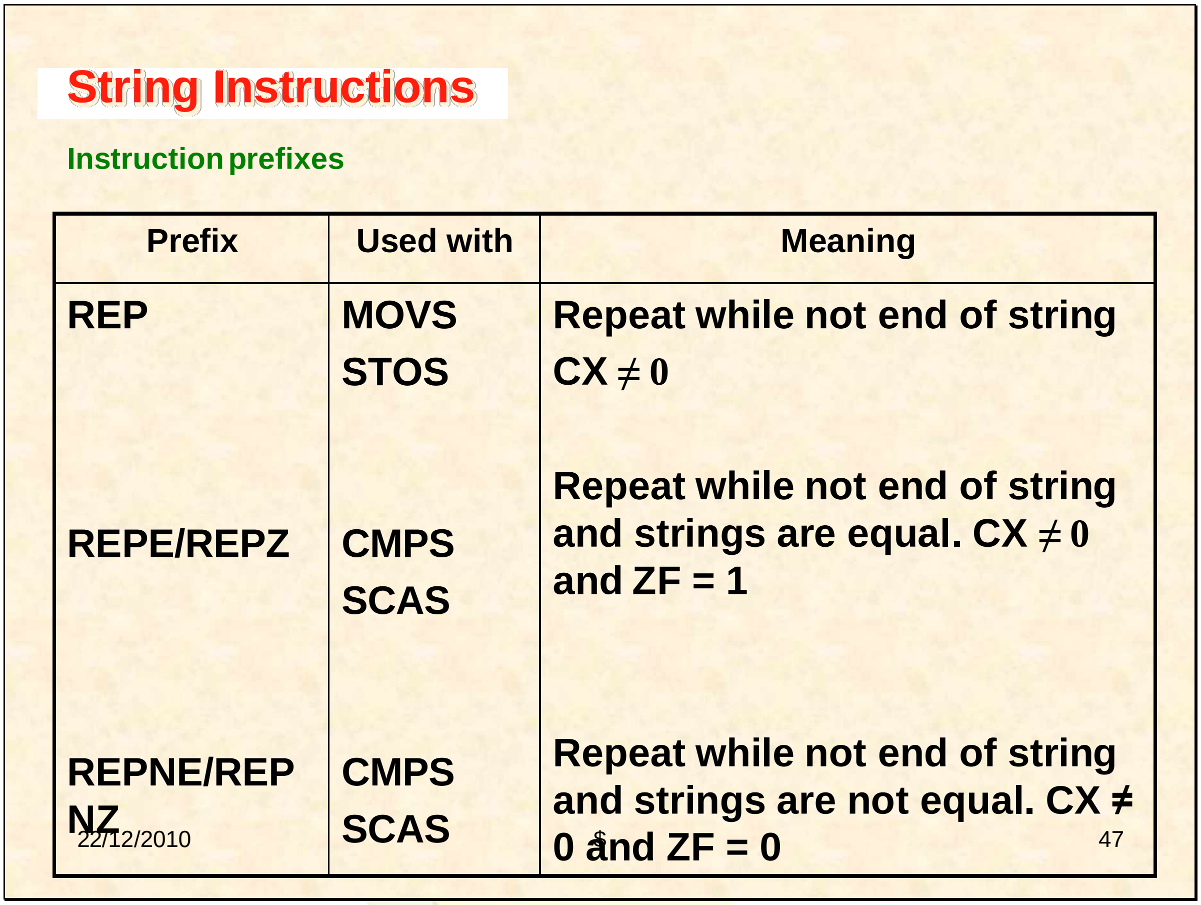
22/12/2010 $ 45



String Instruction Basics

- Source DS:SI, Destination ES:DI
  - You must ensure DS and ES are correct
  - You must ensure SI and DI are offsets into DS and ES respectively
- Direction Flag (0 = Up, 1 = Down)
  - CLD - Increment addresses (left to right)
  - STD - Decrement addresses (right to left)

22/12/2010 $ 46

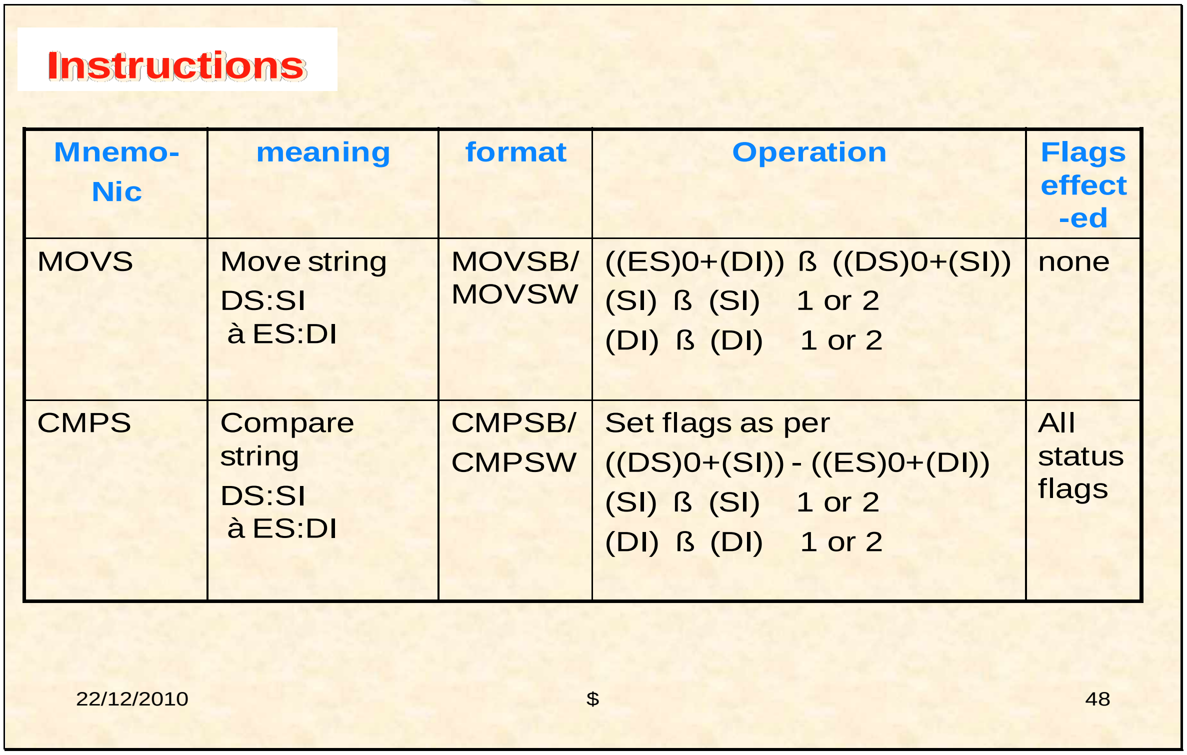


String Instructions

Instruction prefixes

Prefix	Used with	Meaning
REP	MOVS STOS	Repeat while not end of string CX ≠ 0
REPE/REPZ	CMPS SCAS	Repeat while not end of string and strings are equal. CX ≠ 0 and ZF = 1
REPNE/REPNZ	CMPS SCAS	Repeat while not end of string and strings are not equal. CX ≠ 0 and ZF = 0

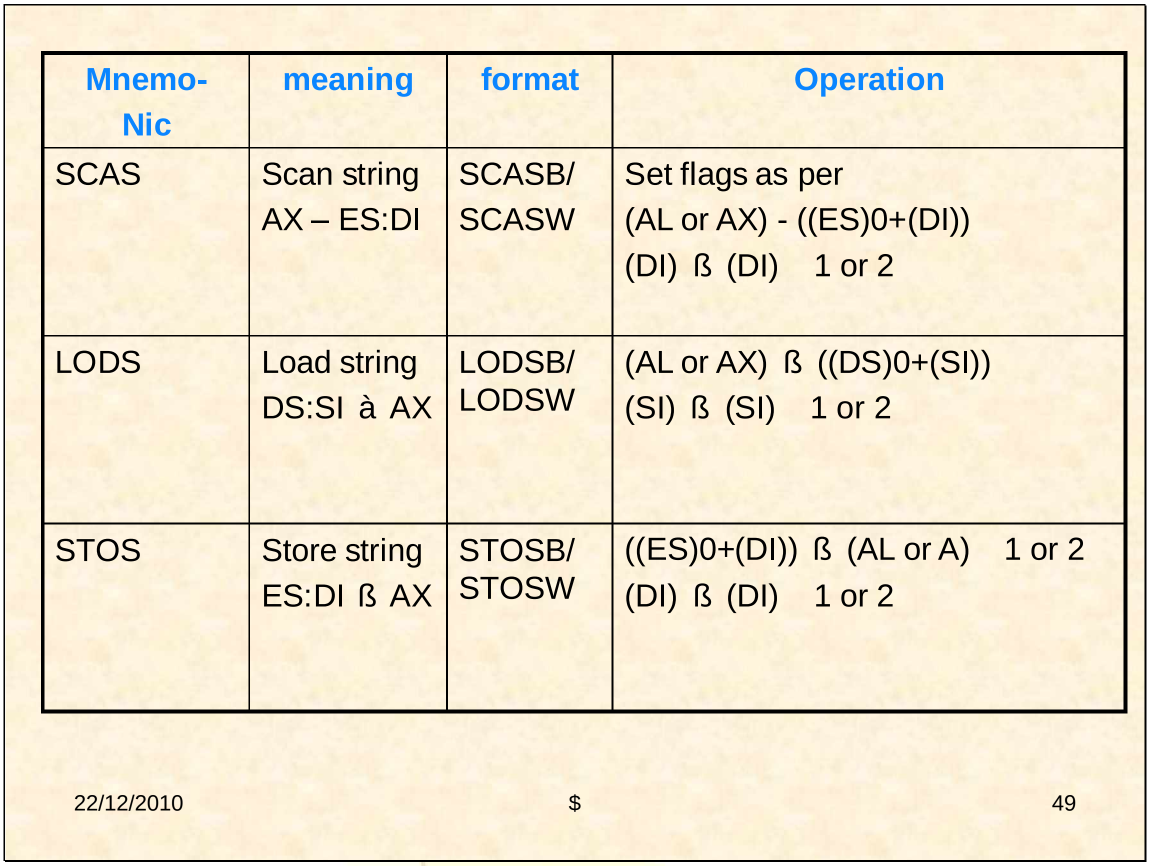
22/12/2010 $ 47



Instructions

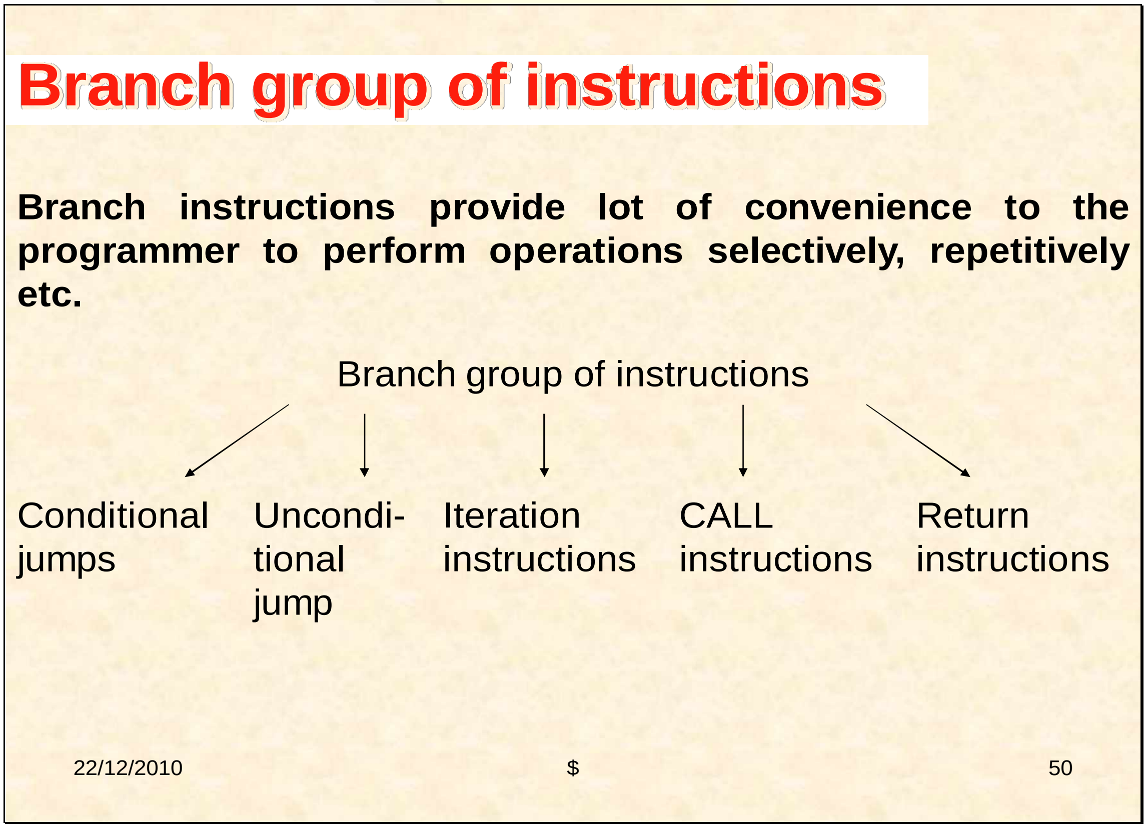
Mnemo-Nic	meaning	format	Operation	Flags effect-ed
MOVS	Move string DS:SI à ES:DI	MOVSB/ MOVSW	((ES)0+(DI)) ß ((DS)0+(SI)) (SI) ß (SI) 1 or 2 (DI) ß (DI) 1 or 2	none
CMPS	Compare string DS:SI à ES:DI	CMPSB/ CMPSW	Set flags as per ((DS)0+(SI)) - ((ES)0+(DI)) (SI) ß (SI) 1 or 2 (DI) ß (DI) 1 or 2	All status flags

22/12/2010 $ 48



Mnemo-Nic	meaning	format	Operation
SCAS	Scan string AX – ES:DI	SCASB/ SCASW	Set flags as per (AL or AX) - ((ES)0+(DI)) (DI) ß (DI) 1 or 2
LODS	Load string DS:SI à AX	LODSB/ LODSW	(AL or AX) ß ((DS)0+(SI)) (SI) ß (SI) 1 or 2
STOS	Store string ES:DI ß AX	STOSB/ STOSW	((ES)0+(DI)) ß (AL or A) 1 or 2 (DI) ß (DI) 1 or 2

22/12/2010 $ 49

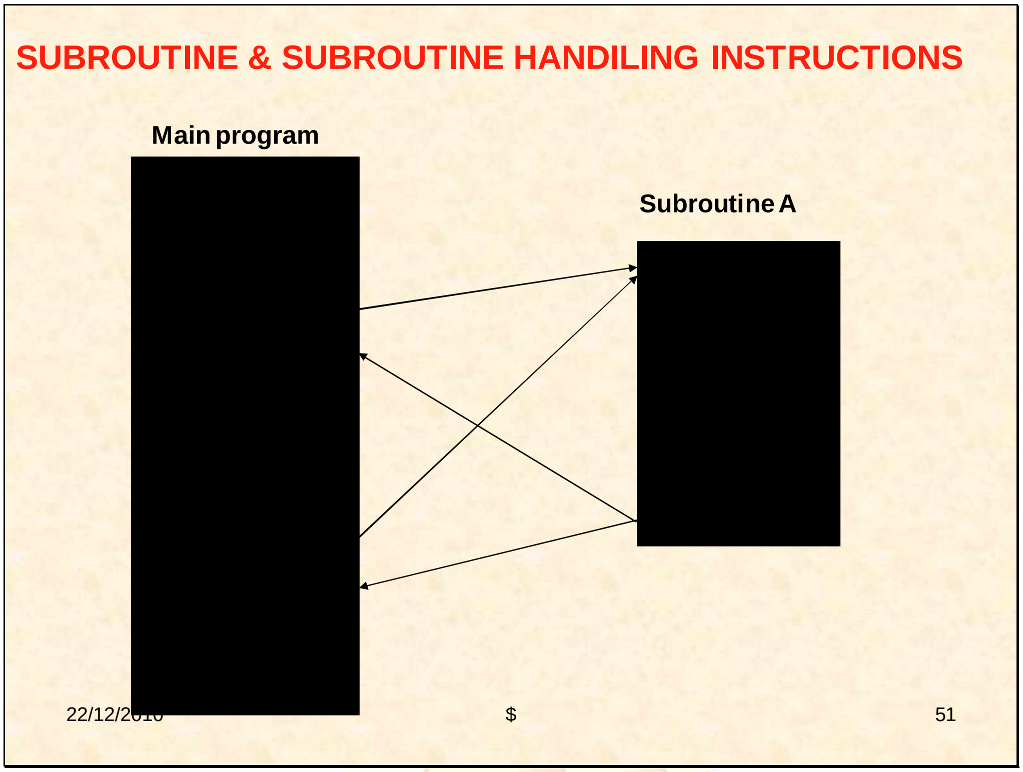


Branch group of instructions

Branch instructions provide lot of convenience to the programmer to perform operations selectively, repetitively etc.

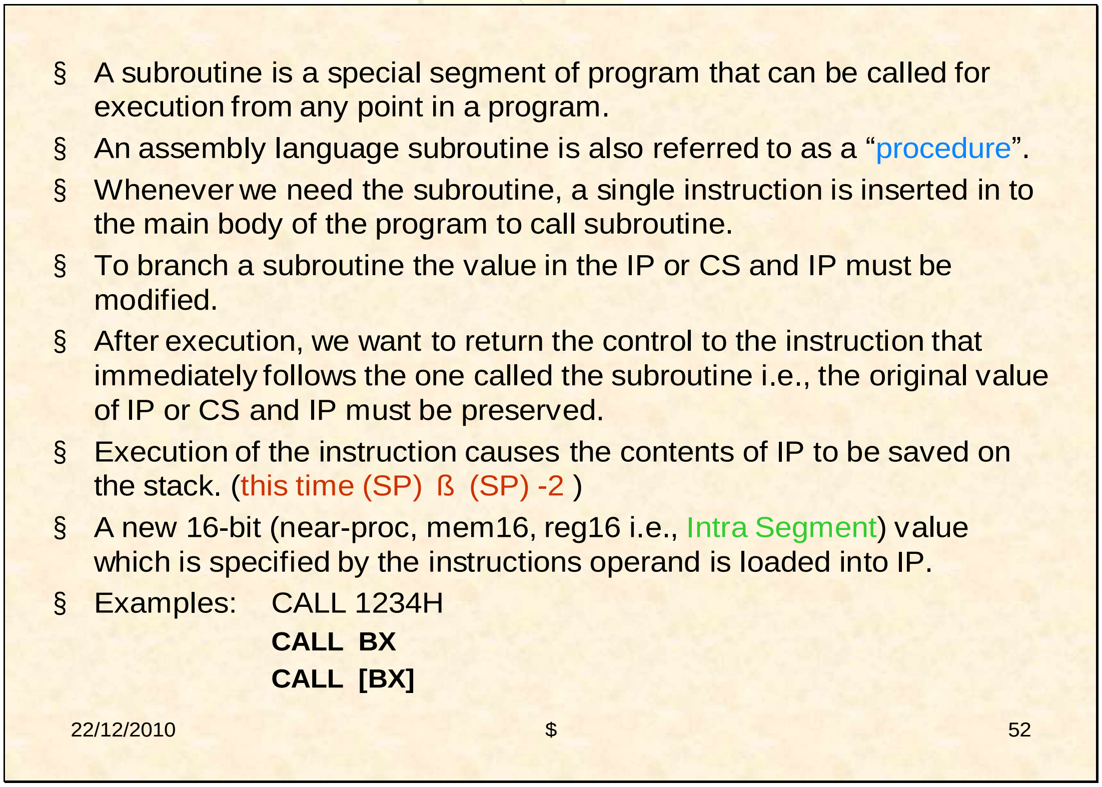
Branch group of instructions
Conditional jumps
Unconditional jump
Iteration instructions
CALL instructions
Return instructions

22/12/2010 $ 50



SUBROUTINE & SUBROUTINE HANDILING INSTRUCTIONS

22/12/2010 $ 51

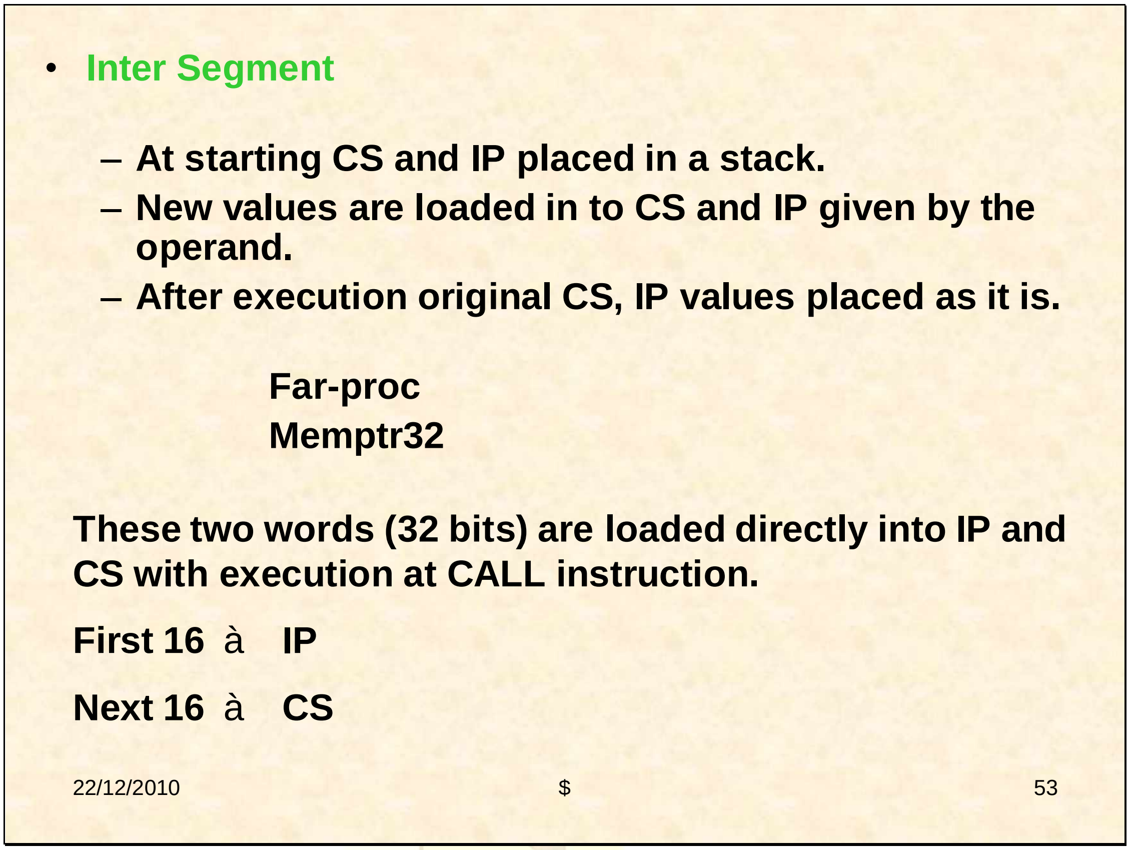


- A subroutine is a special segment of program that can be called for execution from any point in a program.
- An assembly language subroutine is also referred to as a “procedure”.
- Whenever we need the subroutine, a single instruction is inserted in to the main body of the program to call subroutine.
- To branch a subroutine the value in the IP or CS and IP must be modified.
- After execution, we want to return the control to the instruction that immediately follows the one called the subroutine i.e., the original value of IP or CS and IP must be preserved.
- Execution of the instruction causes the contents of IP to be saved on the stack. (this time (SP) ß (SP) -2)
- A new 16-bit (near-proc, mem16, reg16 i.e., Intra Segment) value which is specified by the instructions operand is loaded into IP.
- Examples: CALL 1234H

  CALL BX

  CALL [BX]

22/12/2010 $ 52



- Inter Segment
  - At starting CS and IP placed in a stack.
  - New values are loaded in to CS and IP given by the operand.
  - After execution original CS, IP values placed as it is.

Far-proc

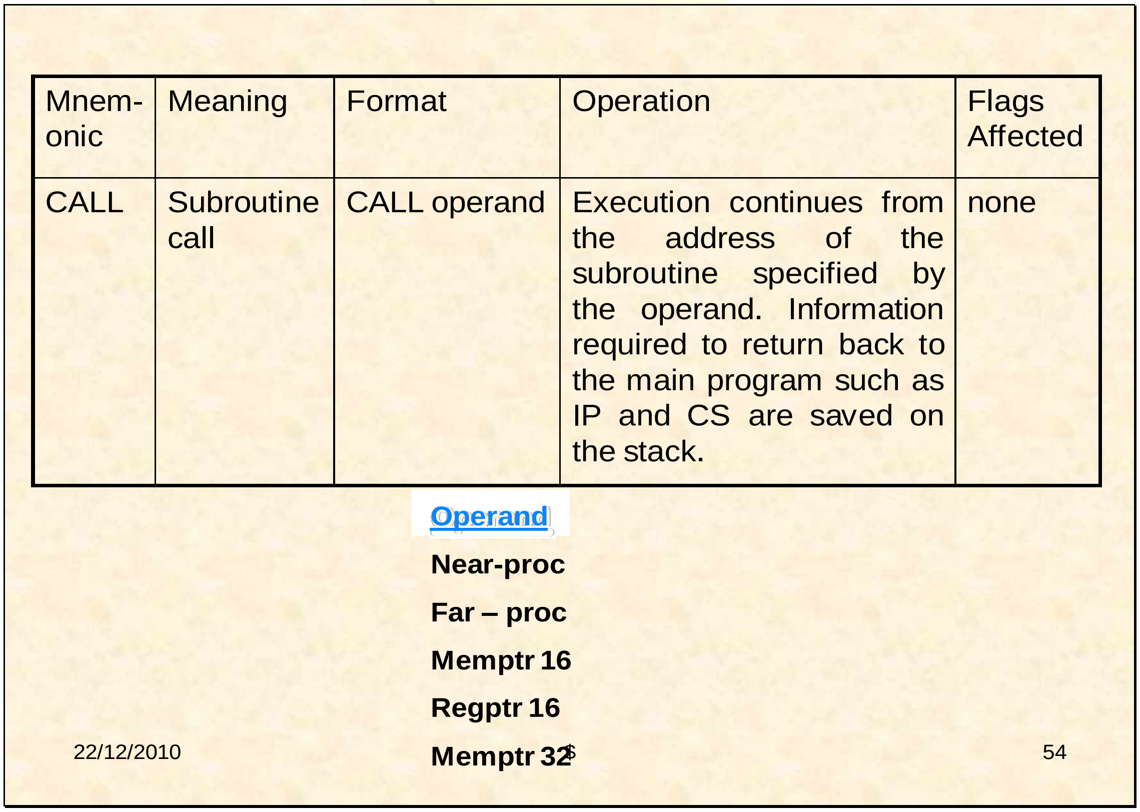
Memptr32

These two words (32 bits) are loaded directly into IP and CS with execution at CALL instruction.

First 16 à IP

Next 16 à CS

22/12/2010 $ 53



Mnem-onic	Meaning	Format	Operation	Flags Affected
CALL	Subroutine call	CALL operand	Execution continues from the address of the subroutine specified by the operand. Information required to return back to the main program such as IP and CS are saved on the stack.	none

Operand

Near-proc

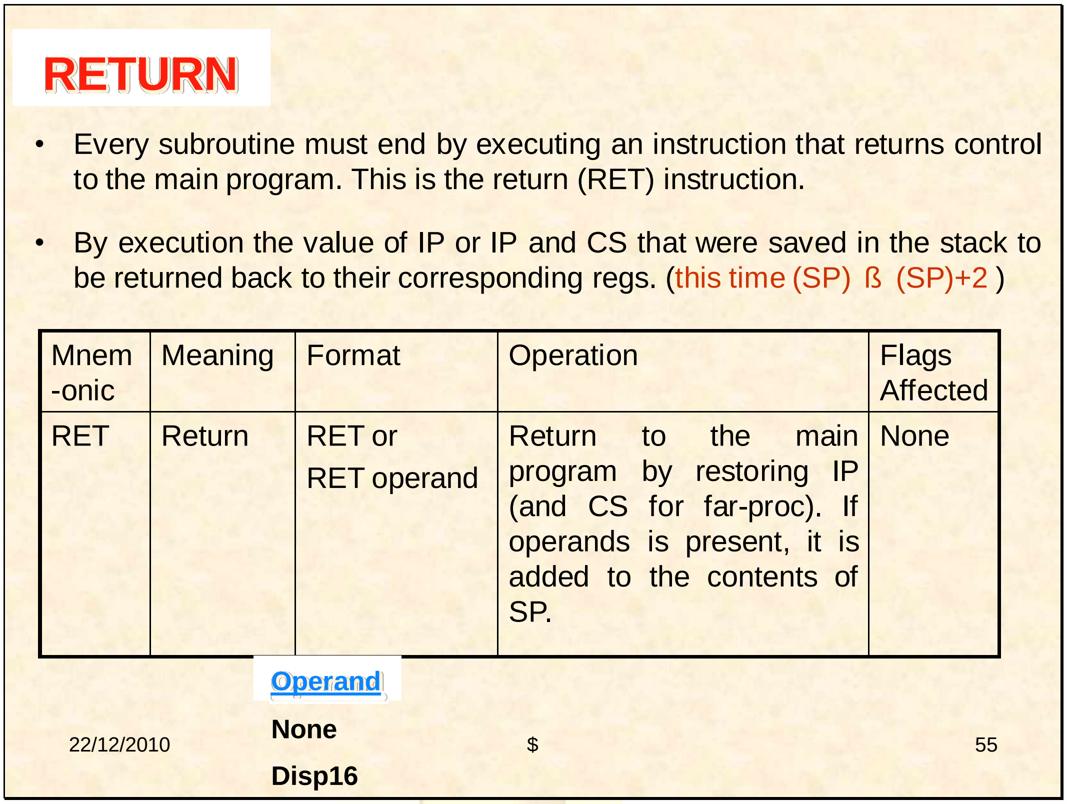
Far – proc

Memptr 16

Regptr 16

Memptr 32

22/12/2010 $ 54



RETURN

- Every subroutine must end by executing an instruction that returns control to the main program. This is the return (RET) instruction.
- By execution the value of IP or IP and CS that were saved in the stack to be returned back to their corresponding regs. (this time (SP) ß (SP)+2)

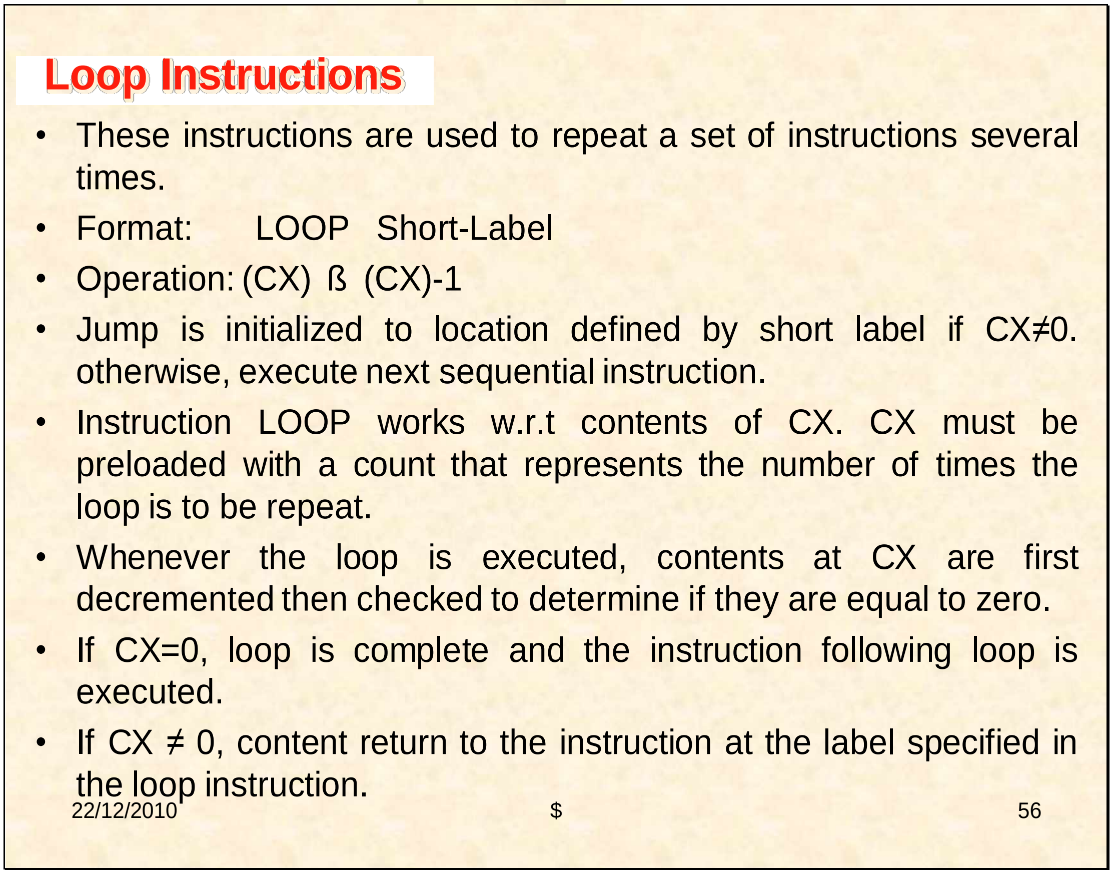
Mnem -onic	Meaning	Format	Operation	Flags Affected
RET	Return	RET or RET operand	Return to the main program by restoring IP (and CS for far-proc). If operands is present, it is added to the contents of SP.	None

Operand

None

Disp16

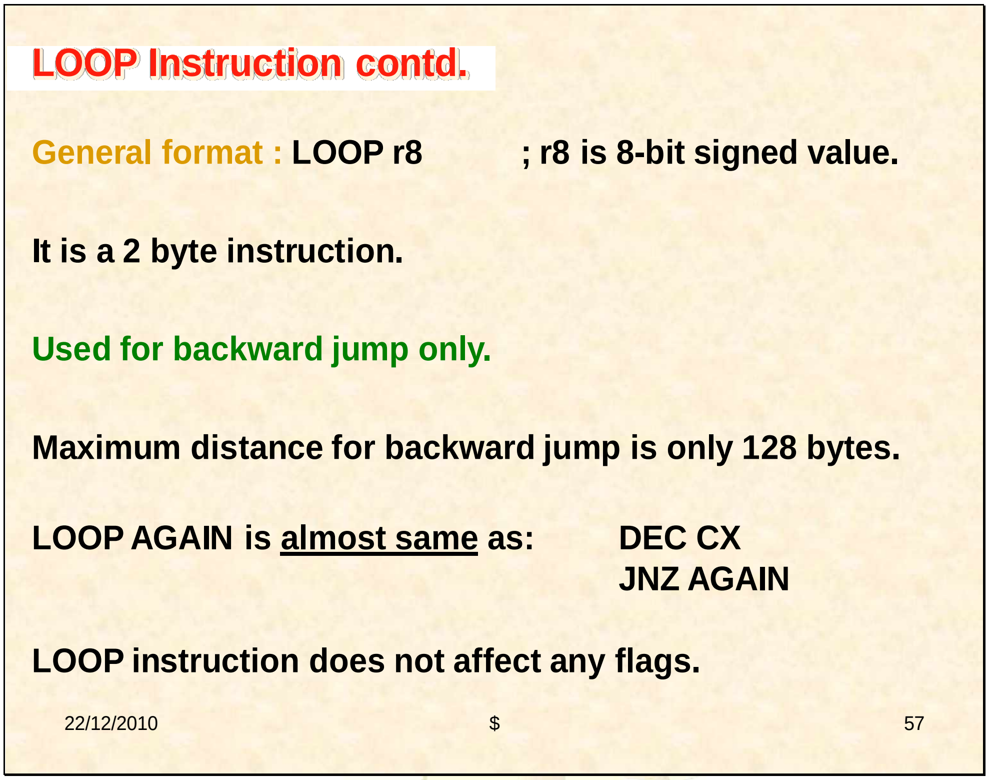
22/12/2010 $ 55



Loop Instructions

- These instructions are used to repeat a set of instructions several times.
- Format: LOOP Short-Label
- Operation: (CX) ß (CX)-1
- Jump is initialized to location defined by short label if CX≠0. otherwise, execute next sequential instruction.
- Instruction LOOP works w.r.t contents of CX. CX must be preloaded with a count that represents the number of times the loop is to be repeat.
- Whenever the loop is executed, contents at CX are first decremented then checked to determine if they are equal to zero.
- If CX=0, loop is complete and the instruction following loop is executed.
- If CX ≠ 0, content return to the instruction at the label specified in the loop instruction.

22/12/2010 $ 56



LOOP Instruction contd.

General format : **LOOP r8** **; r8 is 8-bit signed value.**

It is a 2 byte instruction.

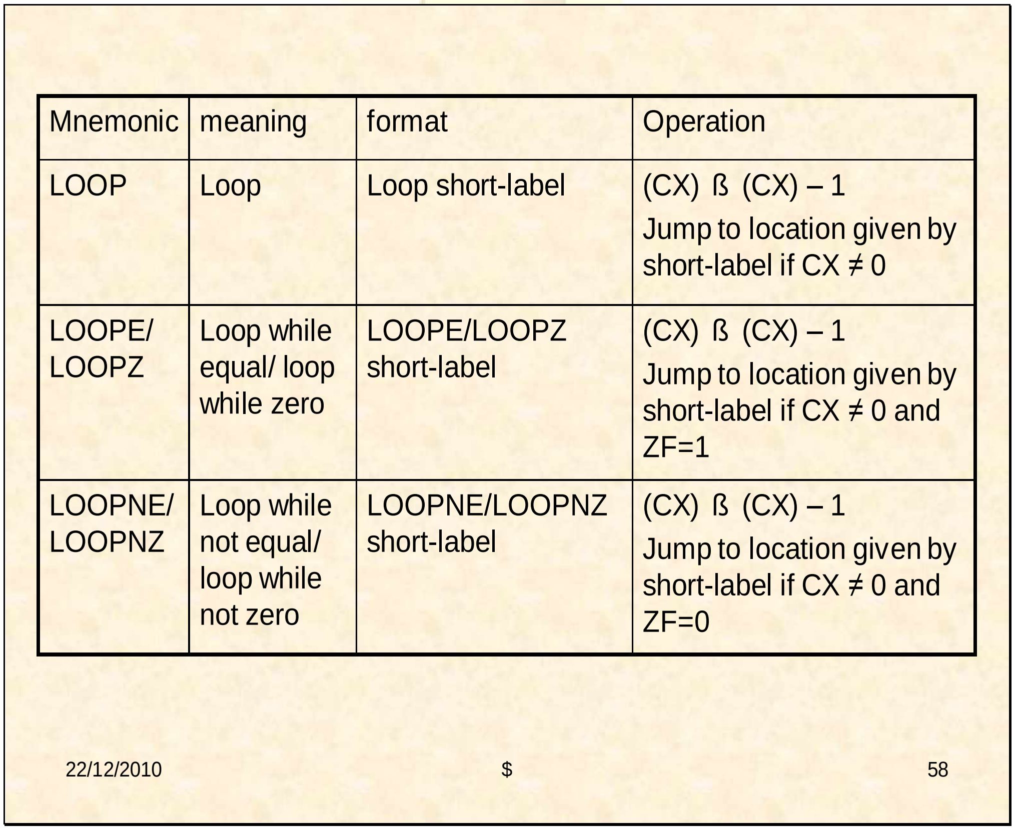
Used for backward jump only.

Maximum distance for backward jump is only 128 bytes.

LOOP AGAIN is <u>almost same</u> as: **DEC CX**
JNZ AGAIN

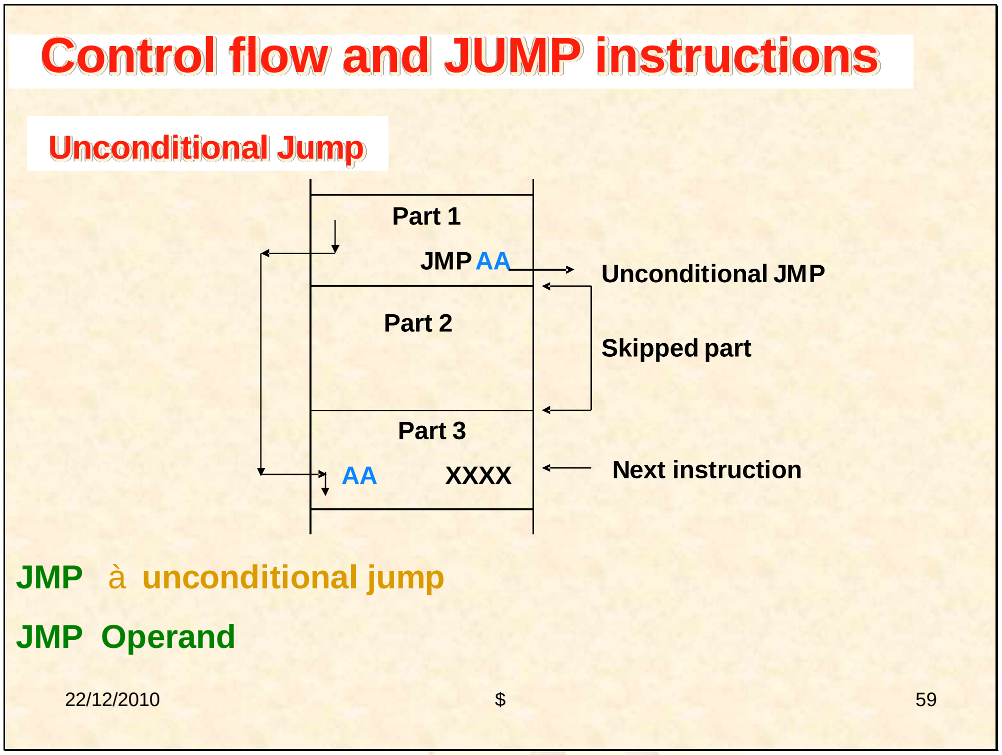
LOOP instruction does not affect any flags.

22/12/2010 $ 57



Mnemonic	meaning	format	Operation
LOOP	Loop	Loop short-label	(CX) ß (CX) – 1 Jump to location given by short-label if CX ≠ 0
LOOPE/ LOOPZ	Loop while equal/ loop while zero	LOOPE/LOOPZ short-label	(CX) ß (CX) – 1 Jump to location given by short-label if CX ≠ 0 and ZF=1
LOOPNE/ LOOPNZ	Loop while not equal/ loop while not zero	LOOPNE/LOOPNZ short-label	(CX) ß (CX) – 1 Jump to location given by short-label if CX ≠ 0 and ZF=0

22/12/2010 $ 58



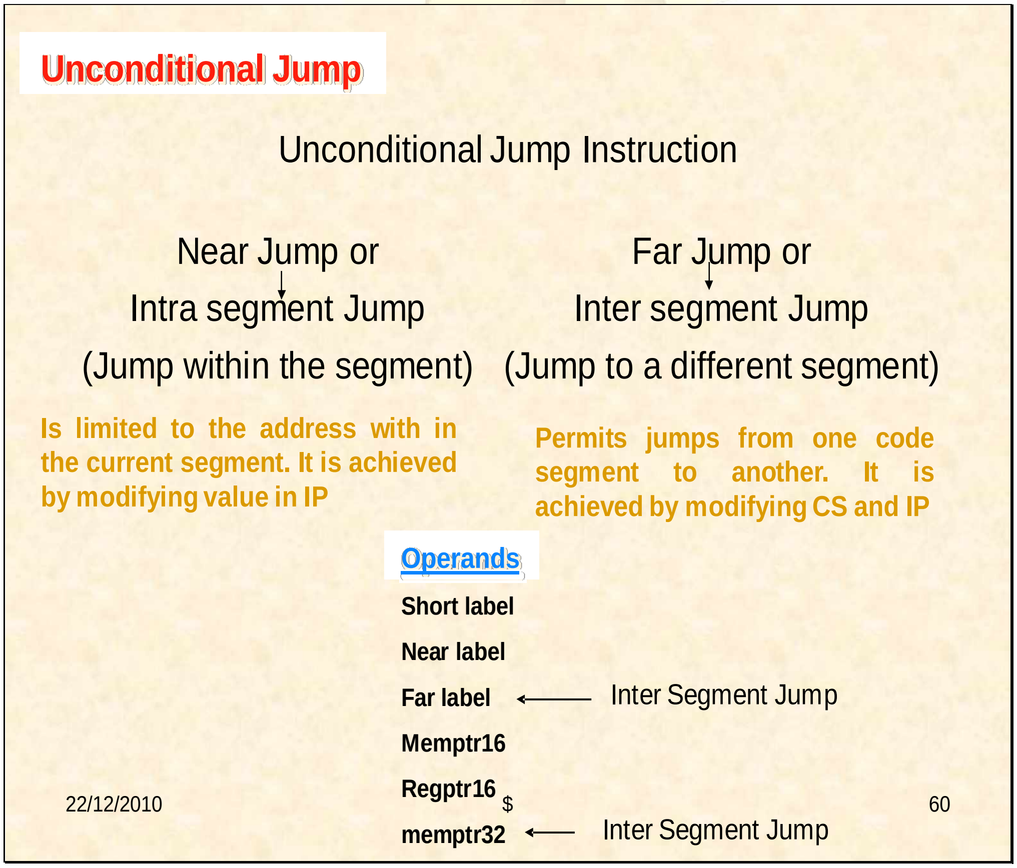
Control flow and JUMP instructions

Unconditional Jump

JMP à unconditional jump

JMP Operand

22/12/2010 $ 59



Unconditional Jump

Unconditional Jump Instruction

Near Jump or Intra segment Jump (Jump within the segment)	Far Jump or Inter segment Jump (Jump to a different segment)
Is limited to the address with in the current segment. It is achieved by modifying value in IP	**Permits jumps from one code segment to another. It is achieved by modifying CS and IP**

Operands

Short label

Near label

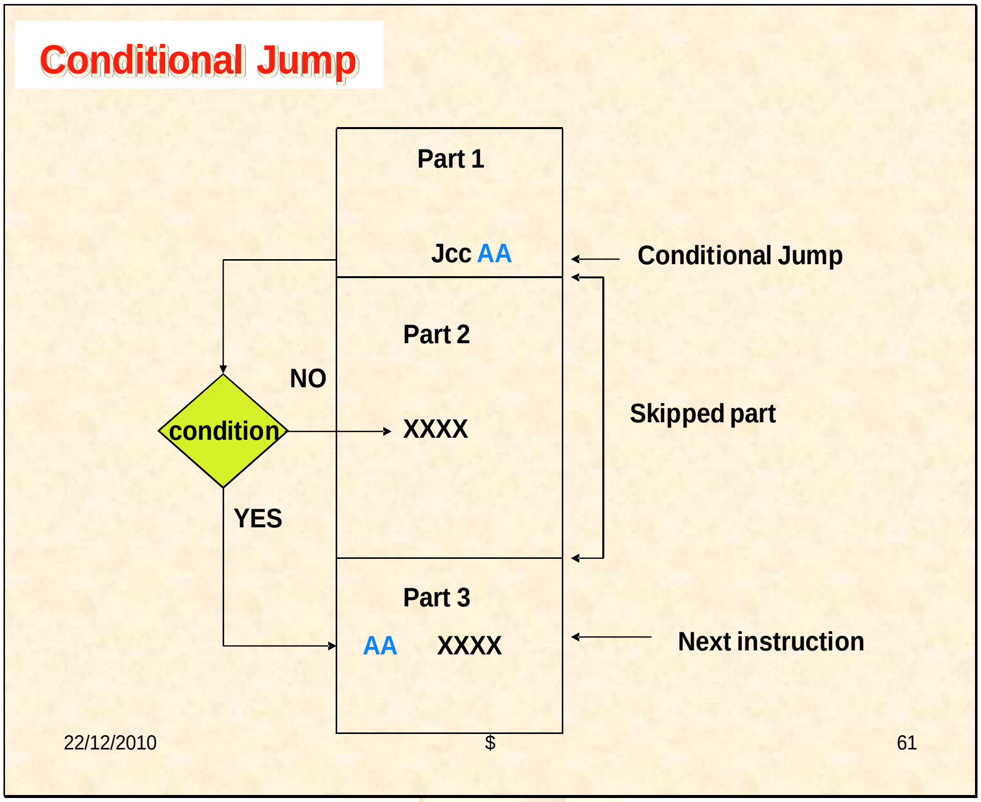
Far label ← Inter Segment Jump

Memptr16

Regptr16

memptr32 ← Inter Segment Jump

22/12/2010 $ 60



Conditional Jump

Part 1

Jcc AA

Conditional Jump

Part 2

NO

condition

XXXX

Skipped part

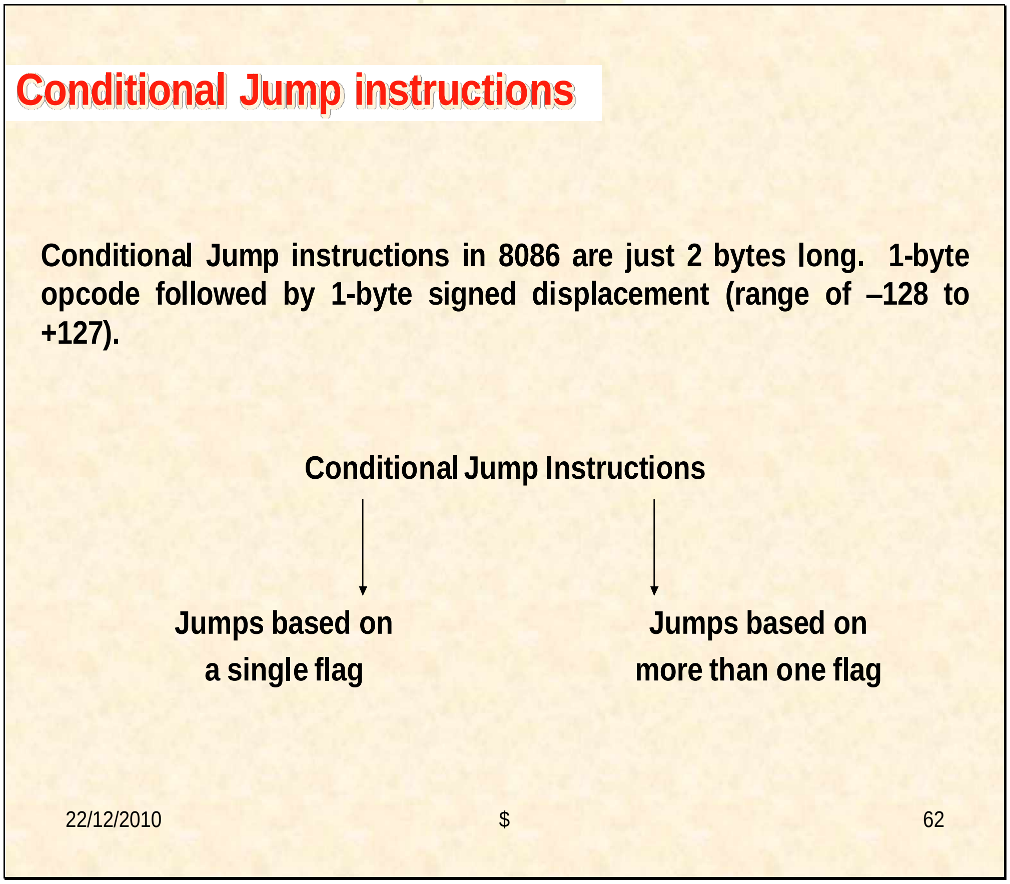
YES

Part 3

AA XXXX

Next instruction

22/12/2010 $ 61



Conditional Jump instructions

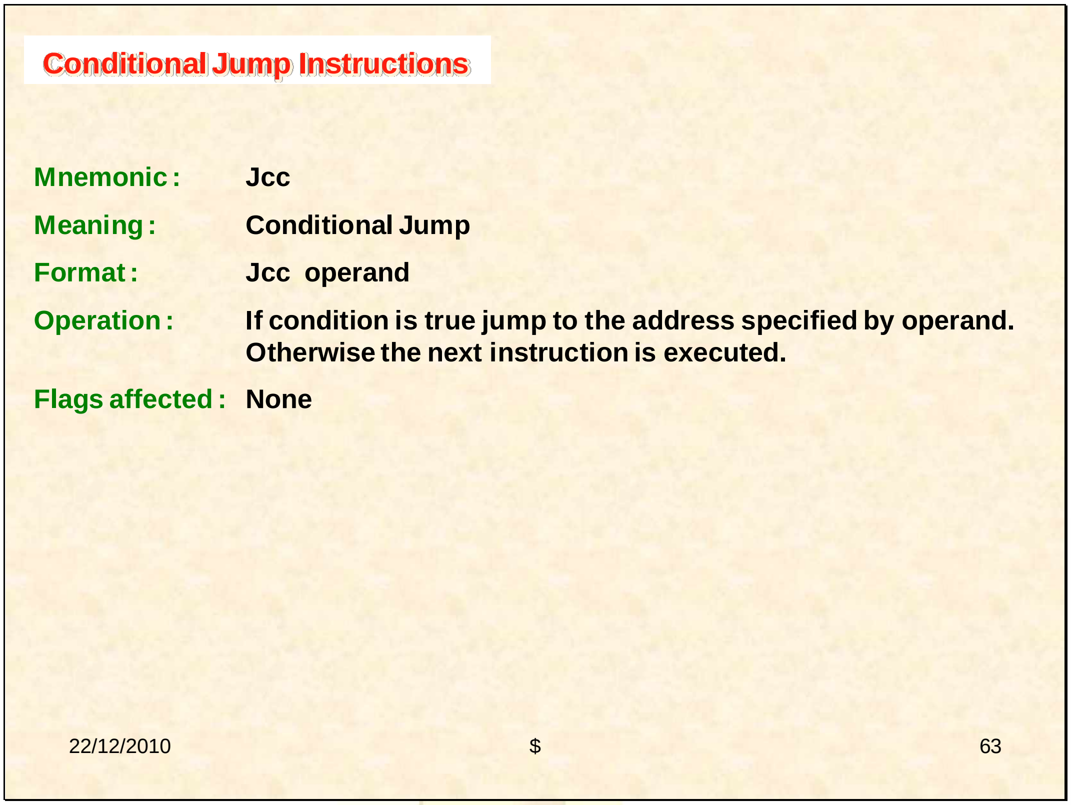
Conditional Jump instructions in 8086 are just 2 bytes long. 1-byte opcode followed by 1-byte signed displacement (range of –128 to +127).

Conditional Jump Instructions

Jumps based on
a single flag

Jumps based on
more than one flag

22/12/2010 $ 62



Conditional Jump Instructions

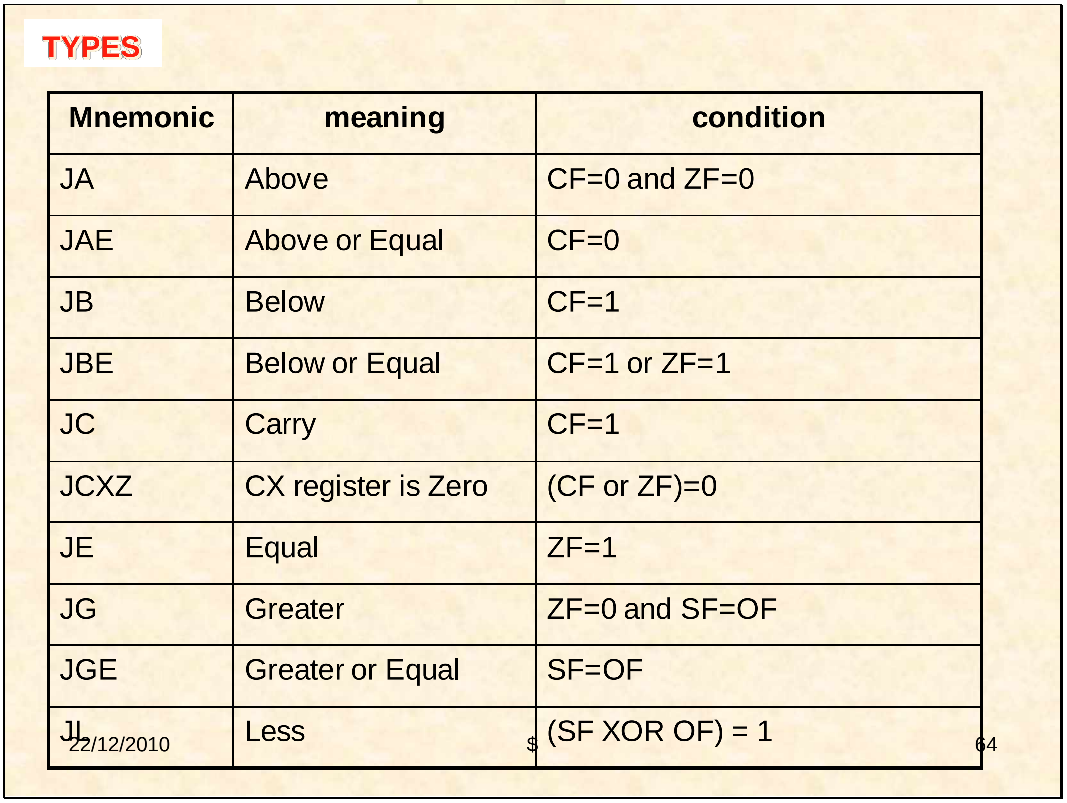
Mnemonic : **Jcc**

Meaning : **Conditional Jump**

Format : **Jcc operand**

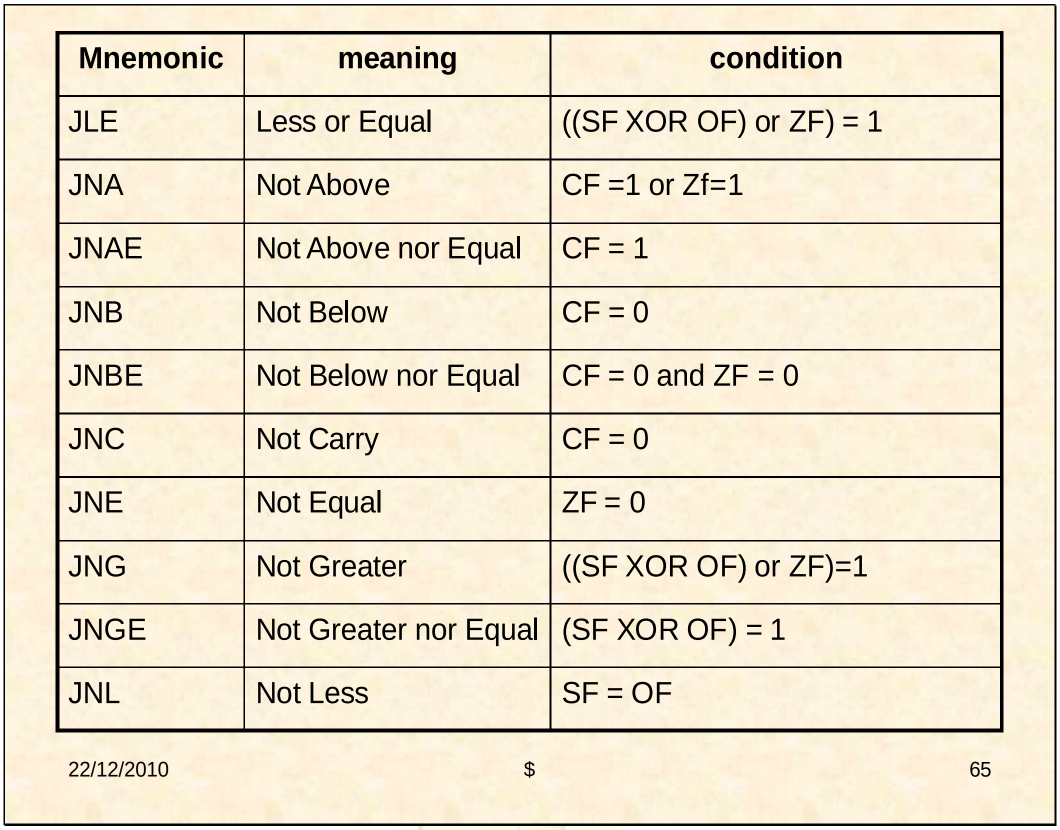
Operation : **If condition is true jump to the address specified by operand. Otherwise the next instruction is executed.**

Flags affected : **None**



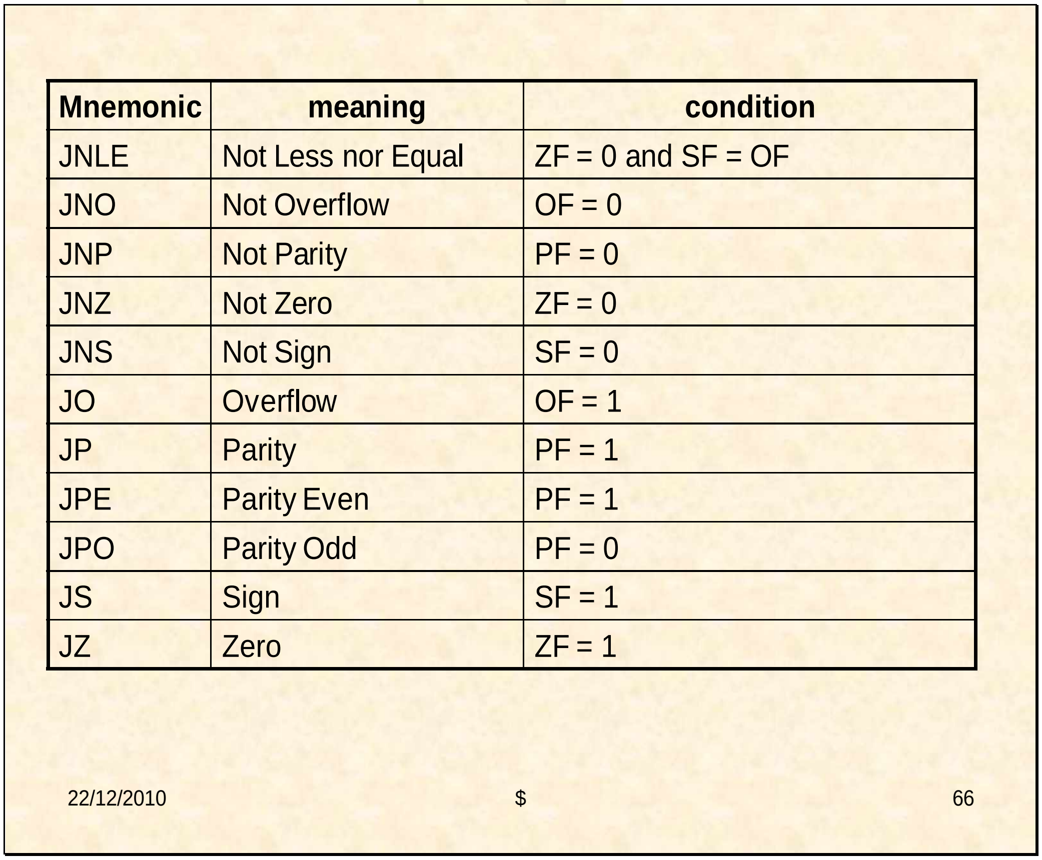
TYPES

Mnemonic	meaning	condition
JA	Above	CF=0 and ZF=0
JAE	Above or Equal	CF=0
JB	Below	CF=1
JBE	Below or Equal	CF=1 or ZF=1
JC	Carry	CF=1
JCXZ	CX register is Zero	(CF or ZF)=0
JE	Equal	ZF=1
JG	Greater	ZF=0 and SF=OF
JGE	Greater or Equal	SF=OF
JL	Less	(SF XOR OF) = 1



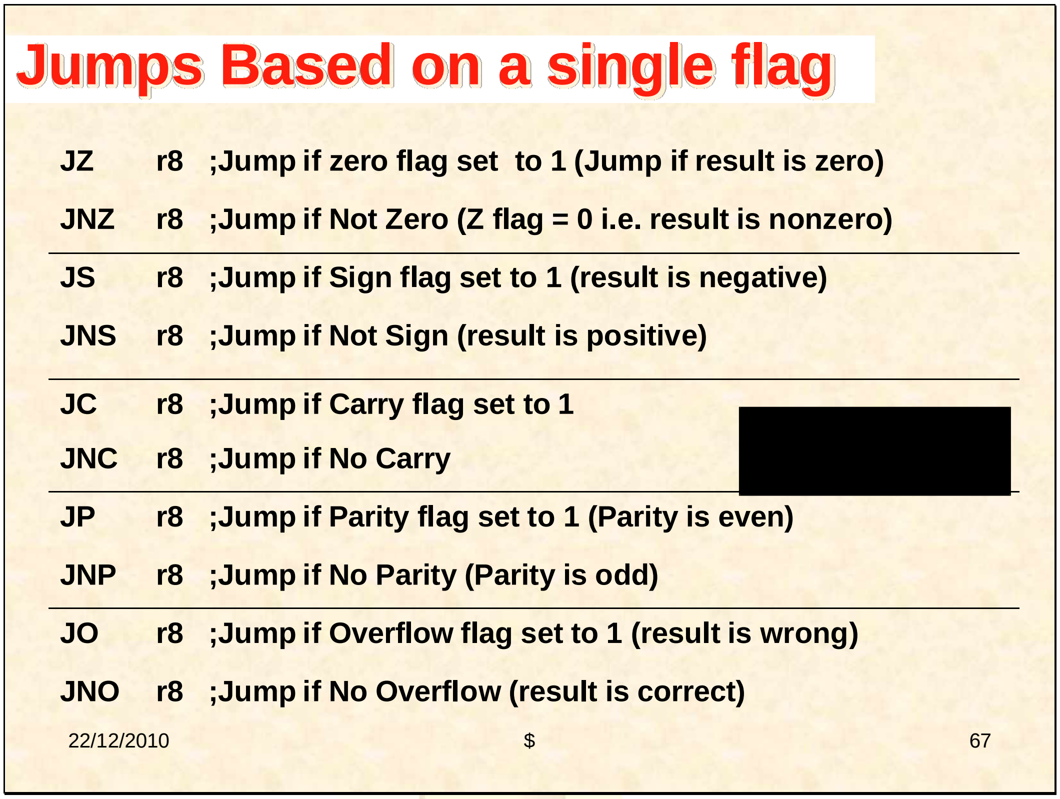
Mnemonic	meaning	condition
JLE	Less or Equal	((SF XOR OF) or ZF) = 1
JNA	Not Above	CF =1 or Zf=1
JNAE	Not Above nor Equal	CF = 1
JNB	Not Below	CF = 0
JNBE	Not Below nor Equal	CF = 0 and ZF = 0
JNC	Not Carry	CF = 0
JNE	Not Equal	ZF = 0
JNG	Not Greater	((SF XOR OF) or ZF)=1
JNGE	Not Greater nor Equal	(SF XOR OF) = 1
JNL	Not Less	SF = OF

22/12/2010 $ 65



Mnemonic	meaning	condition
JNLE	Not Less nor Equal	ZF = 0 and SF = OF
JNO	Not Overflow	OF = 0
JNP	Not Parity	PF = 0
JNZ	Not Zero	ZF = 0
JNS	Not Sign	SF = 0
JO	Overflow	OF = 1
JP	Parity	PF = 1
JPE	Parity Even	PF = 1
JPO	Parity Odd	PF = 0
JS	Sign	SF = 1
JZ	Zero	ZF = 1

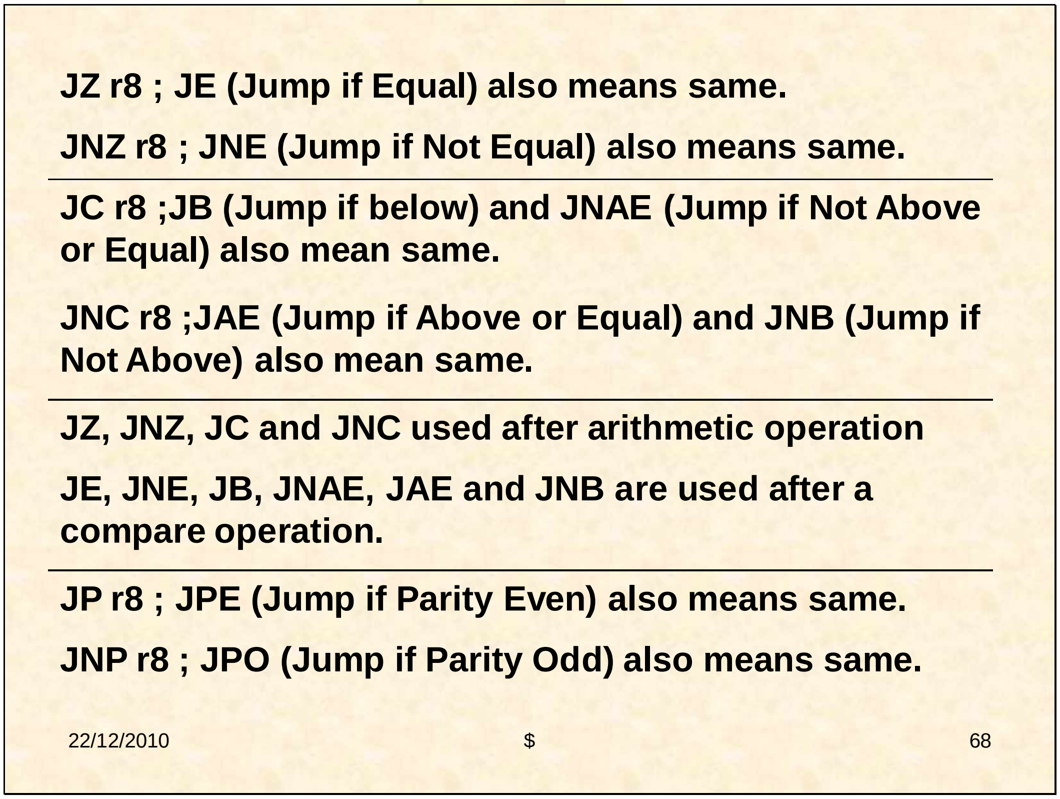
22/12/2010 $ 66



Jumps Based on a single flag

JZ	r8	;Jump if zero flag set to 1 (Jump if result is zero)
JNZ	r8	;Jump if Not Zero (Z flag = 0 i.e. result is nonzero)
JS	r8	;Jump if Sign flag set to 1 (result is negative)
JNS	r8	;Jump if Not Sign (result is positive)
JC	r8	;Jump if Carry flag set to 1
JNC	r8	;Jump if No Carry
JP	r8	;Jump if Parity flag set to 1 (Parity is even)
JNP	r8	;Jump if No Parity (Parity is odd)
JO	r8	;Jump if Overflow flag set to 1 (result is wrong)
JNO	r8	;Jump if No Overflow (result is correct)

[redacted]



JZ r8 ; JE (Jump if Equal) also means same.

JNZ r8 ; JNE (Jump if Not Equal) also means same.

JC r8 ;JB (Jump if below) and JNAE (Jump if Not Above or Equal) also mean same.

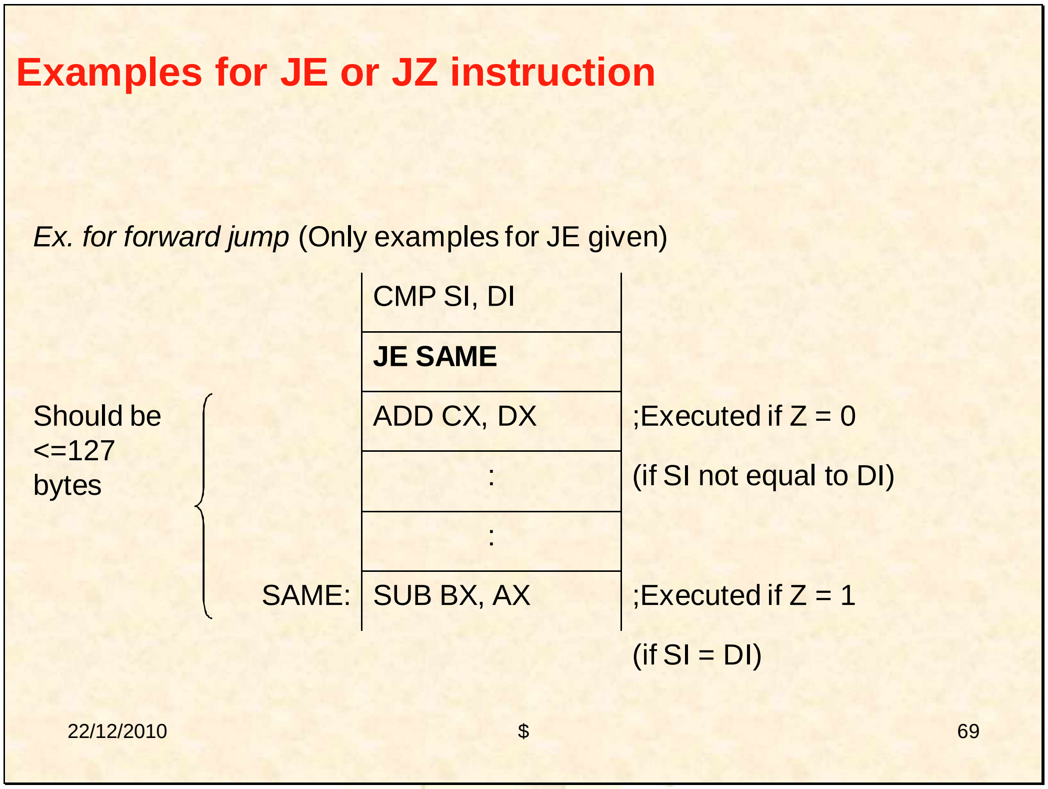
JNC r8 ;JAE (Jump if Above or Equal) and JNB (Jump if Not Above) also mean same.

JZ, JNZ, JC and JNC used after arithmetic operation

JE, JNE, JB, JNAE, JAE and JNB are used after a compare operation.

JP r8 ; JPE (Jump if Parity Even) also means same.

JNP r8 ; JPO (Jump if Parity Odd) also means same.

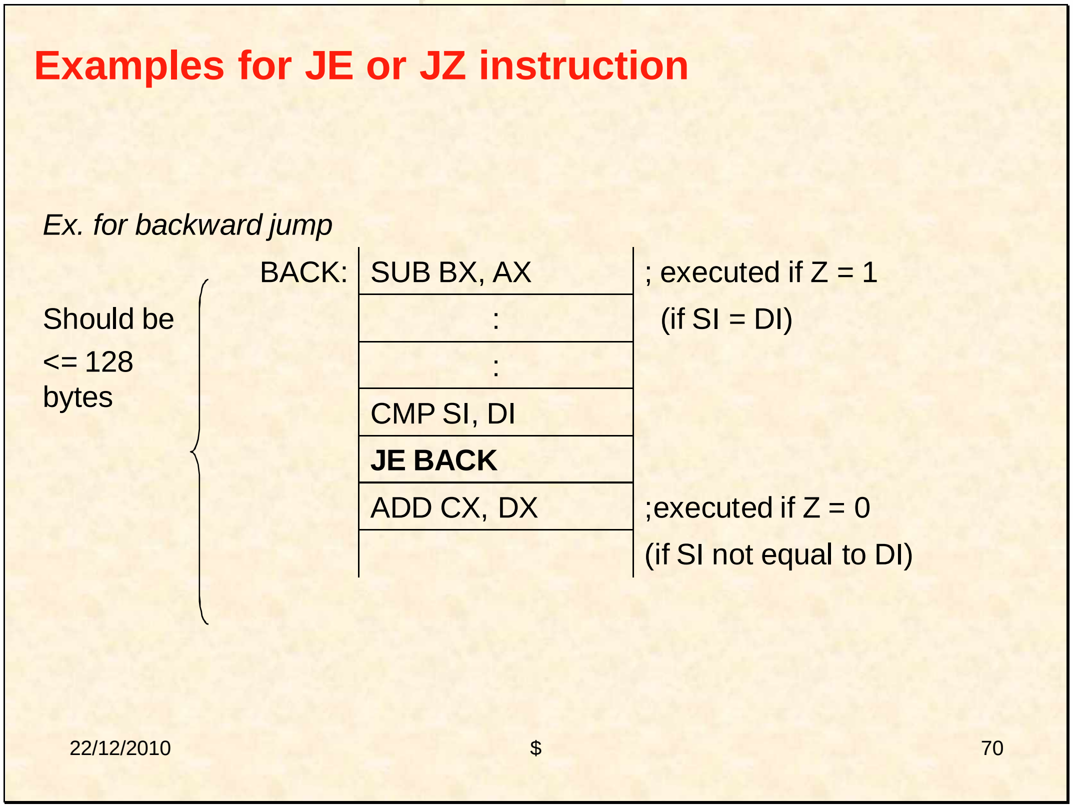


Examples for JE or JZ instruction

Ex. for forward jump (Only examples for JE given)

```
                          CMP SI, DI
                          JE SAME
Should be  (              ADD CX, DX    ;Executed if Z = 0
<=127      (                  :         (if SI not equal to DI)
bytes      (                  :
           (       SAME:  SUB BX, AX    ;Executed if Z = 1
                                        (if SI = DI)
```

22/12/2010 $ 69

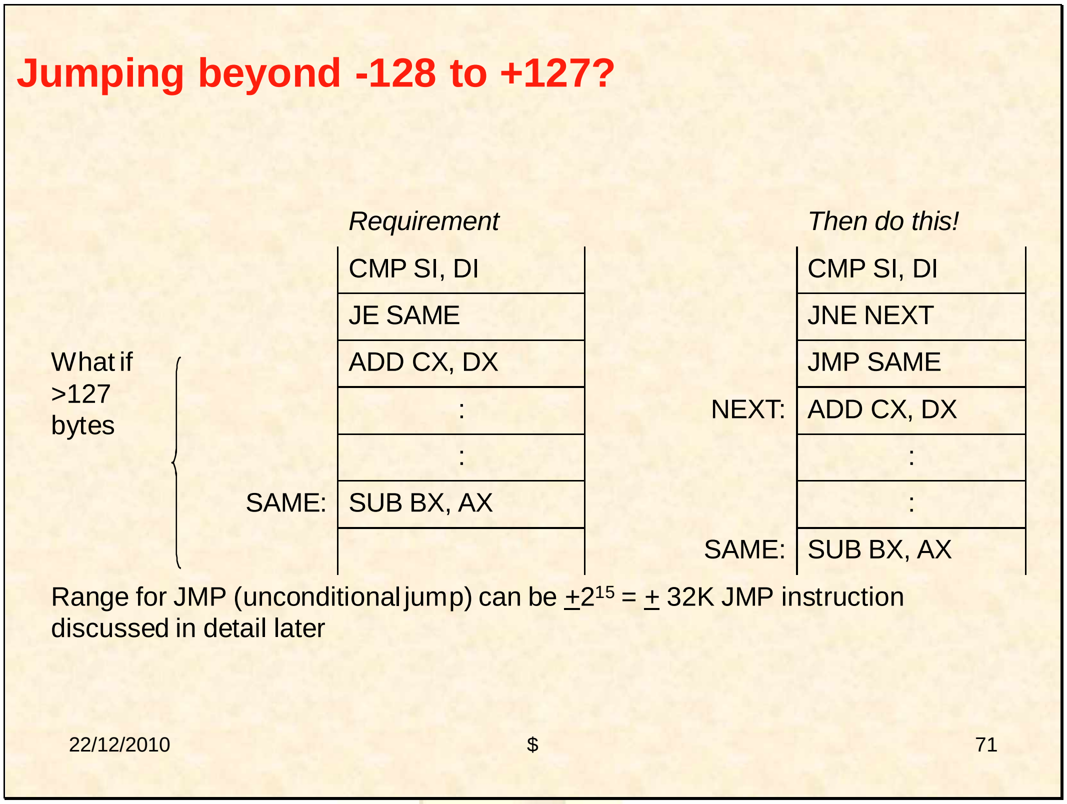


Examples for JE or JZ instruction

Ex. for backward jump

```
                   BACK:  SUB BX, AX    ; executed if Z = 1
Should be  (                  :           (if SI = DI)
<= 128     (                  :
bytes      (              CMP SI, DI
           (              JE BACK
           (              ADD CX, DX    ;executed if Z = 0
                                        (if SI not equal to DI)
```

22/12/2010 $ 70

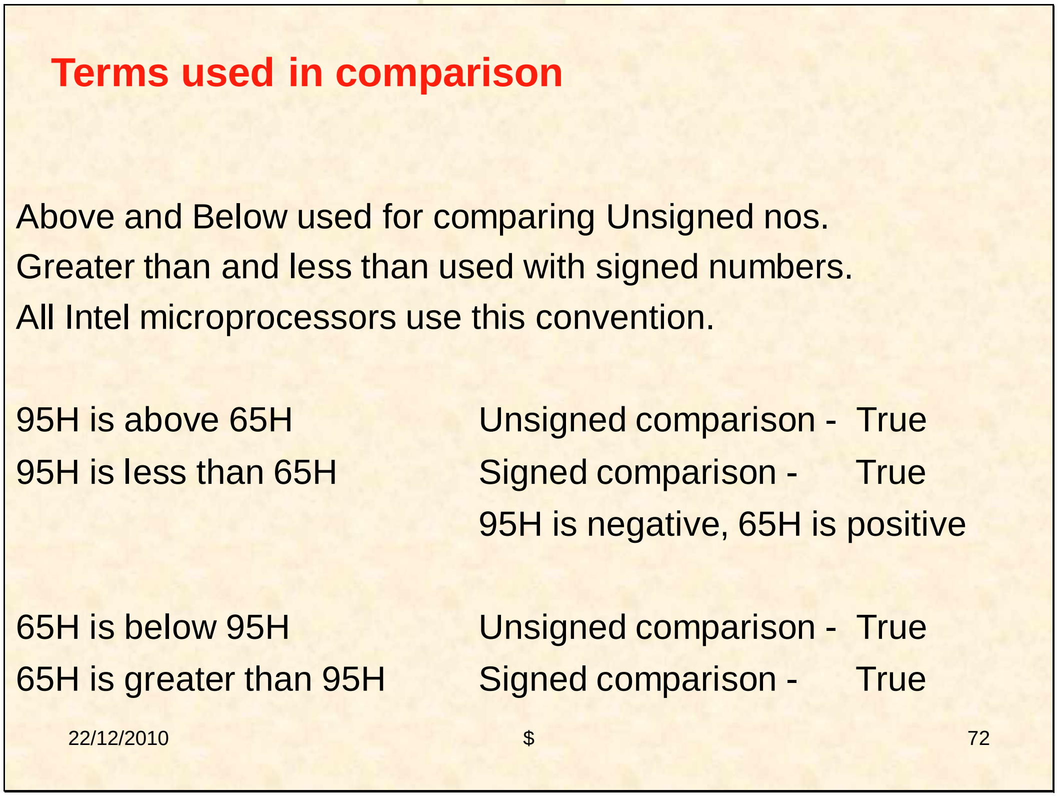


Jumping beyond -128 to +127?

		Requirement		*Then do this!*
		CMP SI, DI		CMP SI, DI
		JE SAME		JNE NEXT
What if >127 bytes		ADD CX, DX		JMP SAME
		:	NEXT:	ADD CX, DX
		:		:
	SAME:	SUB BX, AX		:
			SAME:	SUB BX, AX

Range for JMP (unconditional jump) can be $\underline{+}2^{15}$ = $\underline{+}$ 32K JMP instruction discussed in detail later

22/12/2010 $ 71

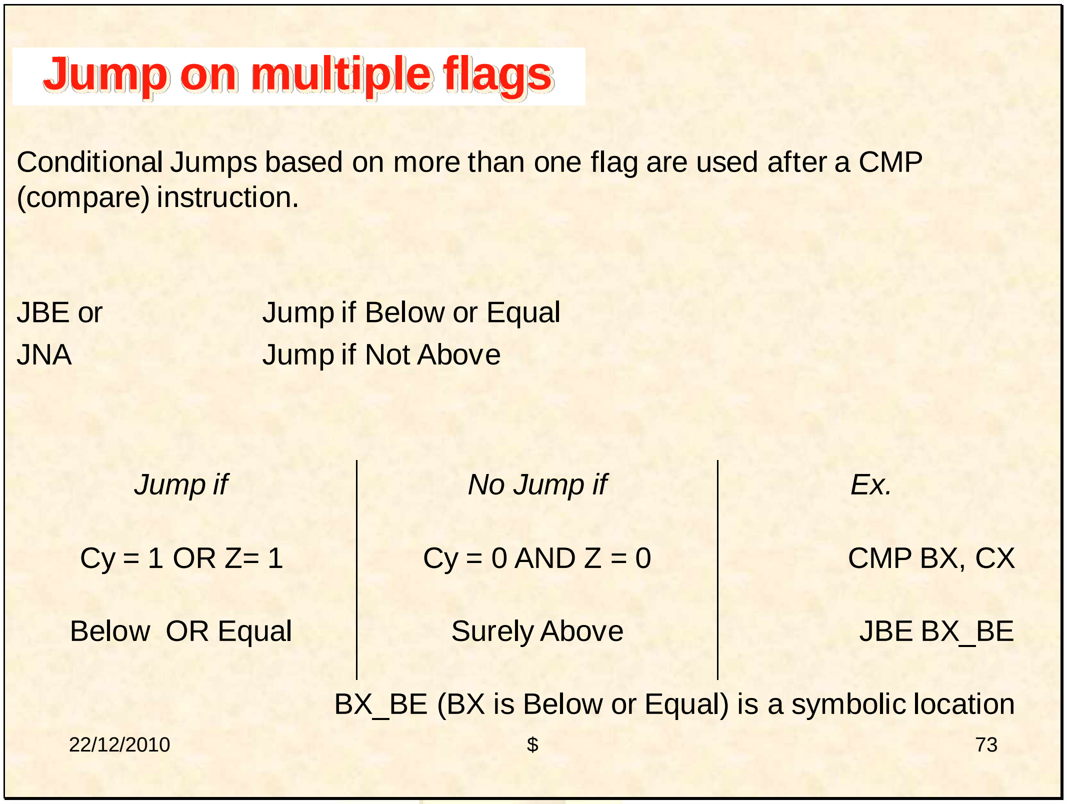


Terms used in comparison

Above and Below used for comparing Unsigned nos.
Greater than and less than used with signed numbers.
All Intel microprocessors use this convention.

95H is above 65H	Unsigned comparison - True
95H is less than 65H	Signed comparison - True
	95H is negative, 65H is positive
65H is below 95H	Unsigned comparison - True
65H is greater than 95H	Signed comparison - True

22/12/2010 $ 72



Jump on multiple flags

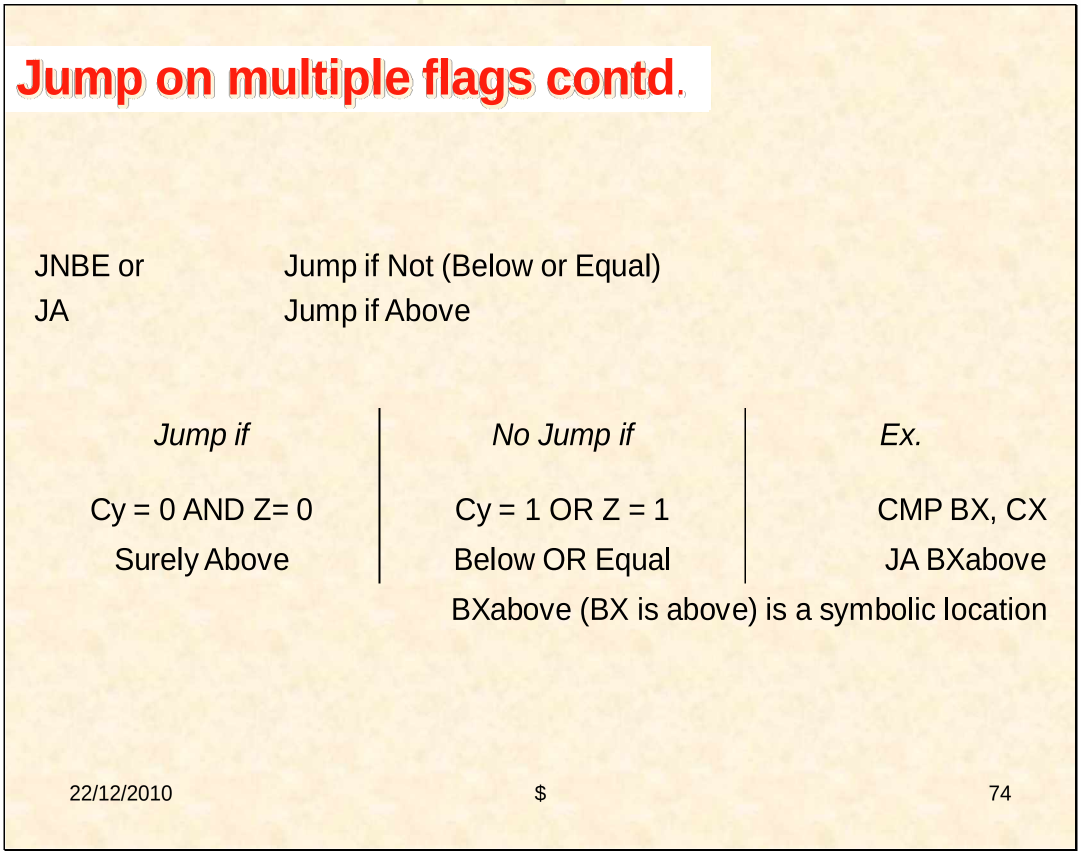
Conditional Jumps based on more than one flag are used after a CMP (compare) instruction.

JBE or	Jump if Below or Equal
JNA	Jump if Not Above

Jump if	*No Jump if*	*Ex.*
Cy = 1 OR Z= 1	Cy = 0 AND Z = 0	CMP BX, CX
Below OR Equal	Surely Above	JBE BX_BE

BX_BE (BX is Below or Equal) is a symbolic location

22/12/2010 $ 73



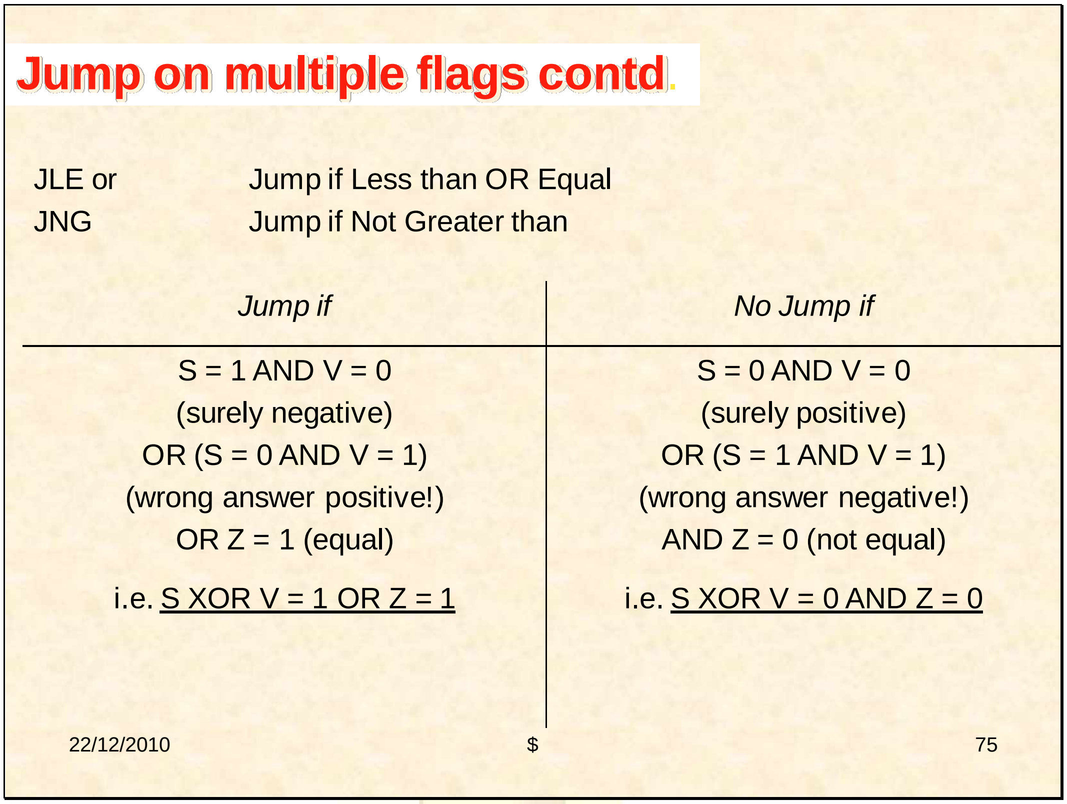
Jump on multiple flags contd.

JNBE or	Jump if Not (Below or Equal)
JA	Jump if Above

Jump if	*No Jump if*	*Ex.*
Cy = 0 AND Z= 0	Cy = 1 OR Z = 1	CMP BX, CX
Surely Above	Below OR Equal	JA BXabove

BXabove (BX is above) is a symbolic location

22/12/2010 $ 74

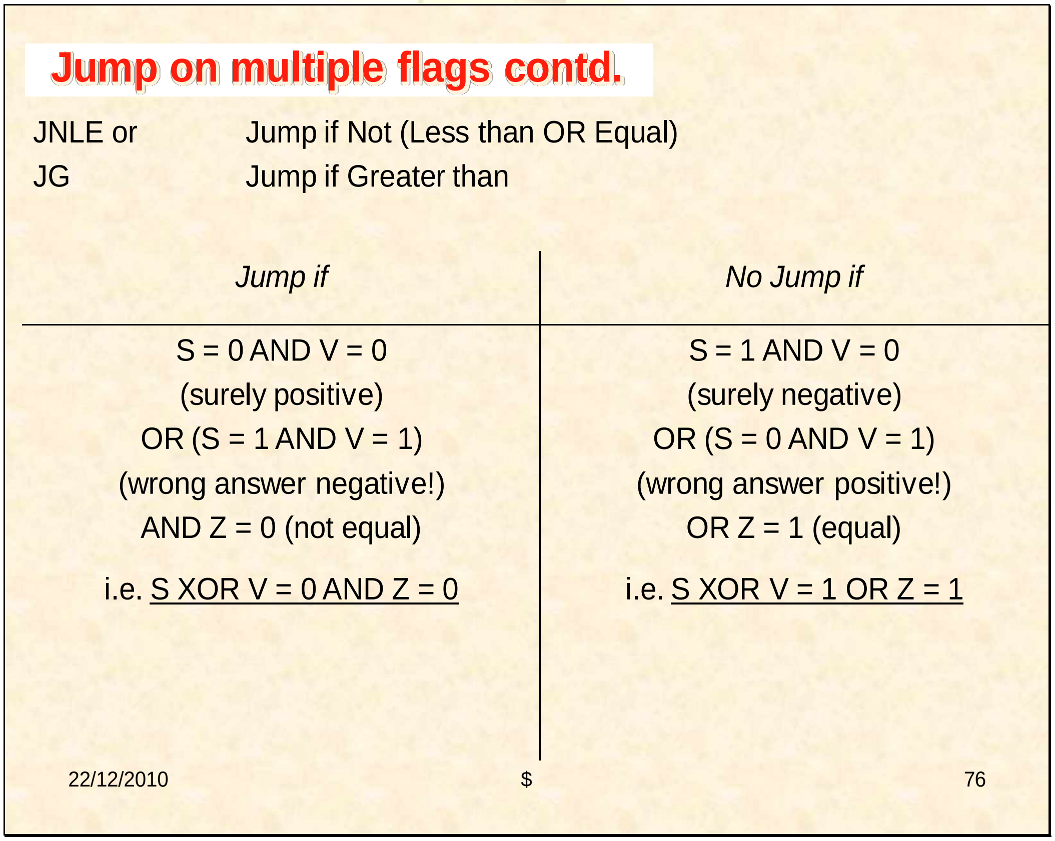


Jump on multiple flags contd.

JLE or Jump if Less than OR Equal
JNG Jump if Not Greater than

Jump if	*No Jump if*
S = 1 AND V = 0 (surely negative) OR (S = 0 AND V = 1) (wrong answer positive!) OR Z = 1 (equal) i.e. S XOR V = 1 OR Z = 1	S = 0 AND V = 0 (surely positive) OR (S = 1 AND V = 1) (wrong answer negative!) AND Z = 0 (not equal) i.e. S XOR V = 0 AND Z = 0

22/12/2010 $ 75

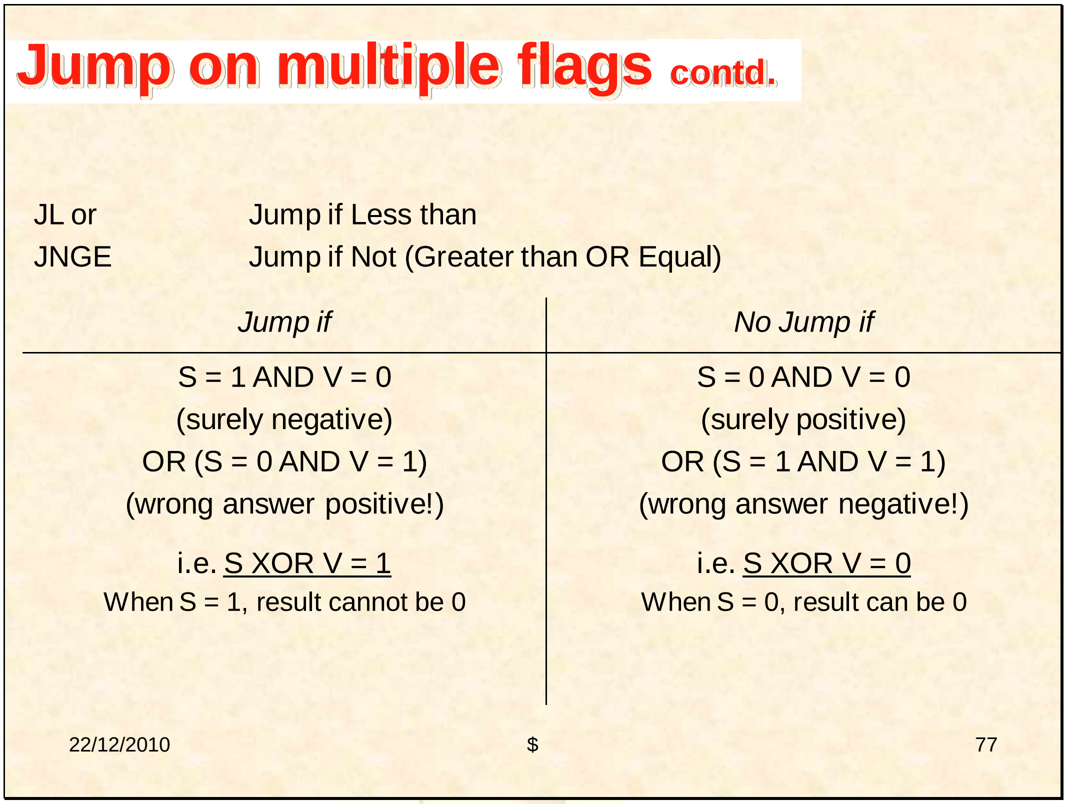


Jump on multiple flags contd.

JNLE or Jump if Not (Less than OR Equal)
JG Jump if Greater than

Jump if	*No Jump if*
S = 0 AND V = 0 (surely positive) OR (S = 1 AND V = 1) (wrong answer negative!) AND Z = 0 (not equal) i.e. S XOR V = 0 AND Z = 0	S = 1 AND V = 0 (surely negative) OR (S = 0 AND V = 1) (wrong answer positive!) OR Z = 1 (equal) i.e. S XOR V = 1 OR Z = 1

22/12/2010 $ 76



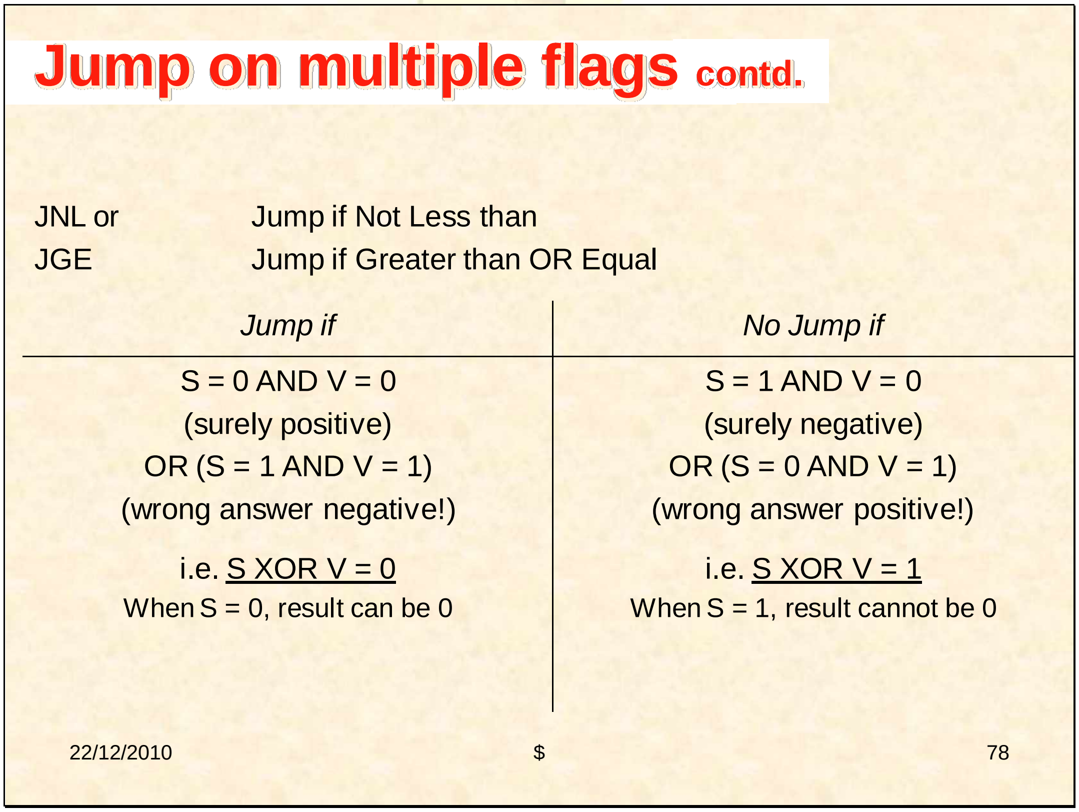
Jump on multiple flags contd.

JL or Jump if Less than

JNGE Jump if Not (Greater than OR Equal)

Jump if	*No Jump if*
S = 1 AND V = 0	S = 0 AND V = 0
(surely negative)	(surely positive)
OR (S = 0 AND V = 1)	OR (S = 1 AND V = 1)
(wrong answer positive!)	(wrong answer negative!)
i.e. S XOR V = 1	i.e. S XOR V = 0
When S = 1, result cannot be 0	When S = 0, result can be 0

22/12/2010 $ 77



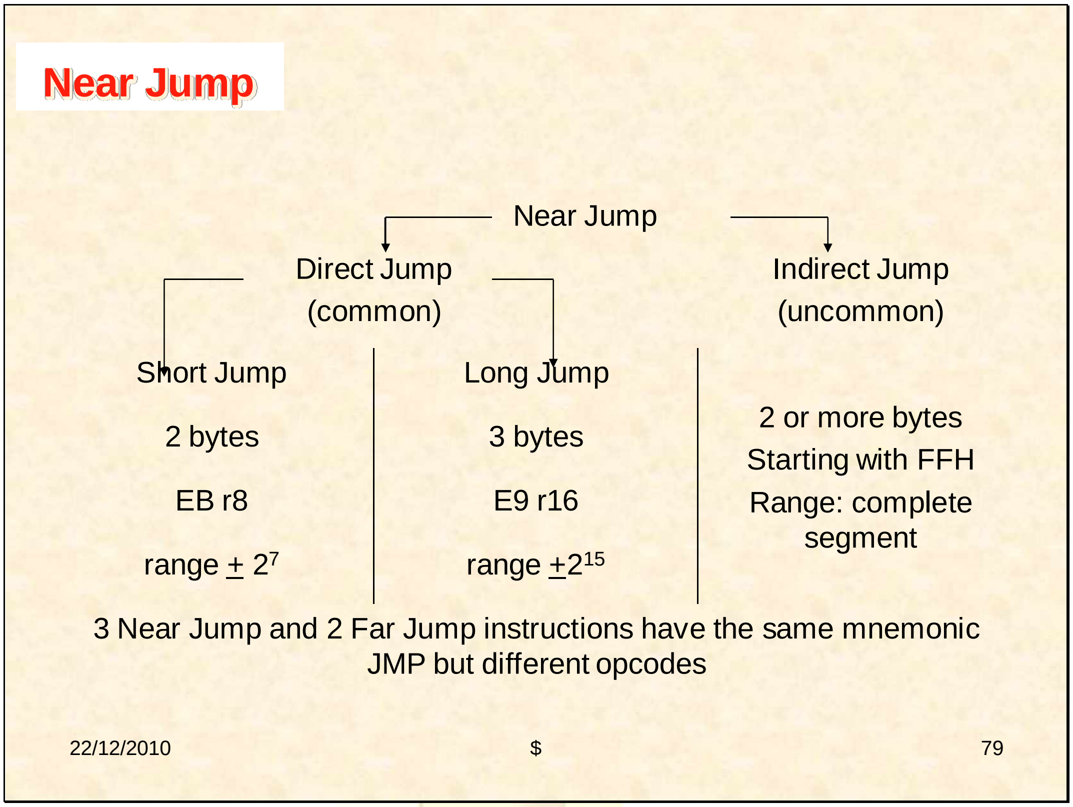
Jump on multiple flags contd.

JNL or Jump if Not Less than

JGE Jump if Greater than OR Equal

Jump if	*No Jump if*
S = 0 AND V = 0	S = 1 AND V = 0
(surely positive)	(surely negative)
OR (S = 1 AND V = 1)	OR (S = 0 AND V = 1)
(wrong answer negative!)	(wrong answer positive!)
i.e. S XOR V = 0	i.e. S XOR V = 1
When S = 0, result can be 0	When S = 1, result cannot be 0

22/12/2010 $ 78



Near Jump

Near Jump

Direct Jump
(common)

Indirect Jump
(uncommon)

Short Jump

2 bytes

EB r8

range ± 2^7

Long Jump

3 bytes

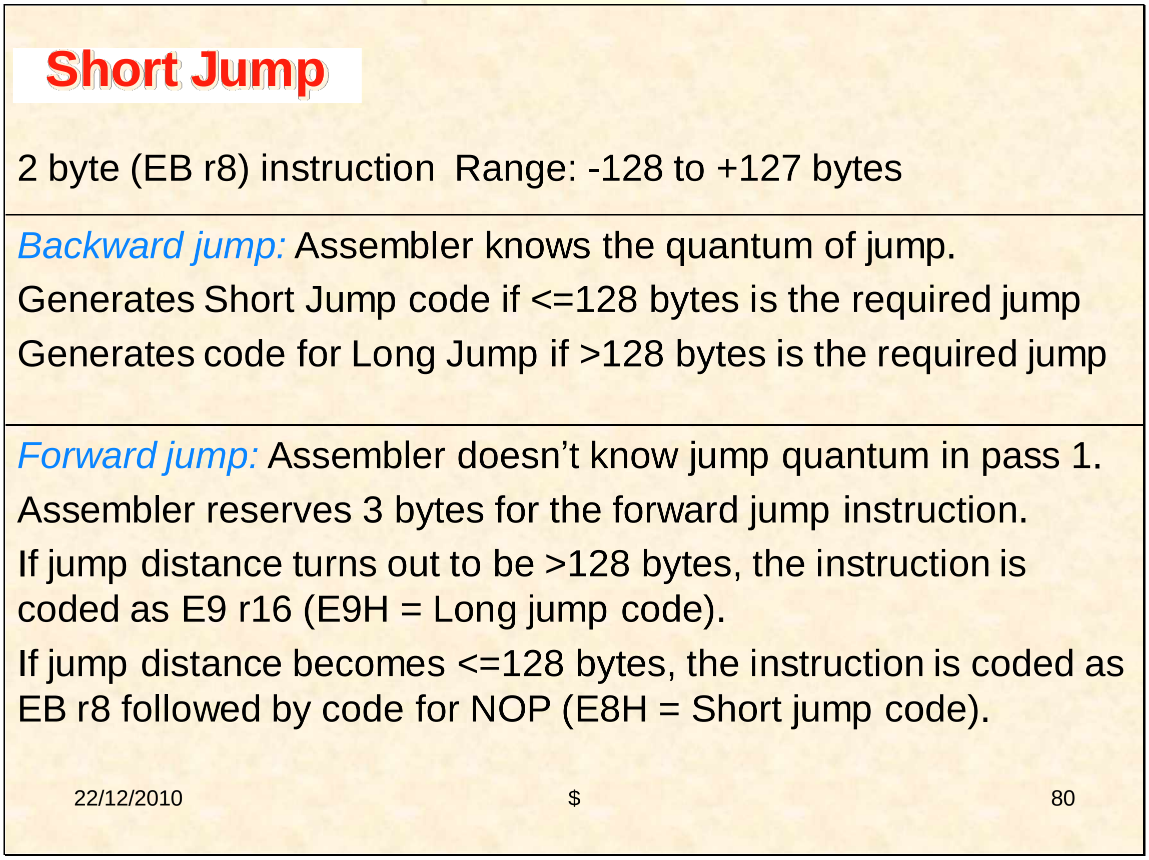
E9 r16

range ±2^{15}

2 or more bytes
Starting with FFH
Range: complete segment

3 Near Jump and 2 Far Jump instructions have the same mnemonic JMP but different opcodes

22/12/2010 $ 79



Short Jump

2 byte (EB r8) instruction Range: -128 to +127 bytes

Backward jump: Assembler knows the quantum of jump.

Generates Short Jump code if <=128 bytes is the required jump

Generates code for Long Jump if >128 bytes is the required jump

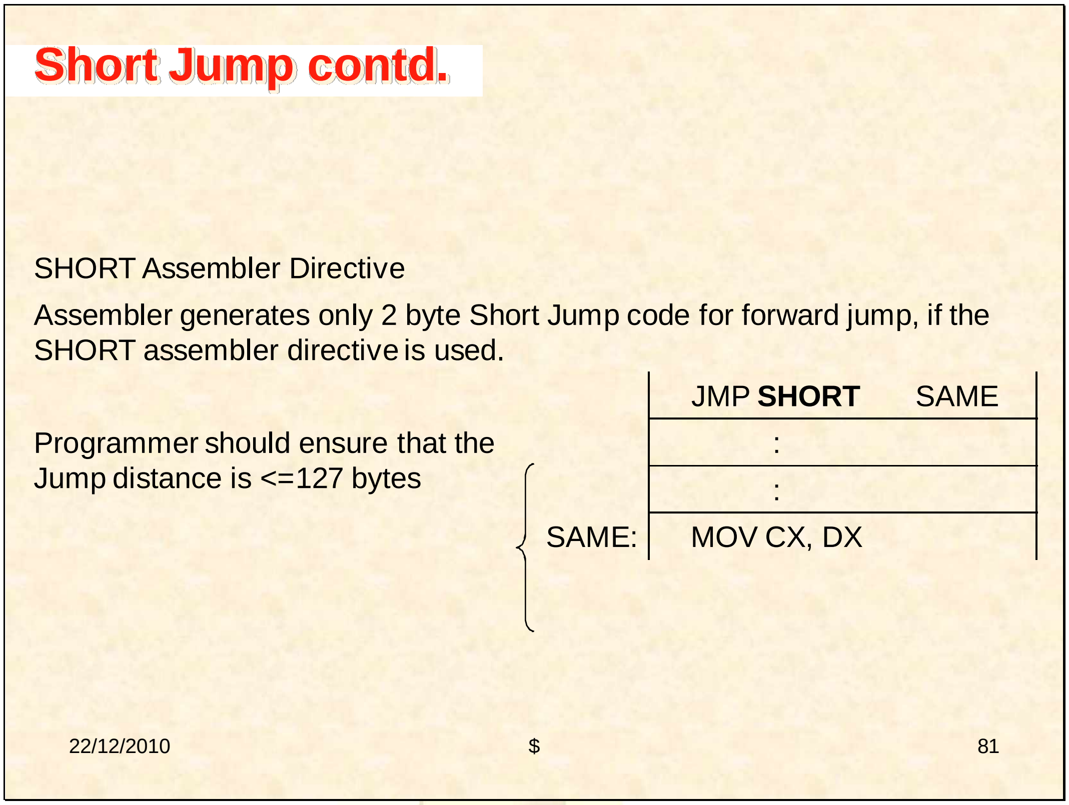
Forward jump: Assembler doesn't know jump quantum in pass 1.

Assembler reserves 3 bytes for the forward jump instruction.

If jump distance turns out to be >128 bytes, the instruction is coded as E9 r16 (E9H = Long jump code).

If jump distance becomes <=128 bytes, the instruction is coded as EB r8 followed by code for NOP (E8H = Short jump code).

22/12/2010 $ 80



Short Jump contd.

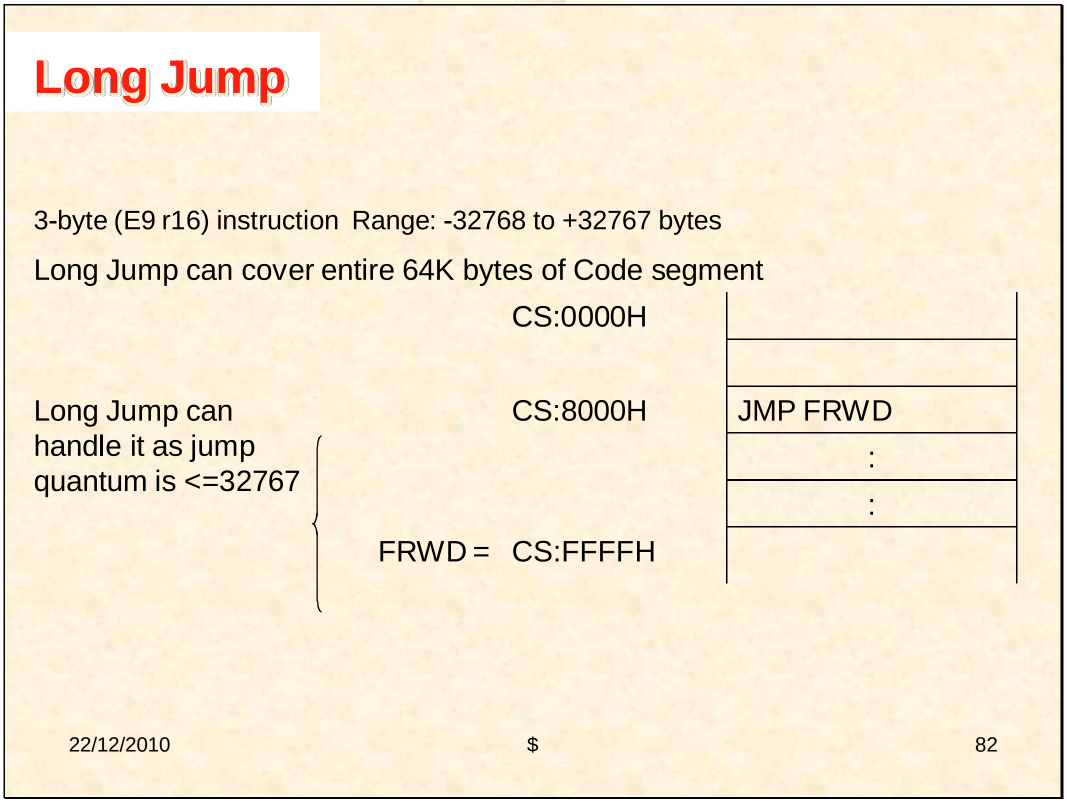
SHORT Assembler Directive

Assembler generates only 2 byte Short Jump code for forward jump, if the SHORT assembler directive is used.

Programmer should ensure that the Jump distance is <=127 bytes

```
        JMP SHORT     SAME
            :
            :
SAME:   MOV CX, DX
```

22/12/2010 $ 81



Long Jump

3-byte (E9 r16) instruction Range: -32768 to +32767 bytes

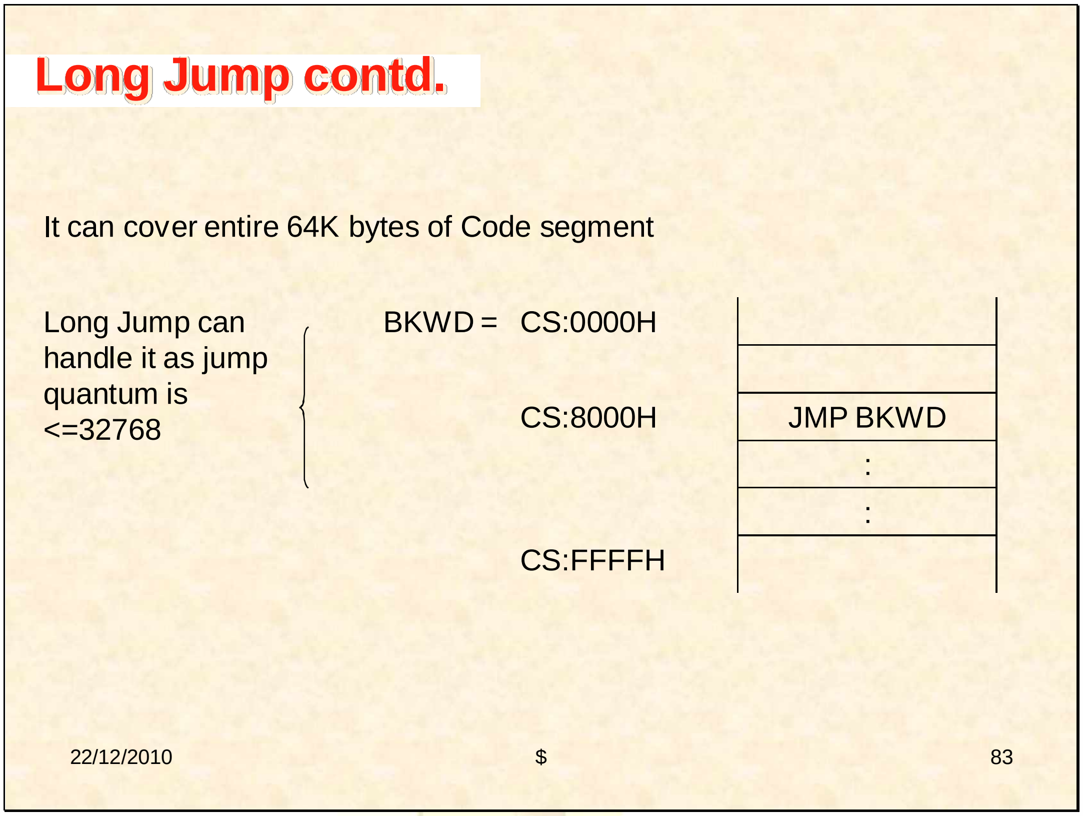
Long Jump can cover entire 64K bytes of Code segment

Long Jump can handle it as jump quantum is <=32767

```
         CS:0000H

         CS:8000H    JMP FRWD
                         :
                         :
FRWD =   CS:FFFFH
```

22/12/2010 $ 82



Long Jump contd.

It can cover entire 64K bytes of Code segment

Long Jump can handle it as jump quantum is <=32768

BKWD = CS:0000H

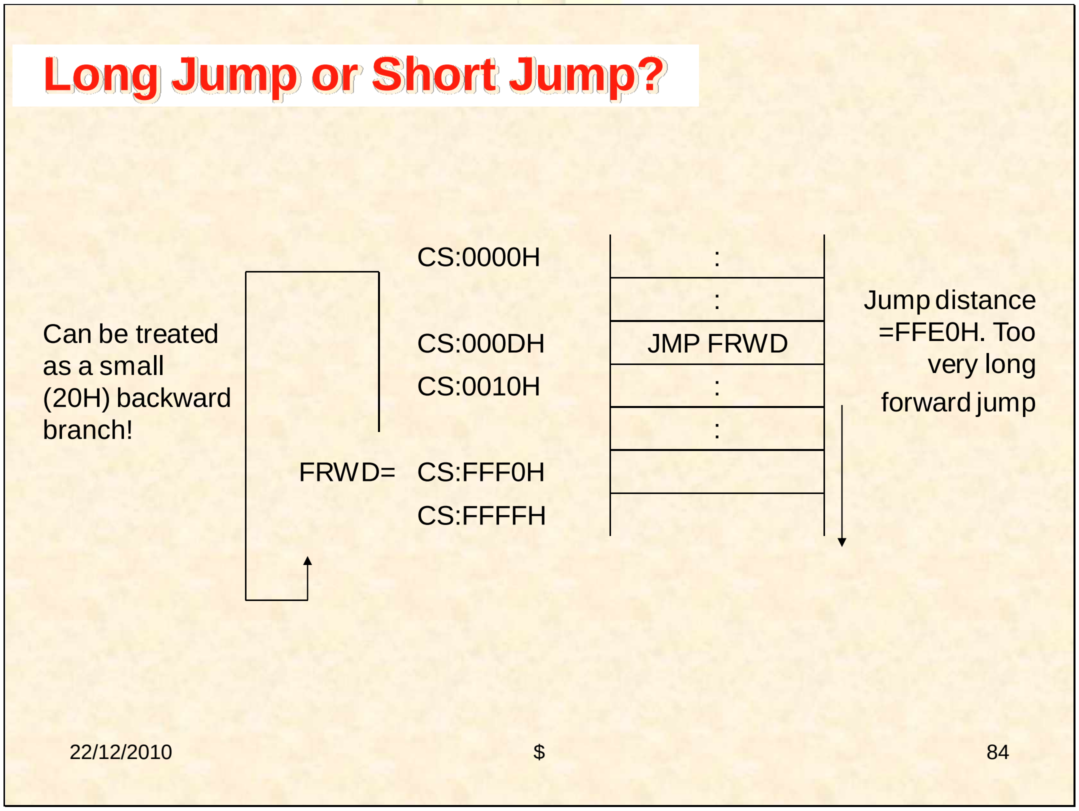
CS:8000H — JMP BKWD

:

:

CS:FFFFH

22/12/2010 $ 83



Long Jump or Short Jump?

Can be treated as a small (20H) backward branch!

CS:0000H — :

:

CS:000DH — JMP FRWD

CS:0010H — :

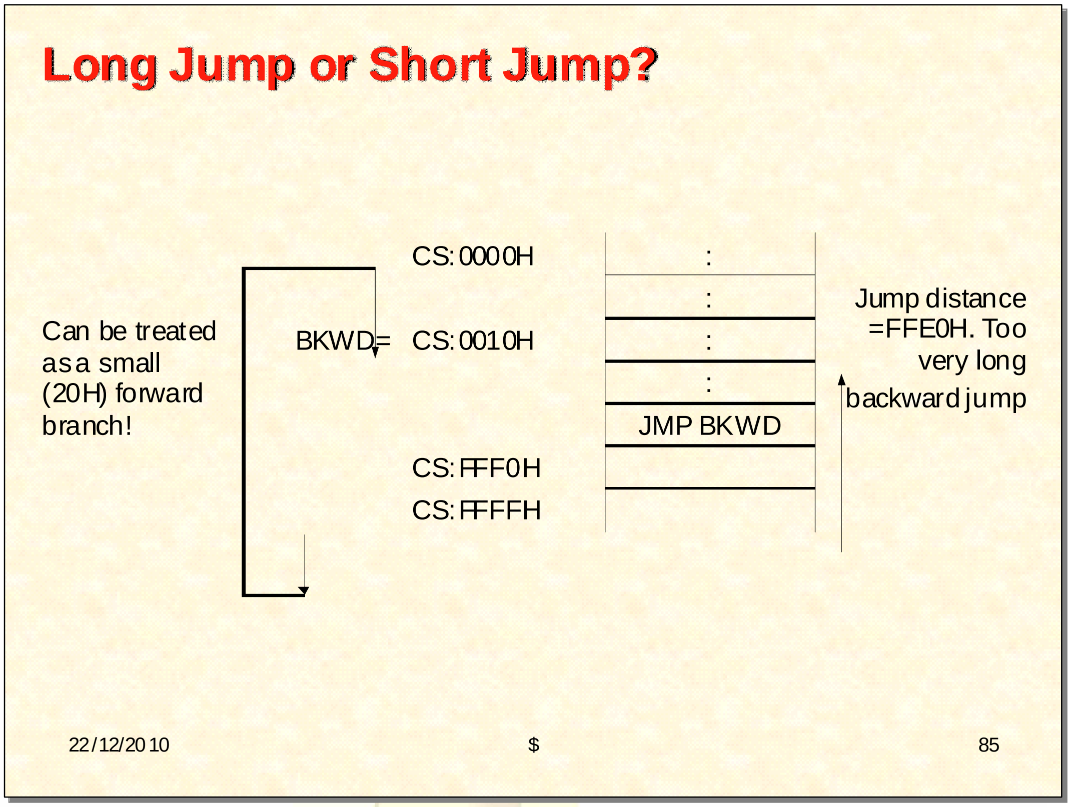
:

FRWD= CS:FFF0H

CS:FFFFH

Jump distance =FFE0H. Too very long forward jump

22/12/2010 $ 84



Long Jump or Short Jump?

Can be treated as a small (20H) forward branch!

BKWD= CS:0010H

CS:0000H

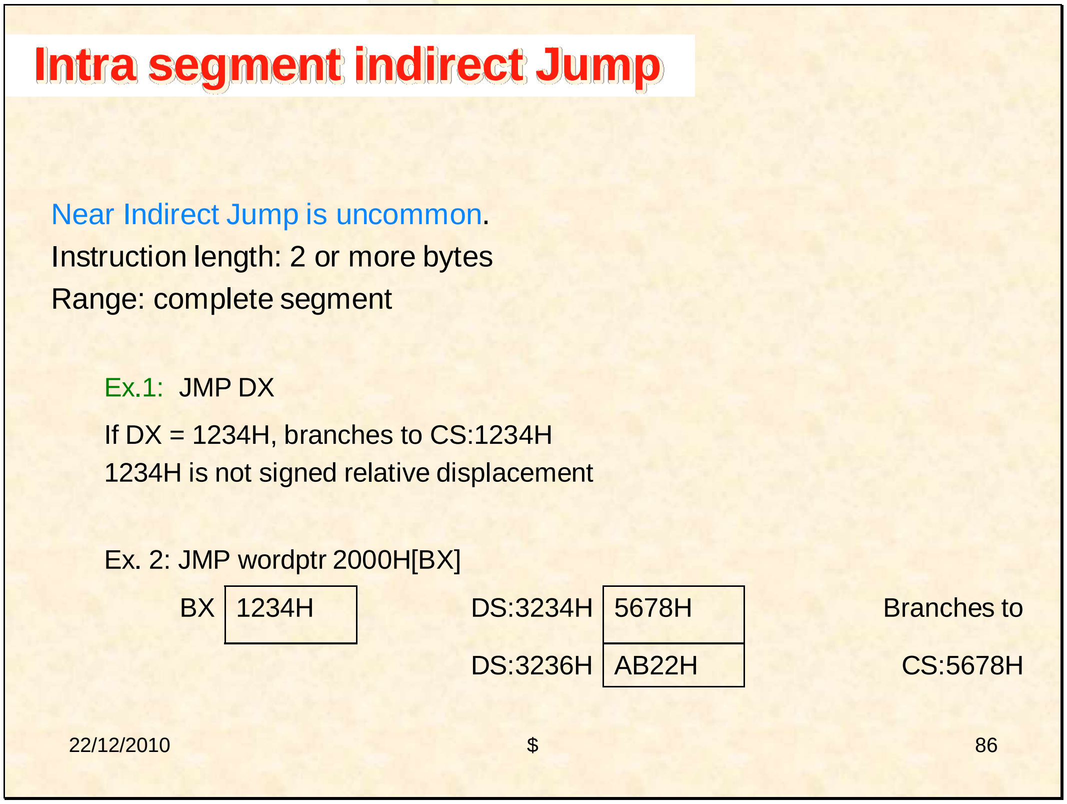
CS:FFF0H

CS:FFFFH

:
:
:
:
JMP BKWD

Jump distance =FFE0H. Too very long backward jump

22/12/2010 $ 85



Intra segment indirect Jump

Near Indirect Jump is uncommon.

Instruction length: 2 or more bytes

Range: complete segment

Ex.1: JMP DX

If DX = 1234H, branches to CS:1234H

1234H is not signed relative displacement

Ex. 2: JMP wordptr 2000H[BX]

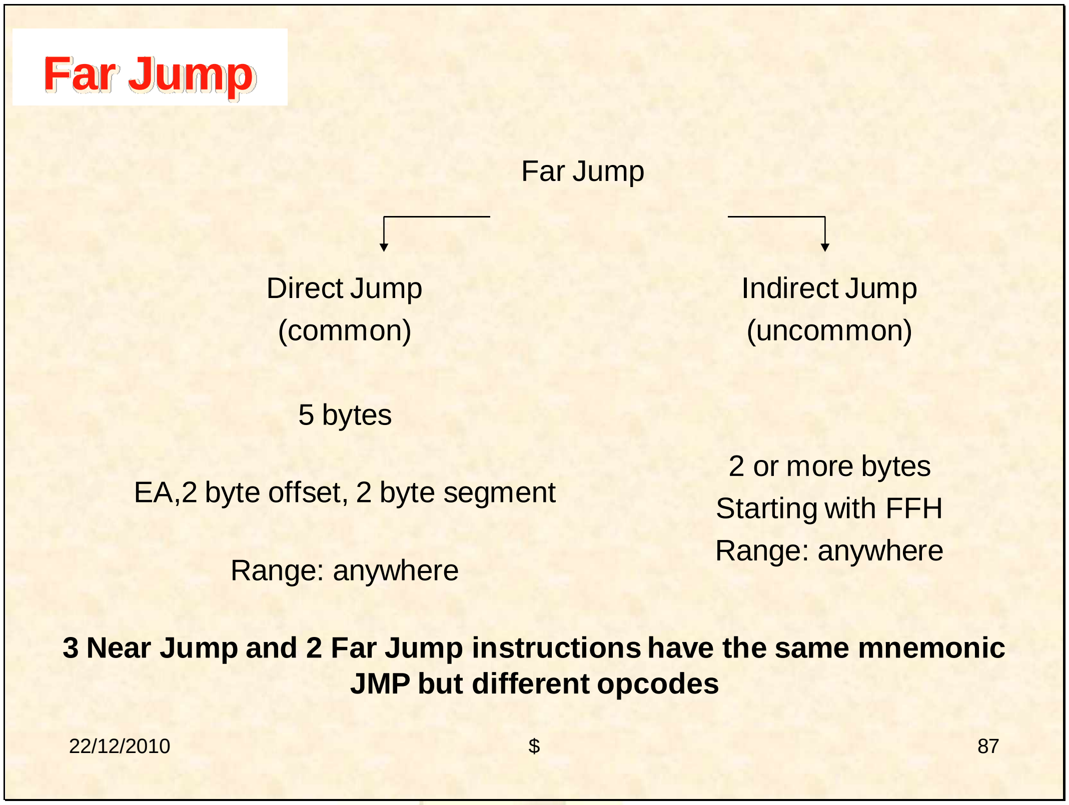
BX 1234H

DS:3234H 5678H

DS:3236H AB22H

Branches to CS:5678H

22/12/2010 $ 86



Far Jump

Far Jump

Direct Jump
(common)

5 bytes

EA,2 byte offset, 2 byte segment

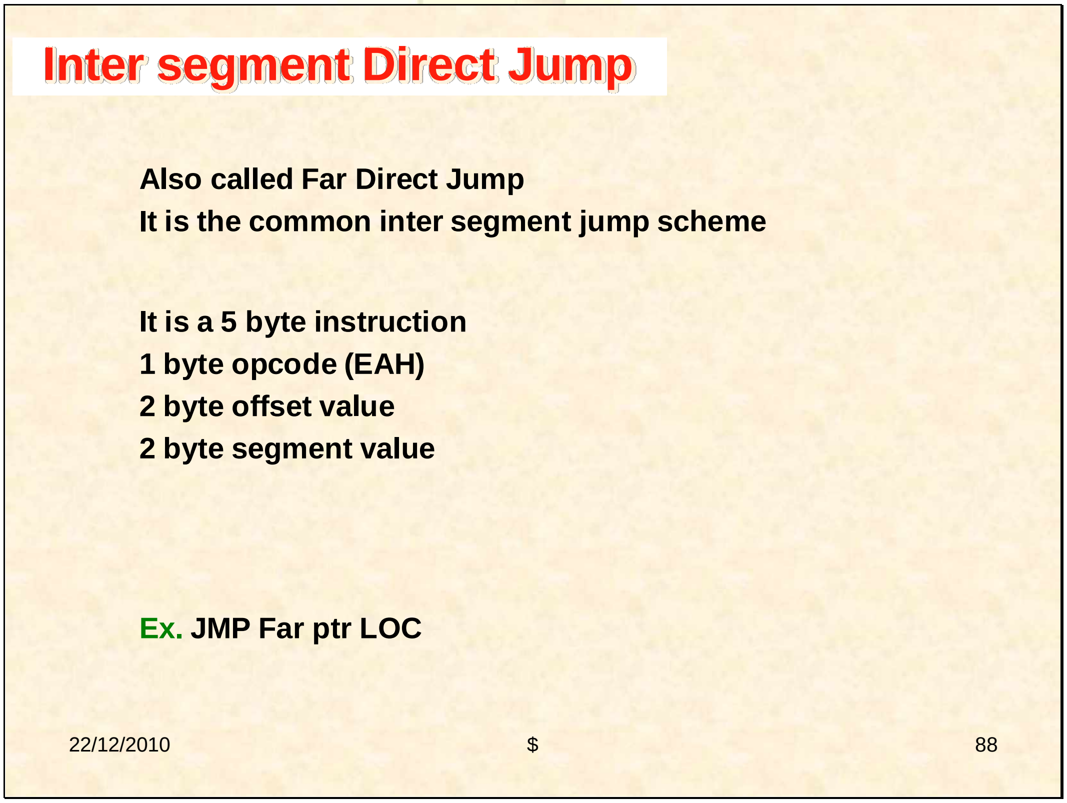
Range: anywhere

Indirect Jump
(uncommon)

2 or more bytes
Starting with FFH
Range: anywhere

3 Near Jump and 2 Far Jump instructions have the same mnemonic JMP but different opcodes

22/12/2010 $ 87



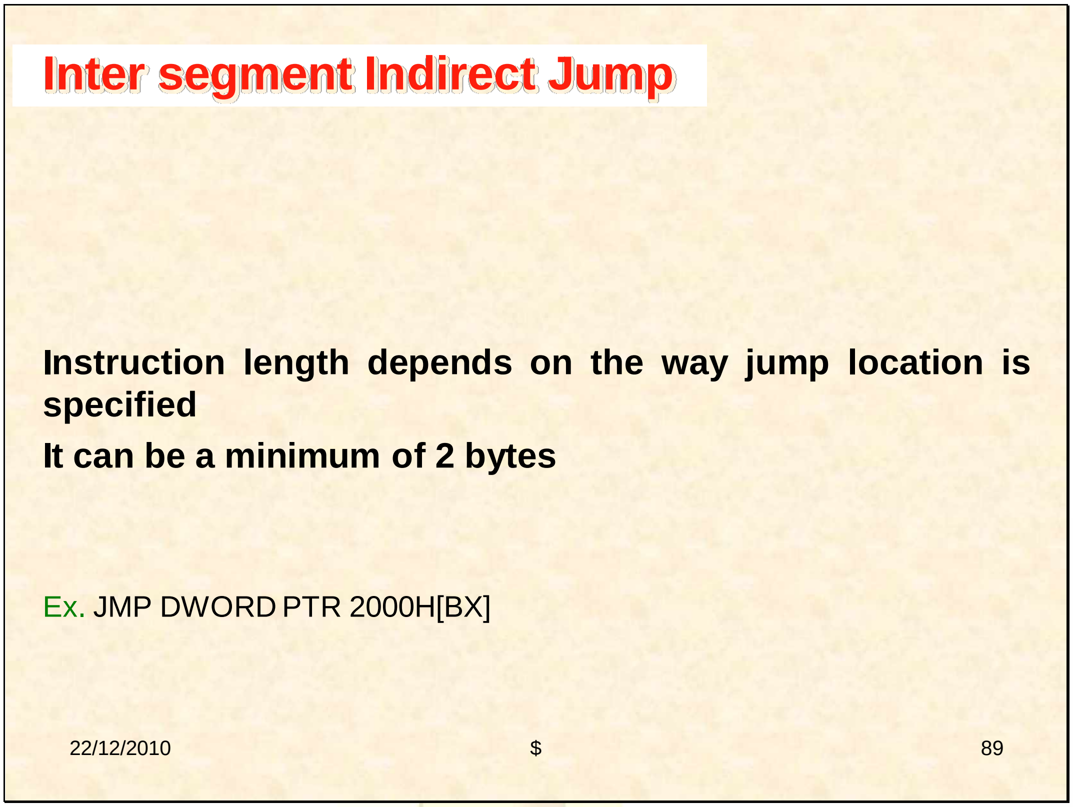
Inter segment Direct Jump

Also called Far Direct Jump
It is the common inter segment jump scheme

It is a 5 byte instruction
1 byte opcode (EAH)
2 byte offset value
2 byte segment value

Ex. JMP Far ptr LOC

22/12/2010 $ 88



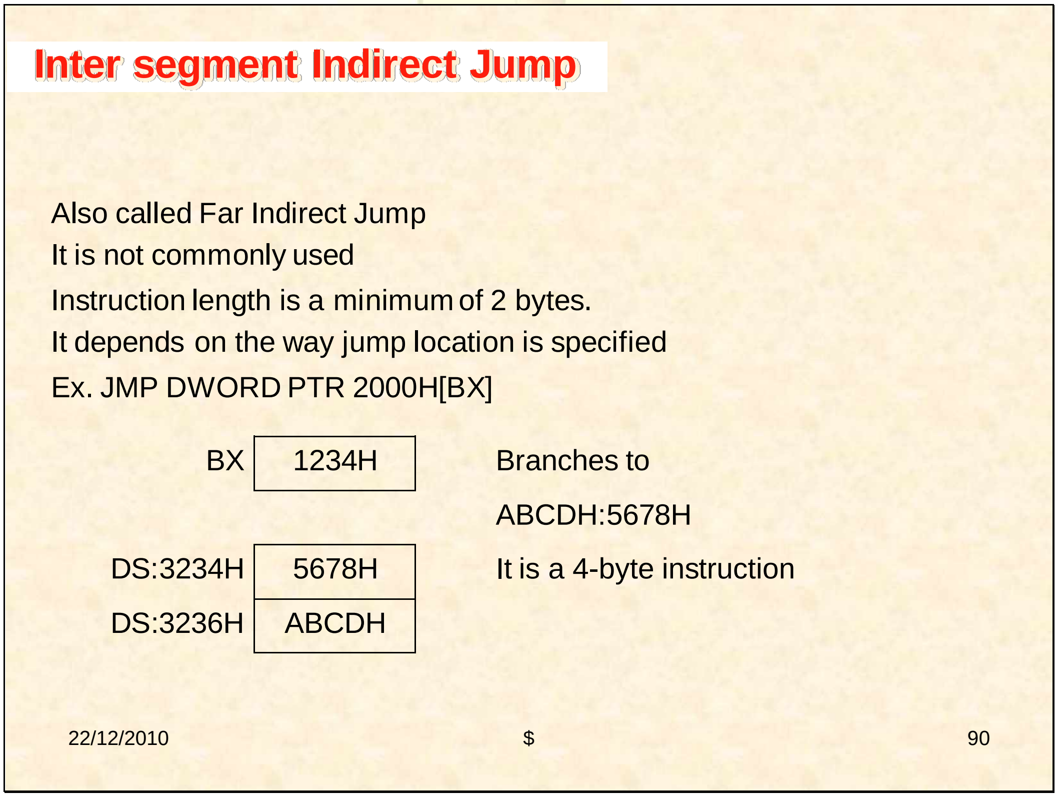
Inter segment Indirect Jump

Instruction length depends on the way jump location is specified

It can be a minimum of 2 bytes

Ex. JMP DWORD PTR 2000H[BX]

22/12/2010 $ 89



Inter segment Indirect Jump

Also called Far Indirect Jump

It is not commonly used

Instruction length is a minimum of 2 bytes.

It depends on the way jump location is specified

Ex. JMP DWORD PTR 2000H[BX]

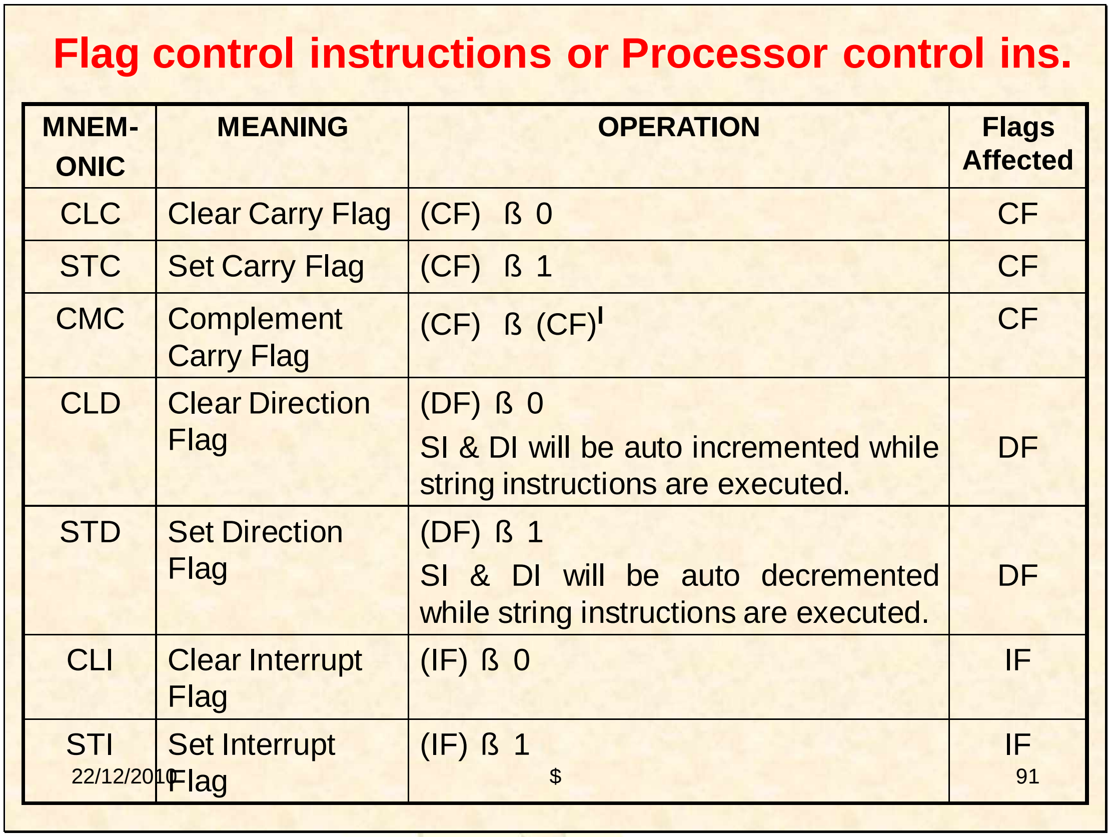
BX	1234H
DS:3234H	5678H
DS:3236H	ABCDH

Branches to

ABCDH:5678H

It is a 4-byte instruction

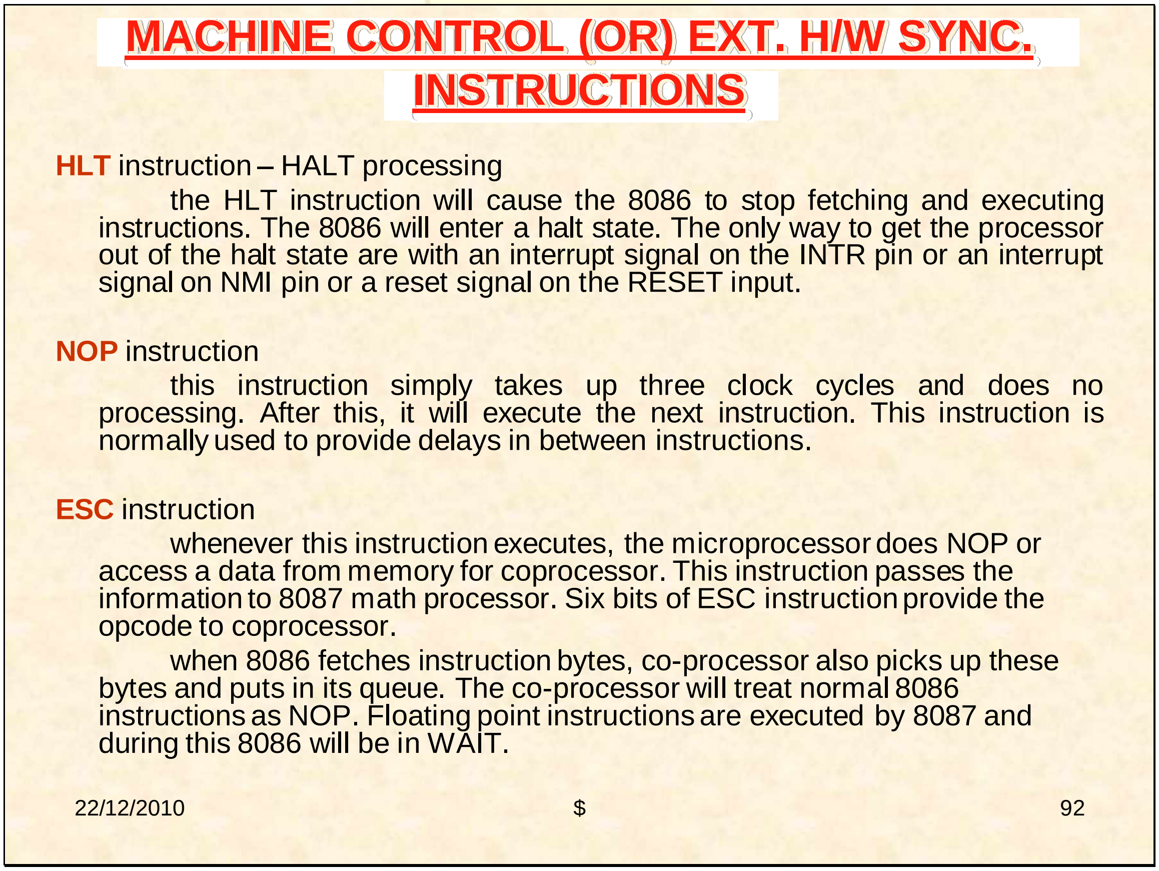
22/12/2010 $ 90



Flag control instructions or Processor control ins.

MNEM-ONIC	MEANING	OPERATION	Flags Affected
CLC	Clear Carry Flag	(CF) ß 0	CF
STC	Set Carry Flag	(CF) ß 1	CF
CMC	Complement Carry Flag	(CF) ß (CF)^{I}	CF
CLD	Clear Direction Flag	(DF) ß 0 SI & DI will be auto incremented while string instructions are executed.	DF
STD	Set Direction Flag	(DF) ß 1 SI & DI will be auto decremented while string instructions are executed.	DF
CLI	Clear Interrupt Flag	(IF) ß 0	IF
STI	Set Interrupt Flag	(IF) ß 1	IF

22/12/2010 $ 91



MACHINE CONTROL (OR) EXT. H/W SYNC. INSTRUCTIONS

HLT instruction – HALT processing

the HLT instruction will cause the 8086 to stop fetching and executing instructions. The 8086 will enter a halt state. The only way to get the processor out of the halt state are with an interrupt signal on the INTR pin or an interrupt signal on NMI pin or a reset signal on the RESET input.

NOP instruction

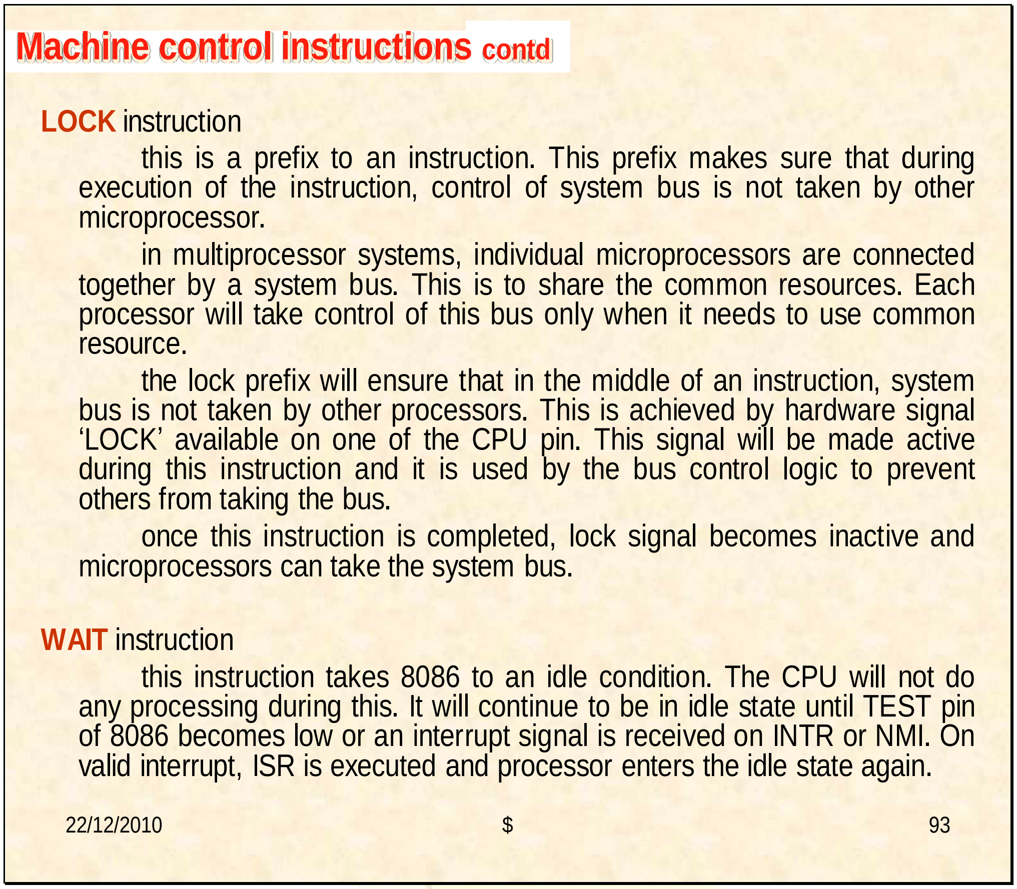
this instruction simply takes up three clock cycles and does no processing. After this, it will execute the next instruction. This instruction is normally used to provide delays in between instructions.

ESC instruction

whenever this instruction executes, the microprocessor does NOP or access a data from memory for coprocessor. This instruction passes the information to 8087 math processor. Six bits of ESC instruction provide the opcode to coprocessor.

when 8086 fetches instruction bytes, co-processor also picks up these bytes and puts in its queue. The co-processor will treat normal 8086 instructions as NOP. Floating point instructions are executed by 8087 and during this 8086 will be in WAIT.

22/12/2010 $ 92



Machine control instructions contd

LOCK instruction

this is a prefix to an instruction. This prefix makes sure that during execution of the instruction, control of system bus is not taken by other microprocessor.

in multiprocessor systems, individual microprocessors are connected together by a system bus. This is to share the common resources. Each processor will take control of this bus only when it needs to use common resource.

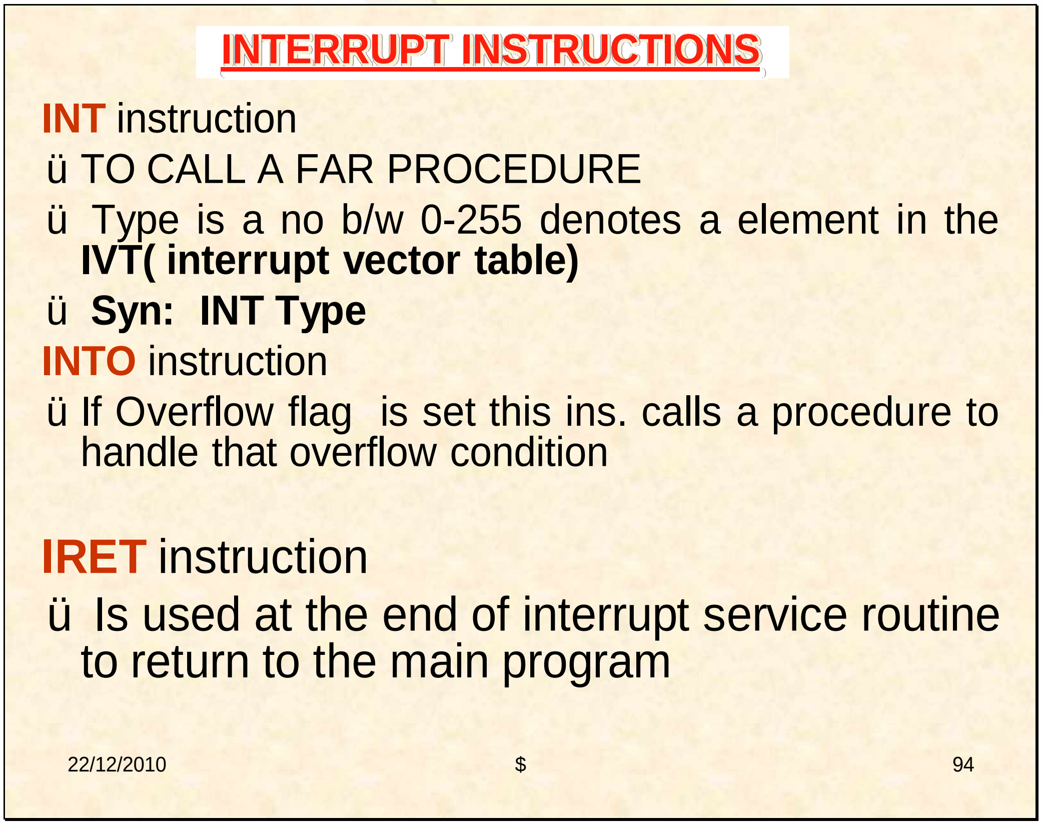
the lock prefix will ensure that in the middle of an instruction, system bus is not taken by other processors. This is achieved by hardware signal 'LOCK' available on one of the CPU pin. This signal will be made active during this instruction and it is used by the bus control logic to prevent others from taking the bus.

once this instruction is completed, lock signal becomes inactive and microprocessors can take the system bus.

WAIT instruction

this instruction takes 8086 to an idle condition. The CPU will not do any processing during this. It will continue to be in idle state until TEST pin of 8086 becomes low or an interrupt signal is received on INTR or NMI. On valid interrupt, ISR is executed and processor enters the idle state again.

22/12/2010 $ 93



INTERRUPT INSTRUCTIONS

INT instruction

ü TO CALL A FAR PROCEDURE

ü Type is a no b/w 0-255 denotes a element in the **IVT(interrupt vector table)**

ü **Syn: INT Type**

INTO instruction

ü If Overflow flag is set this ins. calls a procedure to handle that overflow condition

IRET instruction

ü Is used at the end of interrupt service routine to return to the main program

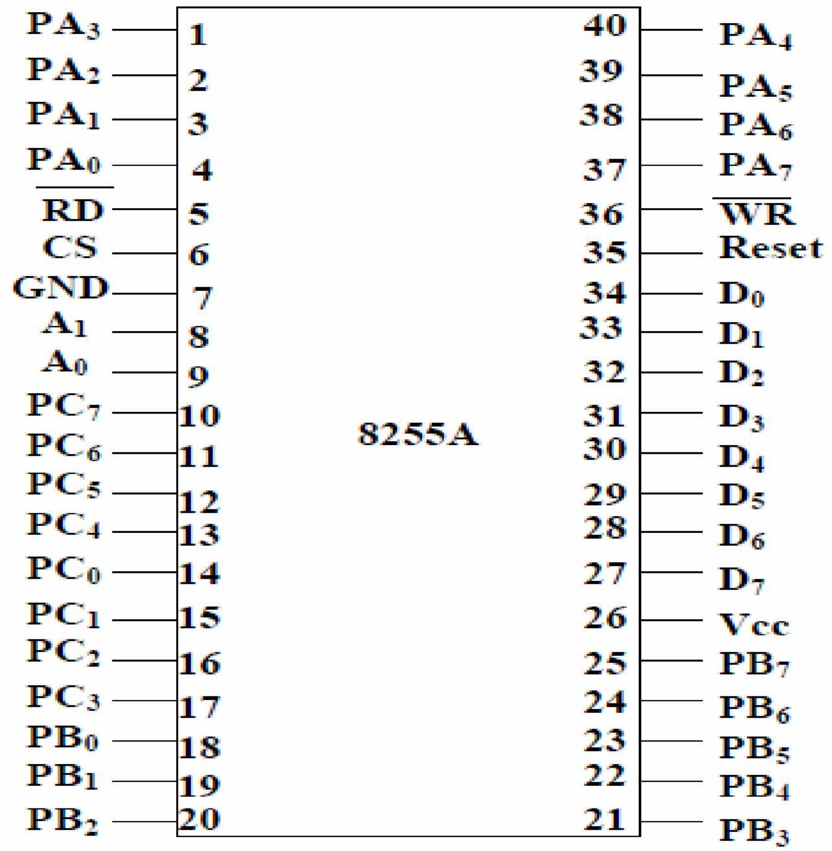
22/12/2010 $ 94

CHAPTER - 4

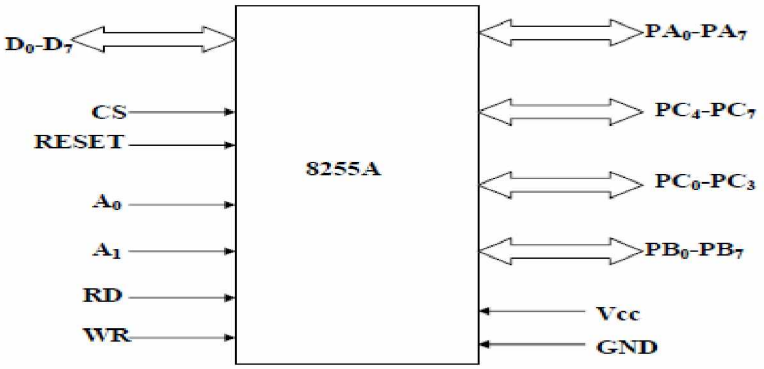
PROGRAMMABLE PERIPHERAL INTERFACE (8255)

- The parallel input-output port chip 8255 is also called as programmable ***peripheral input-output port.*** The Intel’s 8255 is designed for use with Intel’s 8-bit, 16-bit and higher capability microprocessors. It has 24 input/output lines which may be individually programmed in two groups of twelve lines each, or three groups of eight lines.
- The two groups of I/O pins are named as Group A and Group B. Each of these two groups contains a subgroup of eight I/O lines called as 8-bit port and another subgroup of four lines or a 4-bit port. Thus Group A contains an 8-bit port A along with a 4-bit port C upper.
- The port A lines are identified by symbols PA0-PA7 while the port C lines are identified as PC4-PC7. Similarly, Group B contains an 8-bit port B, containing lines PB0-PB7 and a 4-bit port C with lower bits PC0- PC3. The port C upper and port C lower can be used in combination as an 8-bit port C.
- Both the port C are assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit ports from 8255. All of these ports can function independently either as input or as output ports. This can be achieved by programming the bits of an internal register of 8255 called as control word register (CWR).
- The internal block diagram and the pin configuration of 8255 are shown in fig.
- The 8-bit data bus buffer is controlled by the read/write control logic. The read/write control logic manages all of the internal and external transfers of both data and control words. RD , WR , A1, A0 and RESET are the inputs provided by the microprocessor to the READ/ WRITE control logic of 8255. The 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus.
- This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor. The control words or status information is also transferred through the buffer.
- **The signal description of 8255 are briefly presented as follows :**
- **PA7-PA0**: These are eight port A lines that acts as either latched output or buffered input lines depending upon the control word loaded into the control word register.
- **PC7-PC4** : Upper nibble of port C lines. They may act as either output latches or input buffers lines.
- This port also can be used for generation of handshake lines in mode 1 or mode 2.

- **PC3-PC0** : These are the lower port C lines, other details are the same as PC7-PC4 lines.
- **PB0-PB7** : These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.
- **RD** : This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.
- **WR** : This is an input line driven by the microprocessor. A low on this line indicates write operation.

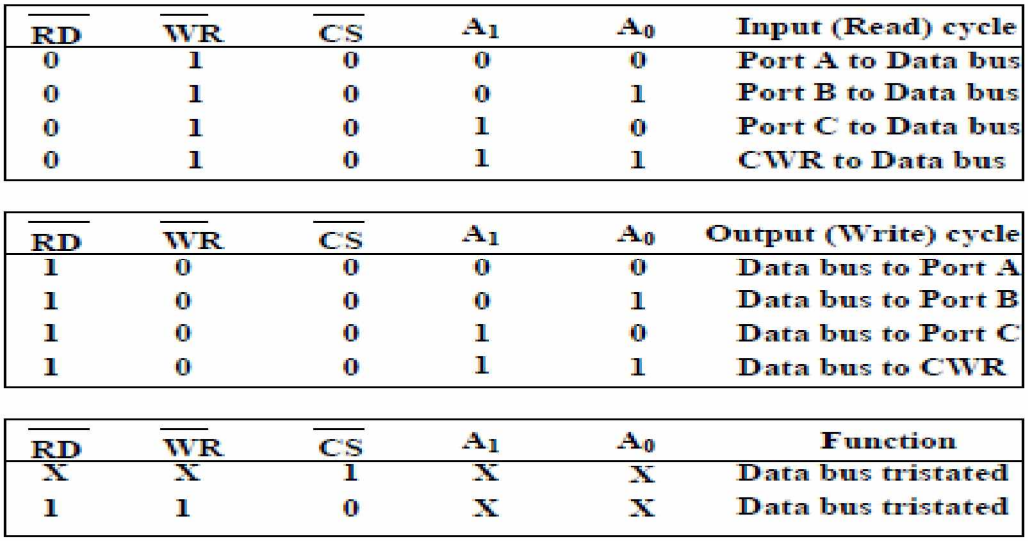


8255A Pin Configuration



Signals of 8255

- **CS** : This is a chip select line. If this line goes low, it enables the 8255 to respond to RD and WR signals, otherwise RD and WR signal are neglected.
- **A1-A0** : These are the address input lines and are driven by the microprocessor. These lines A1-A0 with RD , WR and CS from the following operations for 8255. These address lines are used for addressing any one of the four registers, i.e. three ports and a control word register as given in table below.
- In case of 8086 systems, if the 8255 is to be interfaced with lower order data bus, the A0 and A1 pins of 8255 are connected with A1 and A2 respectively.
- **D0-D7** : These are the data bus lines those carry data or control word to/from the microprocessor.
- **RESET** : A logic high on this line clears the control word register of 8255. All ports are set as input ports by default after reset.

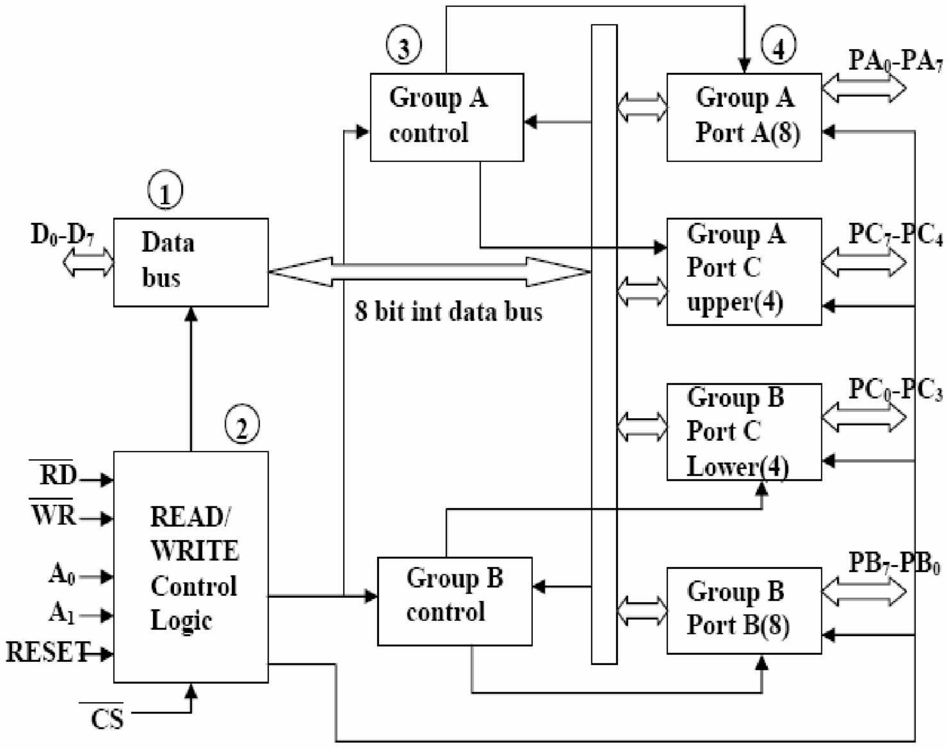


$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Input (Read) cycle
0	1	0	0	0	Port A to Data bus
0	1	0	0	1	Port B to Data bus
0	1	0	1	0	Port C to Data bus
0	1	0	1	1	CWR to Data bus

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Output (Write) cycle
1	0	0	0	0	Data bus to Port A
1	0	0	0	1	Data bus to Port B
1	0	0	1	0	Data bus to Port C
1	0	0	1	1	Data bus to CWR

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Function
X	X	1	X	X	Data bus tristated
1	1	0	X	X	Data bus tristated

Data Transfer Scheme using Control Lines



Block Diagram of 8255

Block Diagram of 8255 (Architecture)

- It has a 40 pins of 4 groups.

1. Data bus buffer
2. Read Write control logic
3. Group A and Group B controls
4. Port A, B and C

- ***Data bus buffer***: This is a tristate bidirectional buffer used to interface the 8255 to system databus. Data is transmitted or received by the buffer on execution of input or output instruction by the CPU.
- Control word and status information are also transferred through this unit.
- ***Read/Write control logic***: This unit accepts control signals (RD, WR) and also inputs from address bus and issues commands to individual group of control blocks (Group A, Group B).
- It has the following pins.

a) **CS – Chip select** : A low on this PIN enables the communication between CPU and 8255.

b) **RD (Read)** – A low on this pin enables the CPU to read the data in the ports or the status word through data bus buffer.

c) **WR (**Write) : A low on this pin, the CPU can write data on to the ports or on to the control register through the data bus buffer.

d) **RESET**: A high on this pin clears the control register and all ports are set to the input mode

e) **A0** and **A1** (Address pins): These pins in conjunction with RD and WR pins control the selection of one of the 3 ports.

• ***Group A and Group B controls*** : These block receive control from the CPU and issues commands to their respective ports.

• Group A - PA and PCU (PC7 –PC4)

• Group B - PCL (PC3 – PC0)

• Control word register can only be written into no read operation of the CW register is allowed.

• a) **Port A**: This has an 8 bit latched/buffered O/P and 8 bit input latch. It can be programmed in 3 modes – mode 0, mode 1, mode 2.

b) **Port B**: This has an 8 bit latched / buffered O/P and 8 bit input latch. It can be programmed in mode 0, mode1.

c) **Port C** : This has an 8 bit latched input buffer and 8 bit out put latched/buffer. This port can be divided into two 4 bit ports and can be used as control signals for port A and port B. it can be programmed in mode 0.

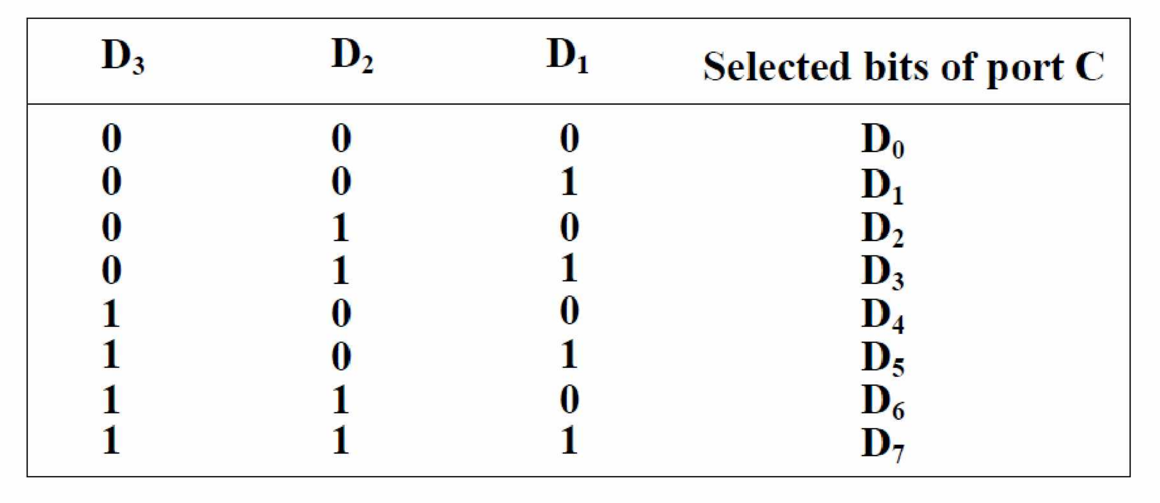
Modes of Operation of 8255

• These are two basic modes of operation of 8255. I/O mode and Bit Set-Reset mode (BSR).

• In I/O mode, the 8255 ports work as programmable I/O ports, while in BSR mode only port C (PC0-PC7) can be used to set or reset its individual port bits.

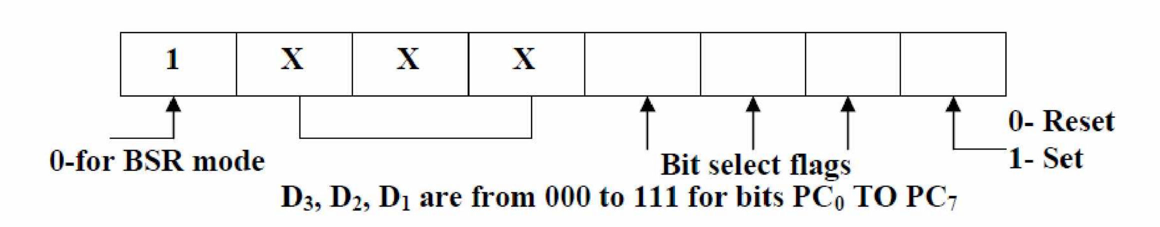
• Under the I/O mode of operation, further there are three modes of operation of 8255, so as to support different types of applications, mode 0, mode 1 and mode 2.

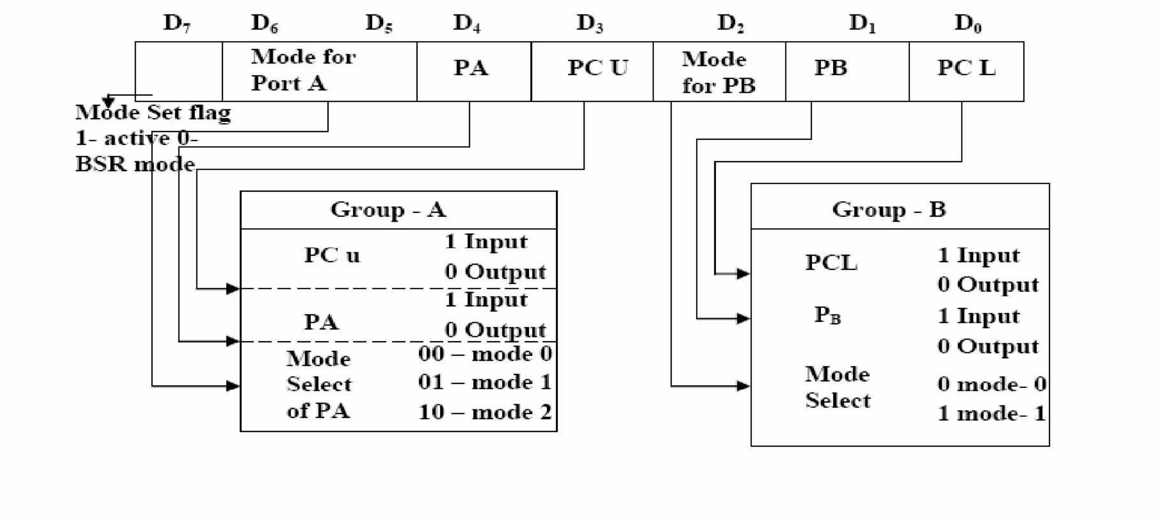
• ***BSR Mode***: In this mode any of the 8-bits of port C can be set or reset depending on D0 of the control word. The bit to be set or reset is selected by bit select flags D3, D2 and D1 of the CWR as given in table.



D_3	D_2	D_1	Selected bits of port C
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

BSR Mode: CWR Format





Control Word Format of 8255

• **I/O Modes:**

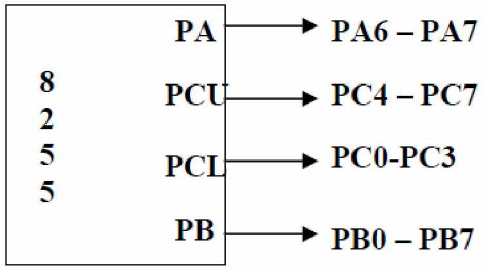
a) ***Mode 0 (Basic I/O mode):*** This mode is also called as basic input/output mode. This mode provides simple input and output capabilities using each of the three ports. Data can be simply read from and written to the input and output ports respectively, after appropriate initialization

• The salient features of this mode are as listed below:

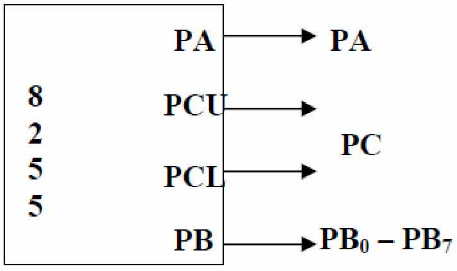
1. Two 8-bit ports (port A and port B)and two 4-bit ports (port C upper and lower) are available. The two 4-bit ports can be combinedly used as a third 8-bit port.
2. Any port can be used as an input or output port.
3. Output ports are latched. Input ports are not latched.
4. A maximum of four ports are available so that overall 16 I/O configuration are possible.

• All these modes can be selected by programming a register internal to 8255 known as CWR.

• The control word register has two formats. The first format is valid for I/O modes of operation, i.e. modes 0, mode 1 and mode 2 while the second format is valid for bit set/reset (BSR) mode of operation. These formats are shown in following fig.



All Output



Port A and Port C acting as O/P. Port B acting as I/P

b) Mode 1: (Strobed input/output mode): In this mode the handshaking control the input and output action of the specified port. Port C lines PC0-PC2, provide strobe or handshake lines for port B. This group which includes port B and PC0-PC2 is called as group B for Strobed data input/output. Port C lines PC3-PC5 provide strobe lines for port A.

This group including port A and PC3-PC5 from group A. Thus port C is utilized for generating handshake signals. The salient features of mode 1 are listed as follows:

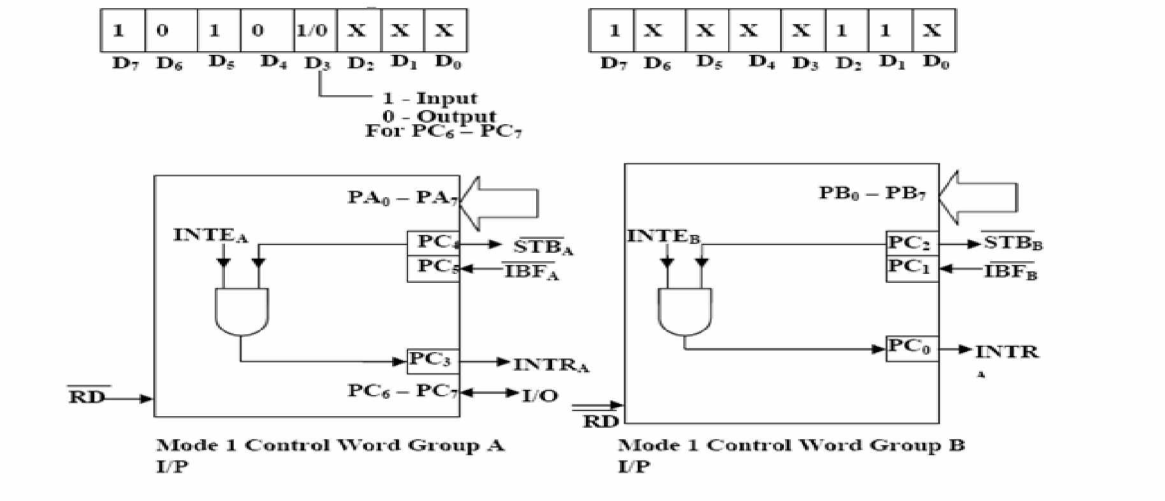
1. Two groups – group A and group B are available for strobed data transfer.
2. Each group contains one 8-bit data I/O port and one 4-bit control/data port.
3. The 8-bit data port can be either used as input and output port. The inputs and outputs both are latched.
4. Out of 8-bit port C, PC0-PC2 are used to generate control signals for port B and PC3-PC5 are used to generate control signals for port A. the lines PC6, PC7 may be used as independent data lines.

- **The control signals for both the groups in input and output modes are explained as follows:**

Input control signal definitions (mode 1):

- **STB(Strobe input)** – If this lines falls to logic low level, the data available at 8-bit input port is loaded into input latches.
- **IBF (Input buffer full)** – If this signal rises to logic 1, it indicates that data has been loaded into latches, i.e. it works as an acknowledgement. IBF is set by a low on STB and is reset by the rising edge of RD input.
- **INTR (Interrupt request)** – This active high output signal can be used to interrupt the CPU whenever an input device requests the service. INTR is set by a high STB pin and a high at IBF pin. INTE is an internal flag that can be controlled by the bit set/reset mode of either PC4 (INTEA) or PC2 (INTEB) as shown in fig. • **INTR** is reset by a falling edge of RD input. Thus an external input device can be request the service of the processor by putting the data on the bus and sending the strobe signal.

Input control signal definitions in Mode 1

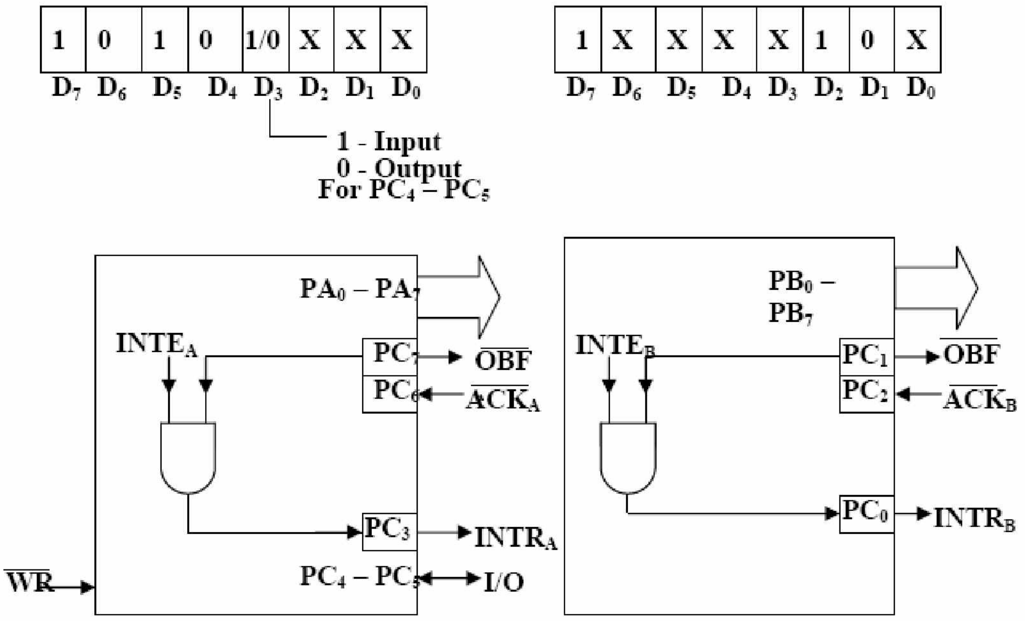


Output control signal definitions (mode 1) :

- **OBF** (Output buffer full) – This status signal, whenever falls to low, indicates that CPU has written data to the specified output port. The OBF flip-flop will be set by a rising edge of WR signal and reset by a low going edge at the ACK input.
- ACK (Acknowledge input) – ACK signal acts as an acknowledgement to be given by an output device. ACK signal, whenever low, informs the CPU that the data transferred by the CPU to the output device through the port is received by the output device.
- **INTR** (Interrupt request) – Thus an output signal that can be used to interrupt the CPU when an output device acknowledges the data received from the CPU. INTR is set when ACK, OBF and INTE are 1. It is reset by a falling edge on WR input. The

INTEA and INTEB flags are controlled by the bit set-reset mode of PC6 and PC2 respectively.

Output control signal definitions Mode 1



1	0	1	0	1/0	X	X	X
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

1	X	X	X	X	1	0	X
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Mode 1 Control Word Group A **Mode 1 Control Word Group B**

Mode 2 *(Strobed bidirectional I/O):*

This mode of operation of 8255 is also called as strobed bidirectional I/O. This mode of operation provides 8255 with an additional features for communicating with a peripheral device on an 8-bit data bus. Handshaking signals are provided to maintain proper data flow and synchronization between the data transmitter and receiver. The interrupt generation and other functions are similar to mode 1.

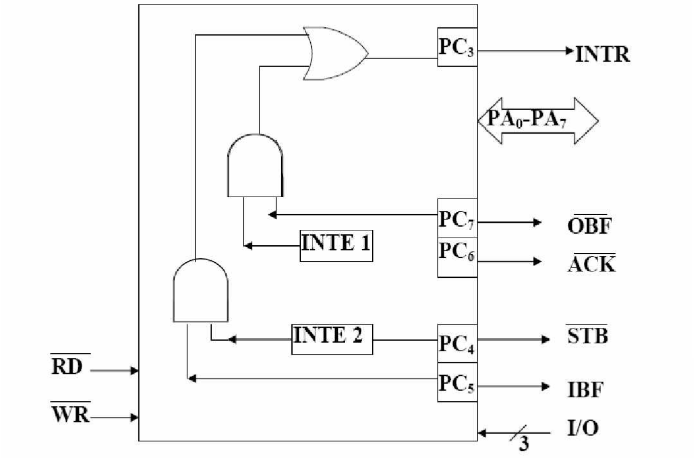
• In this mode, 8255 is a bidirectional 8-bit port with handshake signals. The Rd and WR signals decide whether the 8255 is going to operate as an input port or output port.

• The Salient features of Mode 2 of 8255 are listed as follows:

1. The single 8-bit port in group A is available.
2. The 8-bit port is bidirectional and additionally a 5-bit control port is available.
3. Three I/O lines are available at port C.(PC2 – PC0)
4. Inputs and outputs are both latched.
5. The 5-bit control port C (PC3-PC7) is used for generating / accepting handshake signals for the 8-bit data transfer on port A.

• ***Control signal definitions in mode 2***:

- **INTR** – (Interrupt request) As in mode 1, this control signal is active high and is used to interrupt the microprocessor to ask for transfer of the next data byte to/from it. This signal is used for input (read) as well as output (write) operations.



Mode 2 pin description

- ***Control Signals for Output operations***:
- **OBF (Output buffer full)** – This signal, when falls to low level, indicates that the CPU has written data to port A.

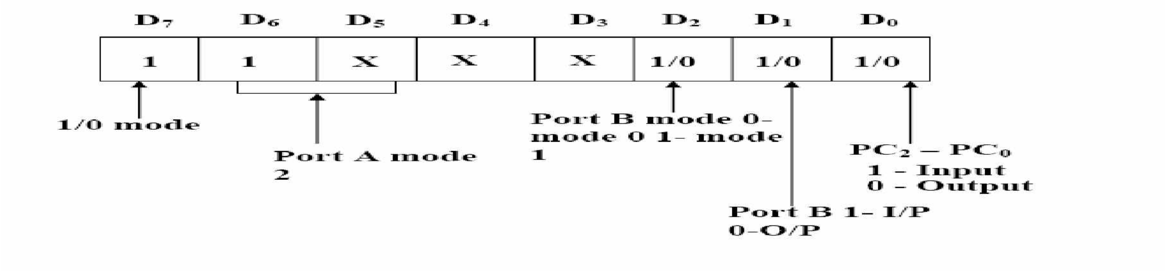
ACK (Acknowledge)- This control input, when falls to logic low level, acknowledges that the previous data byte is received by the destination and next byte may be sent by the processor. This signal enables the internal tristate buffers to send the next data byte on port A.

- **INTE1** (A flag associated with OBF)- This can be controlled by bit set/reset mode with PC6.
- ***Control signals for input operations:***
- **STB (Strobe input)-** A low on this line is used to strobe in the data into the input latches of 8255.
- **IBF (Input buffer full)-** When the data is loaded into input buffer, this signal rises to logic '1'. This can be used as an acknowledge that the data has been received by the receiver.
- Note: WR must occur before ACK and STB must be activated before RD

The following fig shows a schematic diagram containing an 8-bit bidirectional port, 5-bit control port and the relation of INTR with the control pins. Port B can either be set to Mode 0 or 1 with port A(Group A) is in Mode 2.

- **Mode 2 is not available for port B. The following fig shows the control word.**

- The INTR goes high only if either IBF, INTE2, STB and RD go high or OBF, INTE1, ACK and WR go high. The port C can be read to know the status of the peripheral device, in terms of the control signals, using the normal I/O instructions.



Mode-2 control word format

.

3.2 Interfacing I/O Devices

3.2.1 Keyboard Circuit Connections and Interfacing:

- In most keyboards, the keyswitches are connecting in a matrix of rows and columns, as shown in fig.
- We will use simple mechanical switches for our examples, but the principle is same for other type of switches.
- Getting meaningful data from a keyboard, it requires the following three major tasks:

1. Detect a keypress.
2. Debounce the keypress.
3. Encode the keypress

- Three tasks can be done with hardware, software, or a combination of two, depending on the application.

1. Software Keyboard Interfacing:

- ***Circuit connection and algorithm:*** The following fig (a) shows how a hexadecimal keypad can be connected to a couple of microcomputer ports so the three interfacing tasks can be done as part of a program.
- The rows of the matrix are connected to four output port lines. The column lines of matrix are connected to four input-port lines. To make the program simpler, the row lines are also connected to four input lines.
- When no keys are pressed, the column lines are held high by the pull-up resistor connected to +5V. Pressing a key connects a row to a column. If a low is output on a row and a key in that row is pressed, then the low will appear on the column which contains that key and can be detected on the input port.

- If you know the row and column of the pressed key, you then know which key was pressed, and you can convert this information into any code you want to represent that key.
- The following flow chart for a procedure to detect, debounce and produce the hex code for a pressed key.
- An easy way to detect if any key in the matrix is pressed is to output 0's to all rows and then check the column to see if a pressed key has connected a low to a column.
- In the algorithm we first output lows to all the rows and check the columns over and over until the column are all high. This is done before the previous key has been released before looking for the next one. In the standard keyboard terminology, this is called two-key lockout
- Once the columns are found to be all high, the program enters another loop, which waits until a low appears on one of the columns, indicating that a key has been pressed. This second loop does the detect task for us. A simple 20-ms delay procedure then does the debounce task.
- After the debounce time, another check is made to see if the key is still pressed. If the columns are now all high, then no key is pressed and the initial detection was caused by a noise pulse or a light brushing past a key. If any of the columns are still low, then the assumption is made that it was a valid keypress.
- The final task is to determine the row and column of the pressed key and convert this row and column information to the hex code for the pressed key. To get the row and column information, a low is output to one row and the column are read.
- If none of the columns is low, the pressed key is not in that row. So the low is rotated to the next row and the column are checked again. The process is repeated until a low on a row produces a low on one of the column.
- The pressed key then is in the row which is low at that time.
- The connection fig shows the byte read in from the input port will contain a 4-bit code which represents the row of the pressed key and a 4-bit code which represent the column of the pressed key.
- ***Error trapping:*** The concept of detecting some error condition such as " no match found" is called error trapping. Error trapping is a very important part of real programs. Even in simple programs, think what might happen with no error trap if two keys in the same row were pressed at exactly at the same time and a column code with two lows in it was produced.

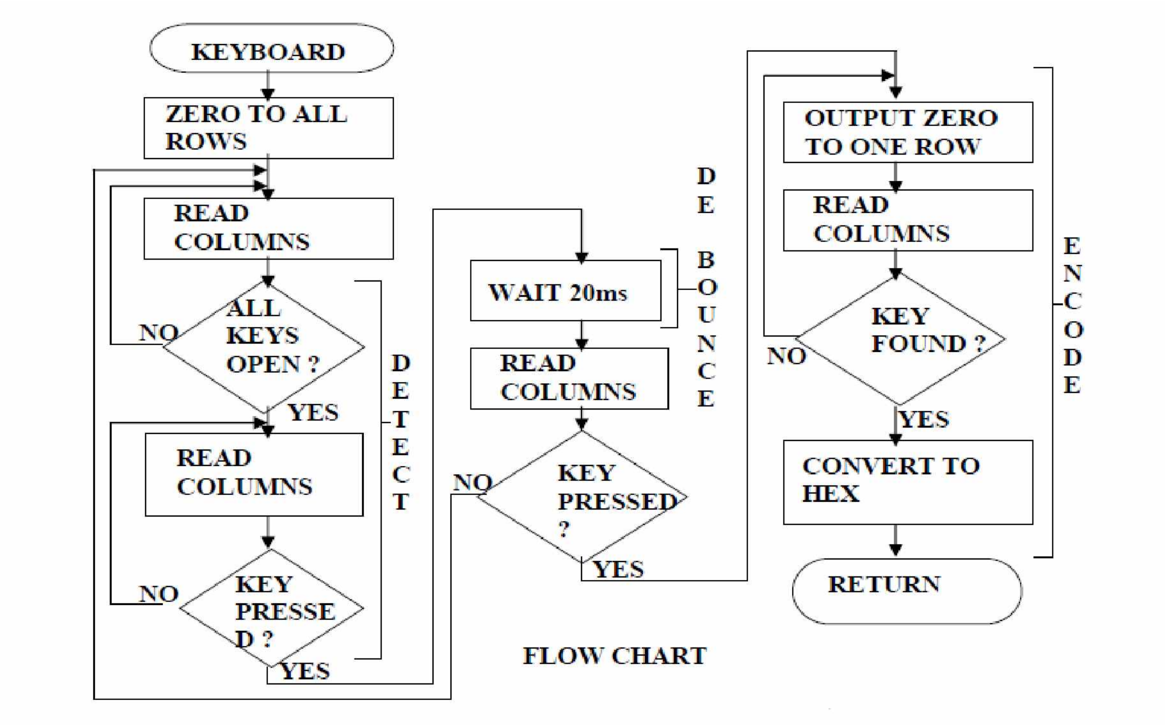
- This code would not match any of the row-column codes in the table, so after all the values in the table were checked, assigned register in program would be decremented from 0000H to FFFFH. The compare decrement cycle would continue through 65,536 memory locations until, by change the value in a memory location matched the row-column code. The contents of the lower byte register at hat point would be passed back to the calling routine. The changes are 1 in 256 that would be the correct value for one of the pressed keys. You should keep an error trap in a program whenever there is a chance for it.

2. Keyboard Interfacing with Hardware: For the system where the CPU is too busy to be bothered doing these tasks in software, an external device is used to do them.

- One of a MOS device which can be do this is the General Instruments AY5-2376 which can be connected to the rows and columns of a keyboard switch matrix.

- The AY5-2376 independently detects a keypress by cycling a low down through the rows and checking the columns. When it finds a key pressed, it waits a debounce time.

- If the key is still pressed after the debounce time, the AY5-2376 produces the 8-bit code for the pressed key and send it out to microcomputer port on 8 parallel lines. The microcomputer knows that a valid ASCII code is on the data lines, the AY5-2376 outputs a strobe pulse.



FLOW CHART

- The microcomputer can detect this strobe pulse and read in ASCII code on a polled basis or it can detect the strobe pulse on an interrupt basis.

- With the interrupt method the microcomputer doesn't have to pay any attention to the keyboard until it receives an interrupt signal.

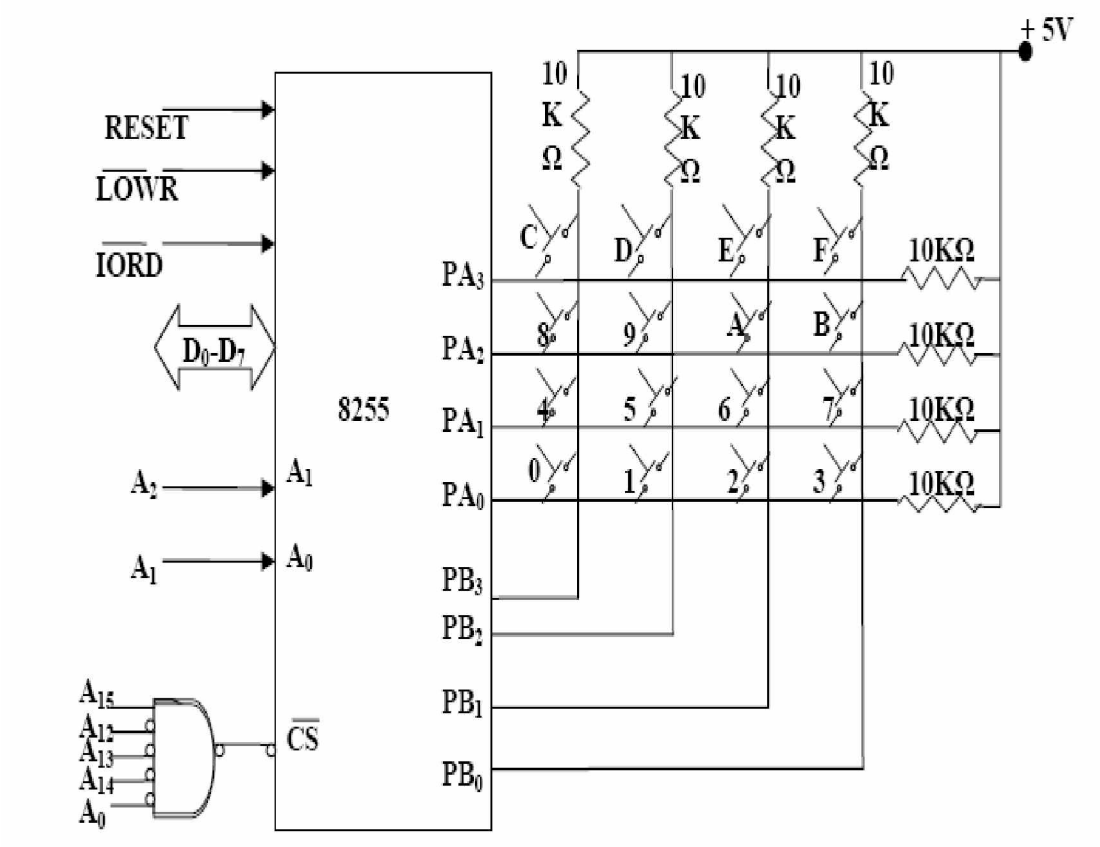
- So this method uses very little of the microcomputer time. The AY5-2376 has a feature called *two-key rollover*. This means that if two keys are pressed at nearly the same time, each key will be detected, debounced and converted to ASCII.

- The ASCII code for the first key and a strobe signal for it will be sent out then the ASCII code for the second key and a strobe signal for it will be sent out and compare this with two-key lockout.

Example : Interface a 4 * 4 keyboard with 8086 using 8255 an write an ALP for detecting a key closure and return the key code in AL. The debounce period for a key is 10ms. Use software debouncing technique. DEBOUNCE is an available 10ms delay routine.

• Solution:

Port A is used as output port for selecting a row of keys while Port B is used as an input port A and the port B lines are polled continuously till a key closure is sensed. The routine DEBOUNCE is called for key debouncing. The key code is depending upon the selected row and a low sensed column.

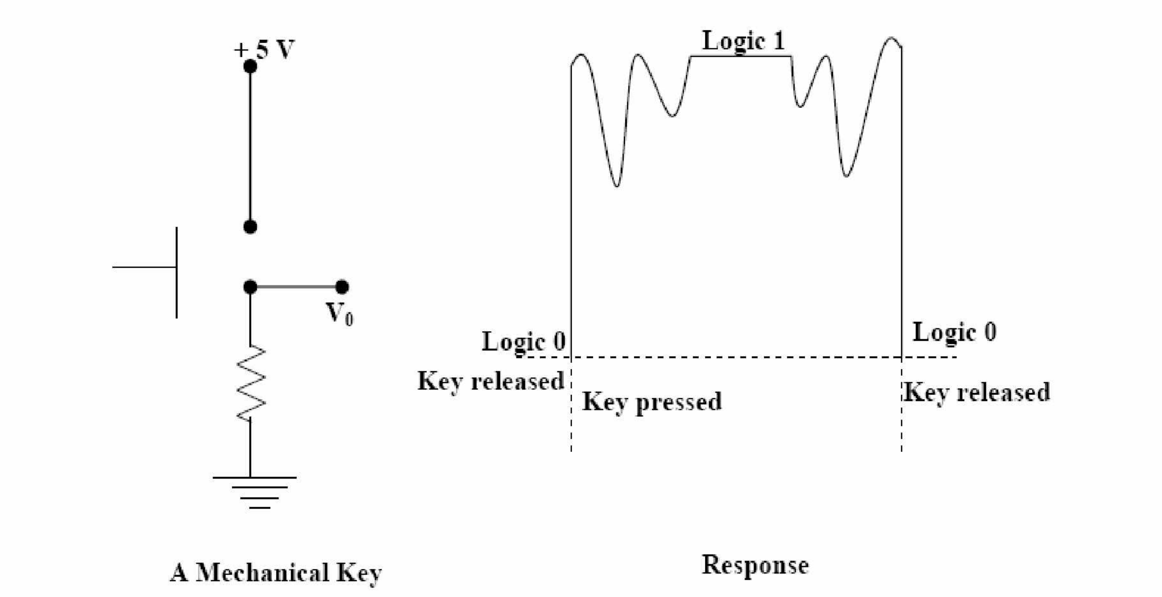


Interfacing 4 * 4 Keyboard

The higher order lines of port A and port B are left unused. The address of port A and port B will respectively 8000H and 8002H while address of CWR will be 8006H. The flow chart of the complete program is as given. The control word for this problem will be 82H. Code segment CS is used for storing the program code.

- **Key Debounce** : Whenever a mechanical push-button is pressed or released once, the mechanical components of the key do not change the position smoothly, rather it generates a transient response .
- These transient variations may be interpreted as the multiple key pressure and responded accordingly by the microprocessor system.

- To avoid this problem, two schemes are suggested: the first one utilizes a bistable multivibrator at the output of the key to debounce .
- The other scheme suggests that the microprocessor should be made to wait for the transient period (usually 10ms), so that the transient response settles down and reaches a steady state.
- A logic ‘0’ will be read by the microprocessor when the key is pressed.
- In a number of high precision applications, a designer may have two options- the first is to have more than one 8-bit port, read (write) the port one by one and then from the multibyte data, the second option allows forming 16-bit ports using two 8-bit ports and use 16-bit read or write operations



3.2.2 Interfacing To Alphanumeric Displays

- To give directions or data values to users, many microprocessor-controlled instruments and machines need to display letters of the alphabet and numbers. In systems where a large amount of data needs to be displayed a CRT is used to display the data. In system where only a small amount of data needs to be displayed, simple digit-type displays are often used.
- There are several technologies used to make these digit-oriented displays but we are discussing only the two major types.
- These are *light emitting diodes* (LED) and *liquid-crystal displays* (LCD).
- LCD displays use very low power, so they are often used in portable, battery-powered instruments. They do not emit their own light, they simply change the reflection of

available light. Therefore, for an instrument that is to be used in low-light conditions, you have to include a light source for LCDs or use LEDs which emit their own light.

- Alphanumeric LED displays are available in three common formats. For displaying only number and hexadecimal letters, simple 7-segment displays such as that as shown in fig are used.
- To display numbers and the entire alphabet, 18 segment displays such as shown in fig or 5 by 7 dot-matrix displays such as that shown in fig can be used. The 7-segment type is the least expensive, most commonly used and easiest to interface with, so we will concentrate first on how to interface with this type.

1. ***Directly Driving LED Displays:*** Figure shows a circuit that you might connect to a parallel port on a microcomputer to drive a single 7-segment , common-anode display. For a common-anode display, a segment is tuned on by applying a logic low to it.

- The 7447 converts a BCD code applied to its inputs to the pattern of lows required to display the number represented by the BCD code. This circuit connection is referred to as a *static display* because current is being passed through the display at all times.

- Each segment requires a current of between 5 and 30mA to light. Let’s assume you want a current of 20mA. The voltage drop across the LED when it is lit is about 1.5V.

- The output low voltage for the 7447 is a maximum of 0.4V at 40mA. So assume that it is about 0.2V at 20mA. Subtracting these two voltage drop from the supply voltage of 5V leaves 3.3V across the current limiting resistor. Dividing 3.3V by 20mA gives a value of 168Ω for the current-limiting resistor. The voltage drops across the LED and the output of 7447 are not exactly predictable and exact current through the LED is not critical as long as we don’t exceed its maximum rating.

2. Software-Multiplexed LED Display:

- The circuit in fig works for driving just one or two LED digits with a parallel output port. However, this scheme has several problem if you want to drive, eight digits.

- The first problem is power consumption. For worst-case calculations, assume that all 8 digits are displaying the digit 8, so all 7 segments are all lit. Seven segment time 20mA per segment gives a current of 140mA per digit. Multiplying this by 8 digits gives a total current of 1120mA or 1.12A for 8 digits.

- A second problem of the static approach is that each display digit requires a separate 7447 decoder, each of which uses of another 13mA. The current required by the decoders and the LED displays might be several times the current required by the reset of the circuitry in the instrument.

- To solve the problem of the static display approach, we use a *multiplex method*, example for an explanation of the multiplexing.

- The fig shows a circuit you can add to a couple of microcomputer ports to drive some common anode LED displays in a multiplexed manner. The circuit has only one 7447 and that the segment outputs of the 7447 are bused in parallel to the segment inputs of all the digits.

- The question that may occur to you on first seeing this is: Aren't all the digits going to display the same number? The answer is that they would if all the digits were turned on at the same time. The tricky of multiplexing displays is that only one display digit is turned on at a time.

- The PNP transistor is series with the common anode of each digit acts as on/off switch for that digit. Here's how the multiplexing process works.

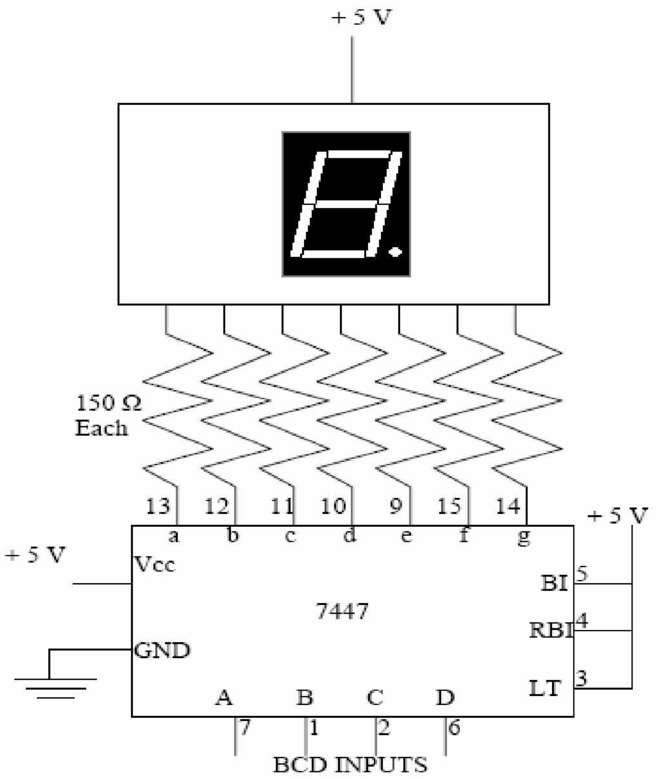
- The BCD code for digit 1 is first output from port B to the 7447. the 7447 outputs the corresponding 7-segment code on the segment bus lines. The transistor connected to digit 1 is then turned on by outputting a low to the appropriate bit of port A. All the rest of the bits of port A are made high to make sure no other digits are turned on. After 1 or 2 ms, digit 1 is turned off by outputting all highs to port A.

- The BCD code for digit 2 is then output to the 7447 on port B, and a word to turn on digit 2 is output on port A.

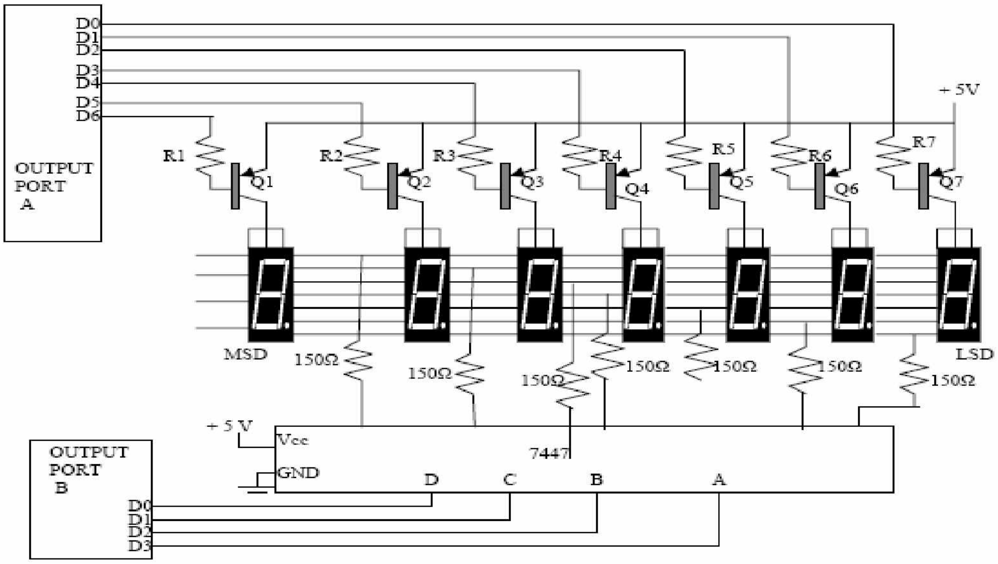
- After 1 or 2 ms, digit 2 is turned off and the process is repeated for digit 3. the process is continued until all the digits have had a turn. Then digit 1 and the following digits are lit again in turn.

- A procedure which is called on an interrupt basis every 2ms to keep these displays refreshed wit some values stored in a table. With 8 digits and 2ms per digit, you get back to digit 1 every 16ms or about 60 times a second.

- This refresh rate is fast enough so that the digits will each appear to be lit all time. Refresh rates of 40 to 200 times a second are acceptable.



Circuit for driving single 7-segment LED display with 7447



- The immediately obvious advantages of multiplexing the displays are that only one 7447 is required, and only one digit is lit at a time. We usually increase the current per segment to between 40 and 60 mA for multiplexed displays so that they will appear as bright as they would if they were not multiplexed. Even with this increased segment current, multiplexing gives a large saving in power and parts.

- The software-multiplexed approach we have just described can also be used to drive 18-segment LED devices and dot-matrix LED device. For these devices, however you replace the 7447 in fig with ROM which generates the required segment codes when the ASCII code for a character is applied to the address inputs of the ROM.

3.2.3 Interfacing Analog to Digital Data Converters

- In most of the cases, the PIO 8255 is used for interfacing the analog to digital converters with microprocessor.
- We have already studied 8255 interfacing with 8086 as an I/O port, in previous section. This section we will only emphasize the interfacing techniques of analog to digital converters with 8255.
- The analog to digital converters is treaded as an input device by the microprocessor, that sends an initialising signal to the ADC to start the analogy to digital data conversation process. The start of conversation signal is a pulse of a specific duration.
- The process of analog to digital conversion is a slow process, and the microprocessor has to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion EOC signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports.
- The time taken by the ADC from the active edge of SOC pulse till the active edge of EOC signal is called as the conversion delay of the ADC.
- It may range any where from a few microseconds in case of fast ADC to even a few hundred milliseconds in case of slow ADCs.
- The available ADC in the market use different conversion techniques for conversion of analog signal to digitals. Successive approximation techniques and dual slope integration techniques are the most popular techniques used in the integrated ADC chip.

• General algorithm for ADC interfacing contains the following steps:

1. Ensure the stability of analog input, applied to the ADC.
2. Issue start of conversion pulse to ADC
3. Read end of conversion signal to mark the end of conversion processes.
4. Read digital data output of the ADC as equivalent digital output.
5. Analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of the conversion to get correct results. This may be ensured by

a sample and hold circuit which samples the analog signal and holds it constant for a specific time duration. The microprocessor may issue a hold signal to the sample and hold circuit.

6. If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.

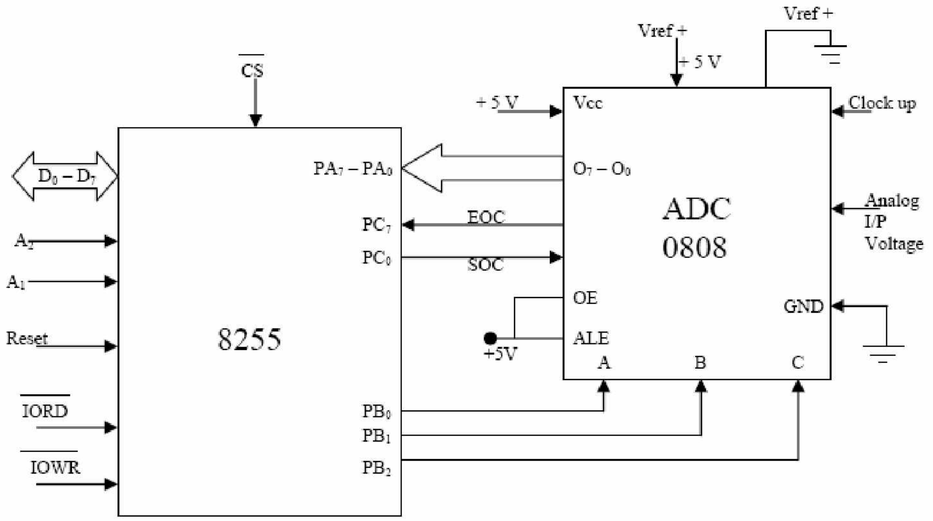
ADC 0808/0809 :

• The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters. This technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100µs at a clock frequency of 640 KHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits.

These converters internally have a 3:8 analog multiplexer so that at a time eight different analog conversion by using address lines - ADD A, ADD B, ADD C. Using these address inputs, multichannel data acquisition system can be designed using a single ADC.

The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hardwired to select the proper input.

• There are unipolar analog to digital converters, i.e. they are able to convert only positive analog input voltage to their digital equivalent. These chips do no contain any internal sample and hold circuit.



Interfacing 0808 with 8086

3.2.4 INTERFACING DIGITAL TO ANALOG CONVERTERS:

The digital to analog converters convert binary number into their equivalent voltages. The DAC find applications in areas like digitally controlled gains, motors speed controls, programmable gain amplifiers etc.

AD 7523 8-bit Multiplying DAC : This is a 16 pin DIP, multiplying digital to analog converter, containing R-2R ladder for D-A conversion along with single pole double thrown NMOS switches to connect the digital inputs to the ladder.

- The pin diagram of AD7523 is shown in fig the supply range is from +5V to +15V, while Vref may be any where between -10V to +10V. The maximum analog output voltage will be any where between -10V to +10V, when all the digital inputs are at logic high state.

- Usually a zener is connected between OUT1 and OUT2 to save the DAC from negative transients. An operational amplifier is used as a current to voltage converter at the output of AD to convert the current out put of AD to a proportional output voltage.

- It also offers additional drive capability to the DAC output. An external feedback resistor acts to control the gain. One may not connect any external feedback resistor, if no gain control is required.

- ***EXAMPLE***: Interfacing DAC AD7523 with an 8086 CPU running at 8MHZ and write an assembly language program to generate a sawtooth waveform of period 1ms with Vmax 5V.

- Solution: Fig shows the interfacing circuit of AD 74523 with 8086 using 8255. program gives an ALP to generate a sawtooth waveform using circuit.

```
ASSUME CS:CODE
CODE SEGMENT
START :MOV AL,80h ; make all ports output
OUT CW, AL
AGAIN :MOV AL,00h ; start voltage for ramp
BACK : OUT PA, AL
INC AL
CMP AL, 0FFh
JB BACK
JMP AGAIN
CODE ENDS
END START
```

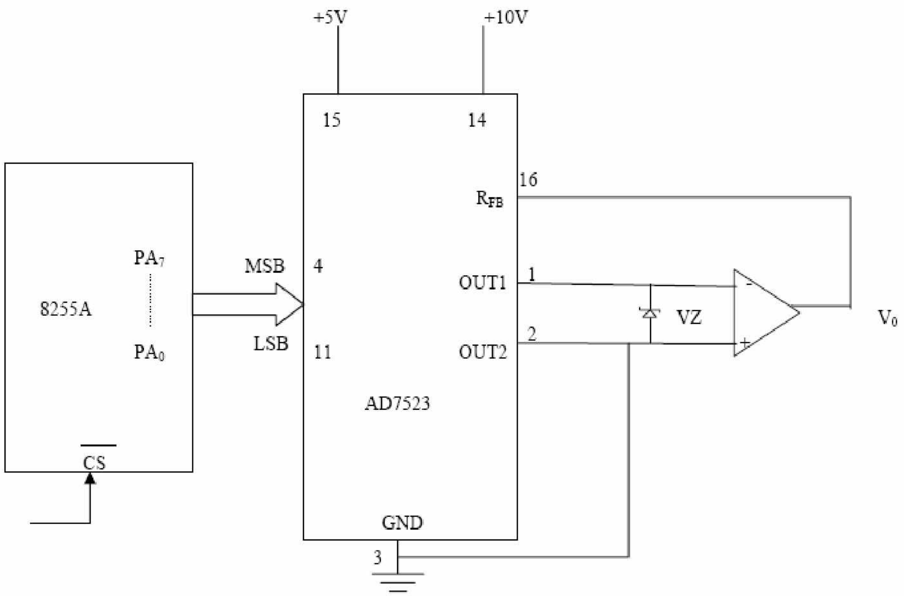


Fig: Interfacing of AD7523

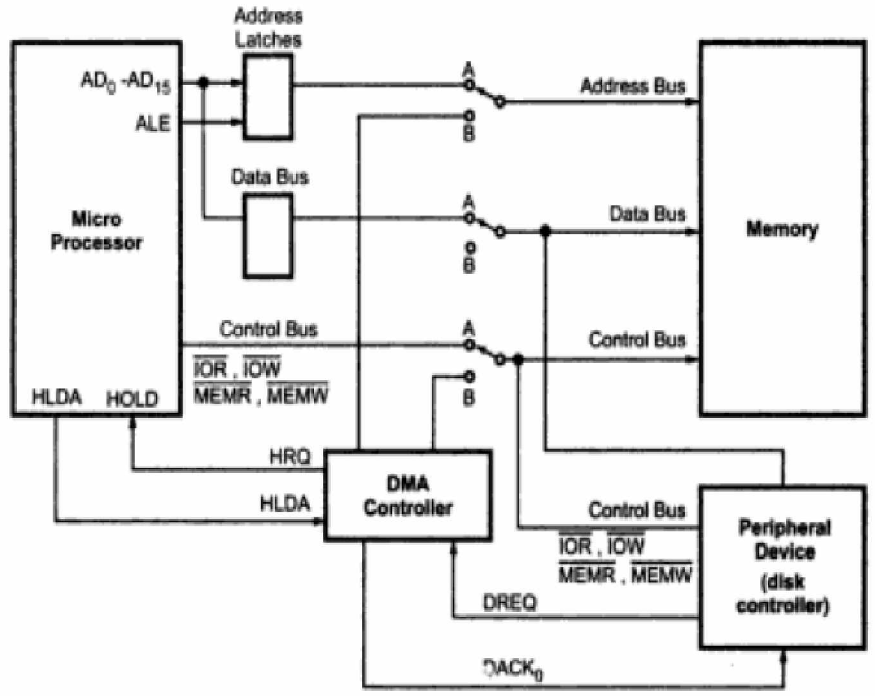
- In the above program, port A is initialized as the output port for sending the digital data as input to DAC. The ramp starts from the 0V (analog), hence AL starts with 00H. To increment the ramp, the content of AL is increased during each execution of loop till it reaches F2H.

- After that the saw tooth wave again starts from 00H, i.e. 0V(analog) and the procedure is repeated. The ramp period given by this program is precisely 1.000625 ms. Here the count F2H has been calculated by dividing the required delay of 1ms by the time required for the execution of the loop once. The ramp slope can be controlled by calling a controllable delay after the OUT instruction.

CHAPTER - 5

Programmable DMA Controller

The Intel* 8257 is a 4-channel direct memory access (DMA) controller. It is specifically designed to simplify the transfer of data at high speeds for the Intel® microcomputer systems. Its primary function is to generate, upon a peripheral request, a sequential memory address which will allow the peripheral to read or write data directly to or from memory. Acquisition of the system bus in accomplished via the CPU's hold function. The 8257 has priority logic that resolves the peripherals requests and issues a composite hold request to the CPU.


Address Latches
AD_0 -AD_{15}
ALE
Data Bus
Micro Processor
HLDA
HOLD
A
B
Address Bus
A
B
Data Bus
Memory
Control Bus
$\overline{IOR}$, $\overline{IOW}$
$\overline{MEMR}$, $\overline{MEMW}$
A
B
Control Bus
HRQ
HLDA
DMA Controller
Control Bus
$\overline{IOR}$, $\overline{IOW}$
$\overline{MEMR}$, $\overline{MEMW}$
Peripheral Device (disk controller)
DREQ
$DACK_0$

DMA controller operating in a microprocessor system

It maintains the DMA cycle count for each channel and outputs a control signal to notify the peripheral that the programmed number of DMA cycles is complete. Other output control signals simplify sectored data transfers. The 8257 represents a significant savings in component count for DMA-based microcomputer systems and greatly simplifies the transfer of data at high speed between peripherals and memories.

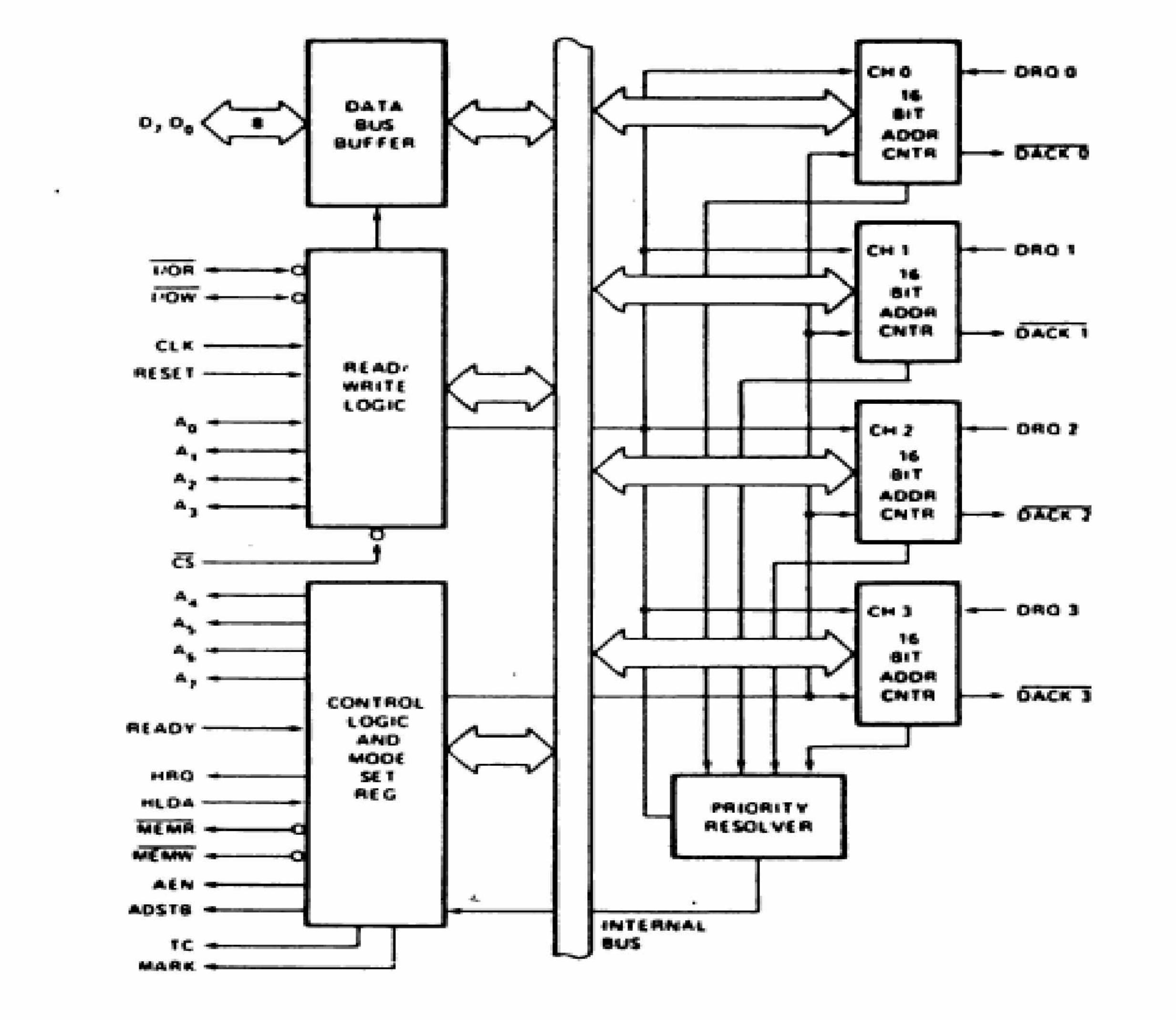


Figure 1. Block Diagram

FUNCTIONAL DESCRIPTION:

General:

The 8257 is a programmable. Direct Memory Access (DMA) device which, when coupled with a single Intel 8212 I/O port device, provides a complete four-channel DMA controller for use in Intel® microcomputer systems. After being initialized by software, the 8257 can transfer a block of data, containing up to 16.384 bytes, between memory and a peripheral device directly, without further intervention required of the CPU.

Upon receiving a DMA transfer request from an enabled peripheral, the 8257:

1. Acquires control of the system bus.

2. Acknowledges that requesting peripheral which is connected to the highest priority channel.

3. Outputs the least significant eight bits of the memory address onto system address lines A0-A7. Outputs the most significant eight bits of the memory address to the 8212 I/O port via.the data bus (the 8212 places these address bits on lines A8-A15), and

4. Generates the appropriate memory and I/O read/write control signals that cause the peripheral to receive or deposit a data byte directly from or to the addressed location in memory. The 8257 will retain control of the system bus and repeat the transfer sequence, as long as a peripheral maintains its DMA request.

Thus, the 8257 can transfer a block of data to/from a high speed peripheral (e.g.. a sector of data on a floppy disk) in a single "burst". When the specified number of data bytes have been transferred, the 8257 activates its Terminal Count (TC) output, informing the CPU that the operation is complete.

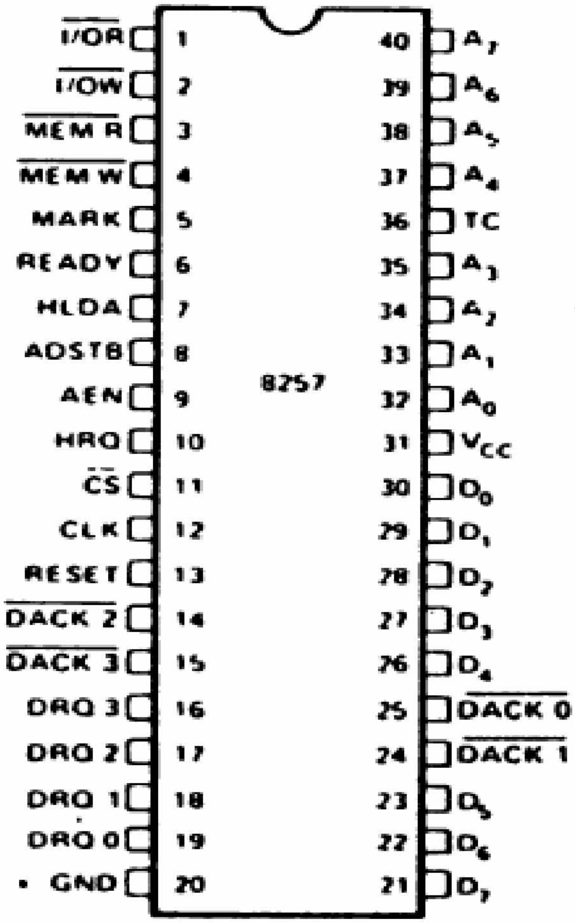


Figure 2. Pin Configuration

The 8257 offers three different modes of operation:

(1) DMA read, which causes data to be transferred from memory to a peripheral:

(2) DMA write, which causes data to be transferred from a peripheral to memory:and

(3) DMA verify, which does not actually involve the transfer of data.

When an 8257 channel is in the DMA verify mode, it will respond the same as described for transfer operations, except that no memory or I/O read/write control signals will be generated, thus preventing the transfer of data The 8257, however, will gain control of the system bus and will acknowledge the peripheral's DMA request for each DMA cycle. The peripheral can use these acknowledge signals to enable an internal access of each byte of a data block in order to execute some verification procedure, such as the accumulation of a CRC (Cyclic Redundancy Code) checkword. For example, a block of DMA verify cycles might follow a block of DMA read cycles (memory to peripheral) to allow the peripheral to verify its newly acquired data.

Block Diagram Description

1. DMA Channels

The 8257 provides four separate DMA channels (labeled CH-0 to CH-3). Each channel includes two sixteen-bit registers: (1) a DMA address register, and (2) a termi nal count register. Both registers must be initialized before a channel is enabled. The DMA address register is loaded with the address of the first memory location to be accessed. The value loaded into the low-order 14-bits of the terminal count register specifies the number of DMA cycles minus one before the Terminal Count (TC) output is activated. For instance, a terminal count of 0 would cause the TC output to be active in the first DMA cycle for that channel. In general, if N = the number of desired DMA cycles, load the value N-1 into the low-order 14-bits of the terminal count register. The most significant two bits of the terminal count register specify the type of DMA operation for that channel.

These two bits are not modified during a DMA cycle, but can be changed between DMA blocks. Each channel accepts a DMA Request (DRQn) input and provides a DMA Acknowledge (DACKn) output

(DRQ 0-DRQ 3)

DMA Request: These are individual asynchronous channel request inputs used by the peripherals to obtain a DMA cycle. If not in the rotating priority mode then DRQ 0 has the highest priority and DRQ 3 has the lowest. A request can be generated by raising the request line and holding it high until DMA acknowledge. For multiple DMA cycles (Burst Mode) the request line is held high until the DMA acknowledge of the last cycle arrives.

(DACK 0 - DACK 3)

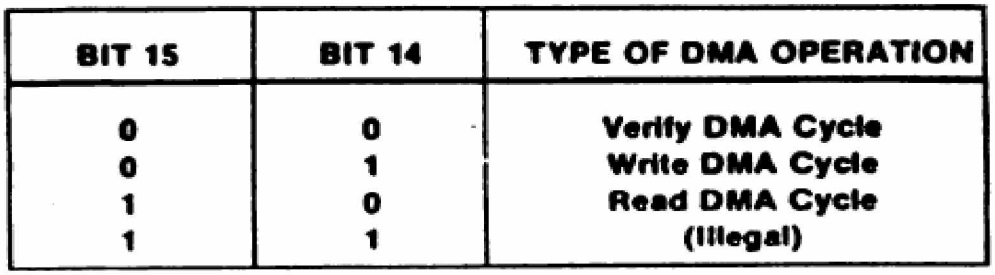
DMA Acknowledge: An active low level on the acknowledge output informs the peripheral connected to that channel that it has been selected for a DMA cycle. The DACK output acts as a "chip select'* for the peripheral device requesting service. This line goes active (low) and inactive (high) once for each byte transferred even if a burst of data is being transferred.

2. Data Bus Buffer

This three-state, bi-directional, eight bit buffer interfaces the 8257 to the system data bus.

(D0-D7)

Data Bus Lines: These are bi-directional three-state lines. When the 8257 is being programmed by the CPU. Eightbits of data for a DMA address register, a terminal count register or the Mode Set register are received on the data bus. When the CPU reads a DMA address register, a terminal count register or the Status register, the data is sent to the CPU over the data bus. During DMA cycles (when the 8257 is the bus master), the 8257 will output the most significant eight-bits of the memory address (from one of the DMA address registers) to the 8212 latch via the data bus. These address bits will be transferred at the beginning of the DMA cycle: the bus will then be released to handle the memory data transfer during the balance of the DMA cycle.



BIT 15	BIT 14	TYPE OF DMA OPERATION
0	0	Verify DMA Cycle
0	1	Write DMA Cycle
1	0	Read DMA Cycle
1	1	(Illegal)

<u>3. Read/Write Logic:</u>

When the CPU is programming or reading one of the 8257*s registers (i.e., when the 8257 is a "slave" device onthe system bus), the Read/Write Logic accepts the I/O Read (USE) or I/O Write (175OT) signal, decodes the least significant four address bits, (A0-A3), and either writes the contents of the data bus into the addressed register (if I/OW is true) or places the contents of the addressed register onto the data bus (if I/OR is true). During DMA cycles (i.e., when the 8257 is the bus "master"), the Read/Write Logic generates the I/O read and memory write (DMA write cycle) or I/O Write and memory read (DMA read cycle) signals which control the data link with the peripheral that has been granted the DMA cycle. Note that during DMA transfers Non-DMA I/O devices should be de-selected (disabled) using "AEN" signal to inhibit I/O device decoding of the memory address as an erroneous device address.

(I/OR)

I/O Read: An active-low, bi-directional three-state line. In the "slave" mode, it is an input which allows the 8-bit status register or the upper/lower byte of a 18-bit DMA address register or terminal count register to be read. In the "master" mode, I/OR is a control output which is used to access data from a peripheral during the DMA write cycle.

(I/OW)

I/O Write: An active-low, bi-directional three-state line. In the "slave" mode, it is an input which allows the contents of the data bus to be loaded into the 8-bit mode set register or the upper/lower byte of a 18-bit DMA address register or terminal count register. In the "master" mode. I/OW is a control output which allows data to be output to a peripheral during a DMA read cycle.

(CLK)

Clock Input: Generally from an Intel® 8224 Clock Gen erator device. (*2 TTL) or Intel® 8085A CLK output.

(RESET)

Reset: An asynchronous input (generally from an 8224 or 8085 device) which disables all DMA channels by clearing the mode register and 3-states all control lines.

(A_0-A_3)

Address Lines: These least significant four address lines are bi-directional. In the "slave" mode they are inputs which select one of the registers to be read or programmed. In the "master" mode, they are outputs which constitute the least significant four bits of the 16-bit memory address generated by the 8257.

(CS)

Chip Select: An active-low input which enables the I/O Read or I/O Write input when the 8257 is being read or programmed in the "slave" mode. In the "master" mode. CS is autDMAtically disabled to prevent the chip from selecting itself while performing the DMA function.

4. Control Logic:

This block controls the sequence of operations during all DMA cycles by generating the appropriate control signals and the 16-bit address that specifies the memory location to be accessed.

(A_4-A_7)

Address Lines: These four address lines are three-state outputs which constitute bits 4 through 7 of the 16-bit memory address generated by the 8257 during all DMA cycles.

(READY)

Ready: This asynchronous input is used to elongate the memory read and write cycles in the 8257 with wait states if the selected memory requires longer cycles. READY must conform to specified setup and hold times.

(HRQ)

Hold Request: This output requests control of the system bus. In systems with only one 8257, HRQ will normally be applied to the HOLD input on the CPU. HRQ must conform to specified setup and hold times.

(HLDA)

Hold Acknowledge: This input from the CPU indicates that the 8257 has acquired control of the system bus.

(MEMR)

Memory Read: This active-low three-state output is used to read data from the addressed memory location during DMA Read cycles.

(MEMW)

Memory Write: This active-low three-state output is used to write data into the addressed memory location during DMA Write cycles.

(ADSTB)

Address Strobe: This output strobes the most significant byte of the memory address into the 8212 device from the data bus.

(AEN)

Address Enable: This output is used to disable (float) the System Data Bus and the System Control Bus. It may also be used to disable (float) the System Address Bus by use of an enable on the Address Bus drivers in systems to inhibit non-DMA devices from responding during DMA cycles. It may be further used to isolate the 8257 data bus from the System Data Bus to facilitate the transfer of the 8 most significant DMA address bits over the 8257 data I/O pins without subjecting the System Data Bus to any timing constraints for the transfer. When the 8257 is used in an I/O device structure (as opposed to memory mapped), this AEN output should be used to disable the selection of an I/O device when the DMA address is on the address bus. The I/O device selection should be determined by the DMA acknowledge outputs for the 4 channels.

(TC)

Terminal Count: This output notifies the currently selected peripheral that the present DMA cycle should be the last cycle for this data block. If the TC STOP bit in the Mode

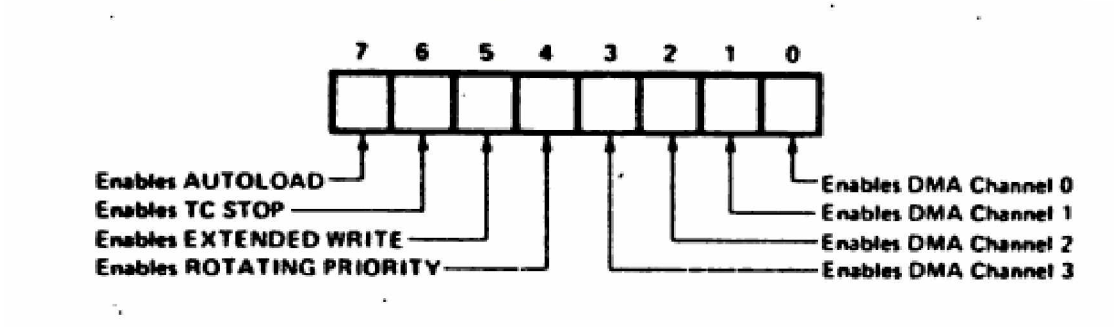
Set register is set. The selected channel will be autDMAtically disabled at the end of that DMA cycle. TC is activated when the 14-bit value in the selected channel's terminal count register equals zero. Recall that the loworder 14-bits of the terminal count register should be loaded with the values (n-1). Where n = the desired number of the DMA cycles.

(MARK)

Modulo 128 Mark: This output notifies the selected peripheral that the current DMA cycle is the 128th cycle since the previous MARK output. MARK always occurs at 128 (and all multiples of 128) cycles from the end of the data block. Only if the total number of DMA cycles (n) is evenly divisable by 128 (and the terminal count register was loaded with n-1). Will MARK occur at 128 (and each succeeding multiple of 128) cycles from the beginning of the data block.

5. Mode Set Register:

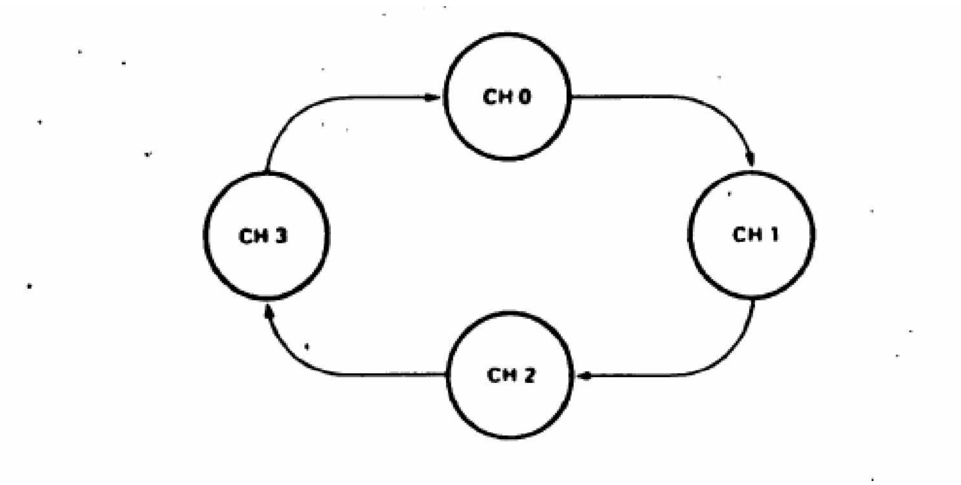
When set, the various bits in the Mode Set register enable each of the four DMA channels, and allow four different options for the 8257:



The Mode Set register is normally programmed by the CPU after the DMA address register(s) and terminal count register(s) are initialized. The Mode Set Register is cleared by the RESET input, thus disabling all options, inhibiting all channels, and preventing bus conflicts on power-up. A channel should not be left enabled unless its DMA address and terminal count registers contain valid values; otherwise, an inadvertent DMA request (DROn) from a peripheral could initiate a DMA cycle that would destroy memory data. The various options which can be enabled by bits in the Mode Set register are explained below:

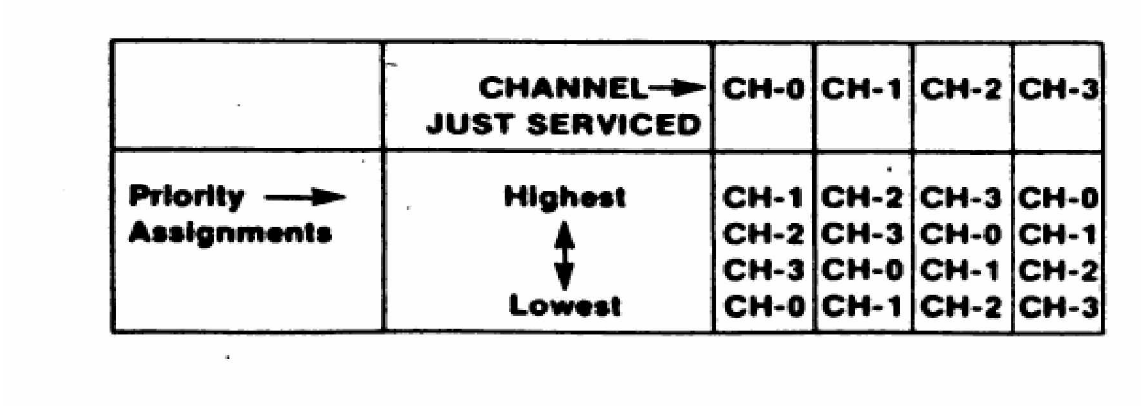
Rotating Priority Bit 4

In the Rotating Priority Mode, the priority of the channels has a circular sequence. After each DMA cycle, the priority of each channel changes. The channel which had just been serviced will have the lowest priority.



If the ROTATING PRIORITY bit is not set (set to a zero).each DMA channel has a fixed priority. In the fixed priority mode. Channel 0 has the highest priority and Channel 3 has the lowest priority. If the ROTATING PRIORITY bit is set to a one. the priority of each channel changes after each DMA cycle (not each DMA request).

Each channel moves up to the next highest priority assignment, while the channel which has just been serviced moves to the lowest priority assignment:



	CHANNEL→ JUST SERVICED	CH-0	CH-1	CH-2	CH-3
Priority → Assignments	Highest	CH-1	CH-2	CH-3	CH-0
	↕	CH-2	CH-3	CH-0	CH-1
		CH-3	CH-0	CH-1	CH-2
	Lowest	CH-0	CH-1	CH-2	CH-3

Note that rotating priority will prevent any one channel from monopolizing the PMA mode; consecutive DMA cycles will service different channels if more,than one channel is enabled and requesting service. There is no overhead penalty associated with this mode of opera tion. All DMA operations began with Channel 0 initially assigned to the highest priority for the first DMA cycle.

Extended Write Bit 5

If the EXTENDED WRITE bit is set. the duration of both the MEMW and I/OW signals is extended by activating them earlier in the DMA cycle. Data transfers within micro computer systems proceed asynchronously to allow use of various types of memory and I/O devices with different access times. If a device cannot be accessed within a specific amount of time it returns a "not ready" indication to the 8257 that causes the 8257 to insert one or more wait states in its internal sequencing. Some devices are fast enough to be accessed without the use of wait states, but if they generate their READY response with the leading edge of the f/SW or MEMW signal (which generally occurs late in the transfer sequence), they would normally cause the 8257 to enter a wait state because it does not receive READY in time. For systems with these types of devices, the Extended Write option provides alternative timing for the I/O and memory write signals which allows the devices to return an early READY and prevents the unnecessary occurrence of wait states in the 8257. thus increasing system throughput.

TC Stop Bit 6

If the TC STOP bit is set. a channel is disabled (i.e.. its enable bit is reset) after the Terminal Count (TC) output goes true, thus autDMAtically preventing further DMA operation on that channel. The enable bit for that channel must be re-programmed to continue or begin another DMA operation. If the TC STOP bit is not set. The occurrence of the TC output has no effect on the channel enable bits. In this case, it is generally the responsibility of the peripheral to cease DMA requests in order to terminate a DMA operation.

Auto Load Bit 7

The Auto Load mode permits Channel 2 to be used for repeat block or block chaining operations, without immediate software intervention between blocks. Chan nel 2 registers are initialized as usual for the first data block; Channel 3 registers, however, are used to store the block re-initialization parameters (DMA starting address, terminal count and DMA transfer mode). After the first block of DMA cycles is executed by Channel 2 (i.e.. after the TC output goes true), the parameters stored in the Channel 3 registers are transferred to Channel 2 during an "update" cycle. Note that the TC STOP feature, described above, has no effect on Channel 2 when the Auto Load bit is set.

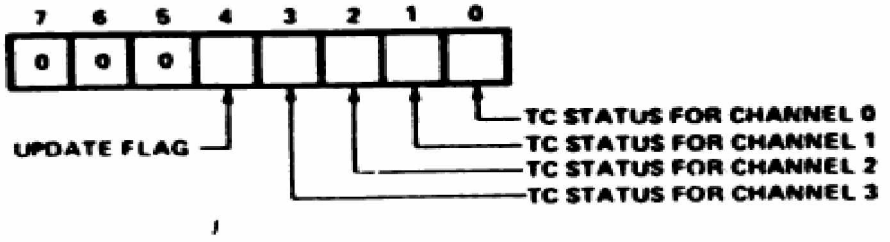
If the Auto Load bit is set. the initial parameters for Channel 2 are autDMAtically duplicated in the Channel 3 registers when Channel 2 is programmed.

This permits repeat block operations to be set up with the programming of a single channel. Repeat block operations can be used in applications such as CRT refreshing. Channels 2 and 3 can still be loaded with separate values if Channel 2 is loaded before loading Channel 3. Note that in the Auto Load mode, Channel 3 is still available to the user if the Channel 3 enable bit is set. but use of this channel will change the values to be auto loaded into Channel 2 at update time. All that is necessary to use the Auto Load feature for chaining operations is to reload Channel 3 registers at the conclusion of each update cycle with the new parameters for the next data block transfer. Each time that the 8257 enters an update cycle, the update flag in the status register is set and parameters in Channel 3 are transferred to Channel 2. non-destructively for Channel 3.

The actual re-initialization of Channel 2 occurs at the beginning of the next channel 2 DMA cycle after the TC cycle. This will be the first DMA cycle of the new data block for Channel 2. The update flag is cleared at the conclusion of this DMA cycle. For chaining operations, the update flag in the status register can be monitored by the CPU to determine when the re-initialization process has been completed so that the next block parameters can be safely loaded into Channel 3.

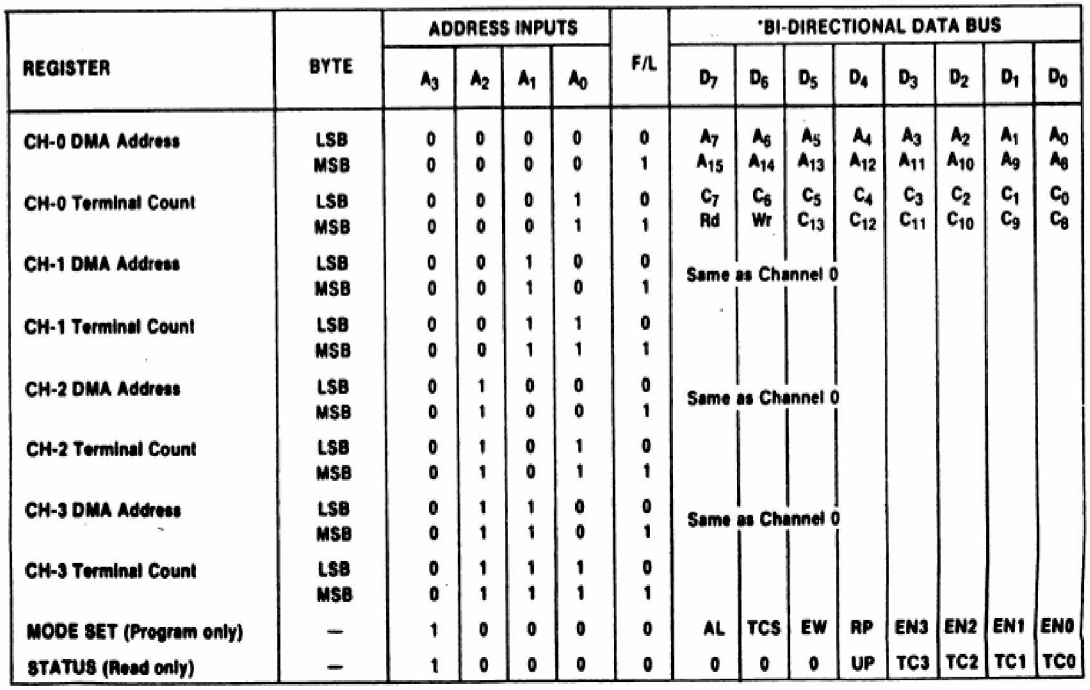
6. Status Register

The eight-bit status register indicates which channels have reached a terminal count condition and includes the update flag described previously.



The TC status bits are set when the Terminal Count (TC) output is activated for that channel. These bits remain set until the status register is read or the 8257 is reset. The UPDATE FLAG, however, is not affected by a status register read operation. The UPDATE FLAG can be cleared by resetting the 8257. by changing to the non-auto load mode (i.e.. by resetting the AUTO LOAD bit in the Mode Set register) or it can be left to clear itself at the completion of the update cycle. The purpose of the UPDATE FLAG is to prevent the CPU from inadvertently skipping a data block by overwriting a starting address or terminal count in the Channel 3 registers before those parameters are properly auto-loaded into Channel 2.

The user is cautioned against reading the TC status register and using this information to re enable channels that have not completed operation. Unless the DMA channels are inhibited a channel could reach terminal count (TC) between the status read and the mode write. DMA can be inhibited by a hardware gate on the HRQ line or by disabling channels with a mode word before reading the TC status.



REGISTER	BYTE	ADDRESS INPUTS				F/L	*BI-DIRECTIONAL DATA BUS							
		A_3	A_2	A_1	A_0		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CH-0 DMA Address	LSB MSB	0 0	0 0	0 0	0 0	0 1	A_7 A_{15}	A_6 A_{14}	A_5 A_{13}	A_4 A_{12}	A_3 A_{11}	A_2 A_{10}	A_1 A_9	A_0 A_8
CH-0 Terminal Count	LSB MSB	0 0	0 0	0 0	1 1	0 1	C_7 Rd	C_6 Wr	C_5 C_{13}	C_4 C_{12}	C_3 C_{11}	C_2 C_{10}	C_1 C_9	C_0 C_8
CH-1 DMA Address	LSB MSB	0 0	0 0	1 1	0 0	0 1	Same as Channel 0							
CH-1 Terminal Count	LSB MSB	0 0	0 0	1 1	1 1	0 1								
CH-2 DMA Address	LSB MSB	0 0	1 1	0 0	0 0	0 1	Same as Channel 0							
CH-2 Terminal Count	LSB MSB	0 0	1 1	0 0	1 1	0 1								
CH-3 DMA Address	LSB MSB	0 0	1 1	1 1	0 0	0 1	Same as Channel 0							
CH-3 Terminal Count	LSB MSB	0 0	1 1	1 1	1 1	0 1								
MODE SET (Program only)	—	1	0	0	0	0	AL	TCS	EW	RP	EN3	EN2	EN1	EN0
STATUS (Read only)	—	1	0	0	0	0	0	0	0	UP	TC3	TC2	TC1	TC0

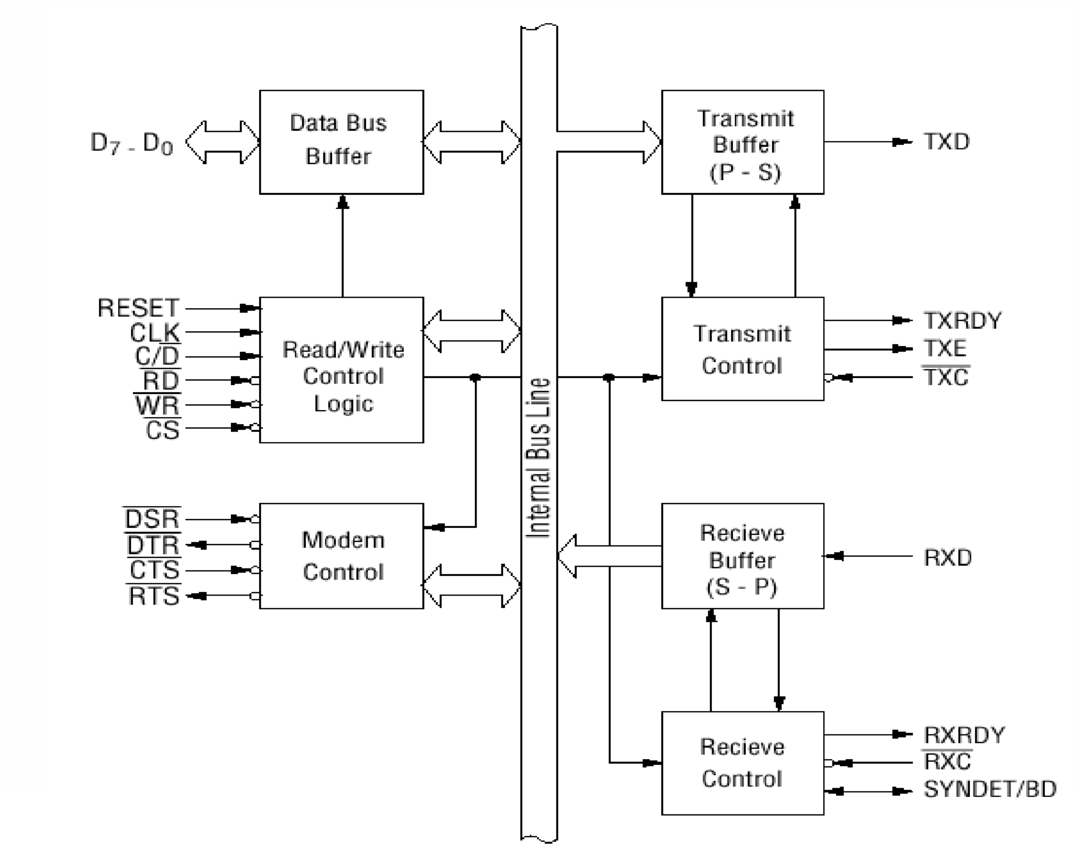
*A_0-A_{15}: DMA Starting Address, C_0-C_{13}: Terminal Count value (N-1), Rd and Wr: DMA Verify (00), Write (01) or Read (10) cycle selection, AL: Auto Load, TCS: TC STOP, EW: EXTENDED WRITE, RP: ROTATING PRIORITY, EN3-EN0: CHANNEL ENABLE MASK, UP: UPDATE FLAG, TC3-TC0: TERMINAL COUNT STATUS BITS.

CHAPTER-6

UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER (USART)-8251

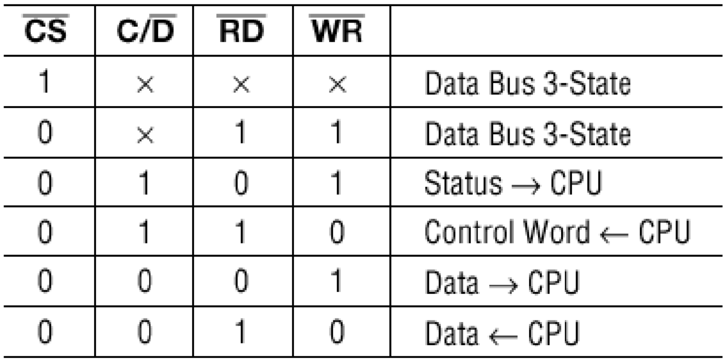
Block Diagram:

The 8251 is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication. As a peripheral device of a microcomputer system, the 8251 receives parallel data from the CPU and transmits serial data after conversion. This device also receives serial data from the outside and transmits parallel data to the CPU after conversion.



Block diagram of the 8251 USART

The 8251 functional configuration is programmed by software. Operation between the 8251 and a CPU is executed by program control. Table 1 shows the operation between a CPU and the device.



$\overline{CS}$	C/D	$\overline{RD}$	$\overline{WR}$	
1	×	×	×	Data Bus 3-State
0	×	1	1	Data Bus 3-State
0	1	0	1	Status → CPU
0	1	1	0	Control Word ← CPU
0	0	0	1	Data → CPU
0	0	1	0	Data ← CPU

Table : Operation between a CPU and 8251

Control Words

There are two types of control word.

1. Mode instruction (setting of function)
2. Command (setting of operation)

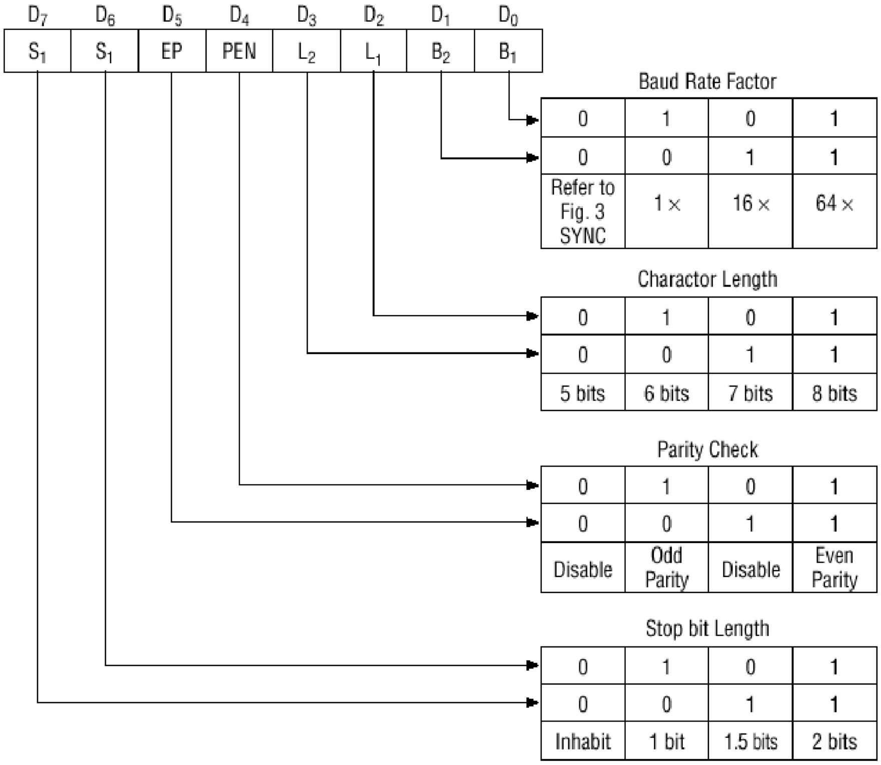
1) Mode Instruction

Mode instruction is used for setting the function of the 8251. Mode instruction will be in "wait for write" at either internal reset or external reset. That is, the writing of a control word after resetting will be recognized as a "mode instruction."

Items set by mode instruction are as follows:

- Synchronous/asynchronous mode
- Stop bit length (asynchronous mode)
- Character length
- Parity bit
- Baud rate factor (asynchronous mode)
- Internal/external synchronization (synchronous mode)
- Number of synchronous characters (Synchronous mode)

The bit configuration of mode instruction is shown in Figures 2 and 3. In the case of synchronous mode, it is necessary to write one-or two byte sync characters. If sync characters were written, a function will be set because the writing of sync characters constitutes part of mode instruction.



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S_1	S_1	EP	PEN	L_2	L_1	B_2	B_1

Baud Rate Factor

0	1	0	1
0	0	1	1
Refer to Fig. 3 SYNC	1 ×	16 ×	64 ×

Charactor Length

0	1	0	1
0	0	1	1
5 bits	6 bits	7 bits	8 bits

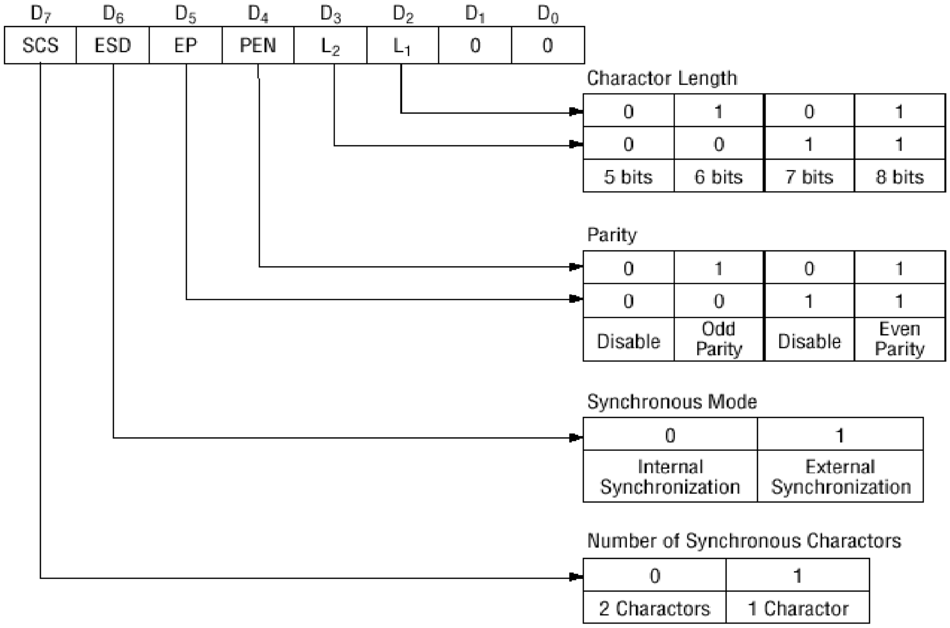
Parity Check

0	1	0	1
0	0	1	1
Disable	Odd Parity	Disable	Even Parity

Stop bit Length

0	1	0	1
0	0	1	1
Inhabit	1 bit	1.5 bits	2 bits

Fig. 2 Bit Configuration of Mode Instruction (Asynchronous)



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SCS	ESD	EP	PEN	L_2	L_1	0	0

Charactor Length

0	1	0	1
0	0	1	1
5 bits	6 bits	7 bits	8 bits

Parity

0	1	0	1
0	0	1	1
Disable	Odd Parity	Disable	Even Parity

Synchronous Mode

0	1
Internal Synchronization	External Synchronization

Number of Synchronous Charactors

0	1
2 Charactors	1 Charactor

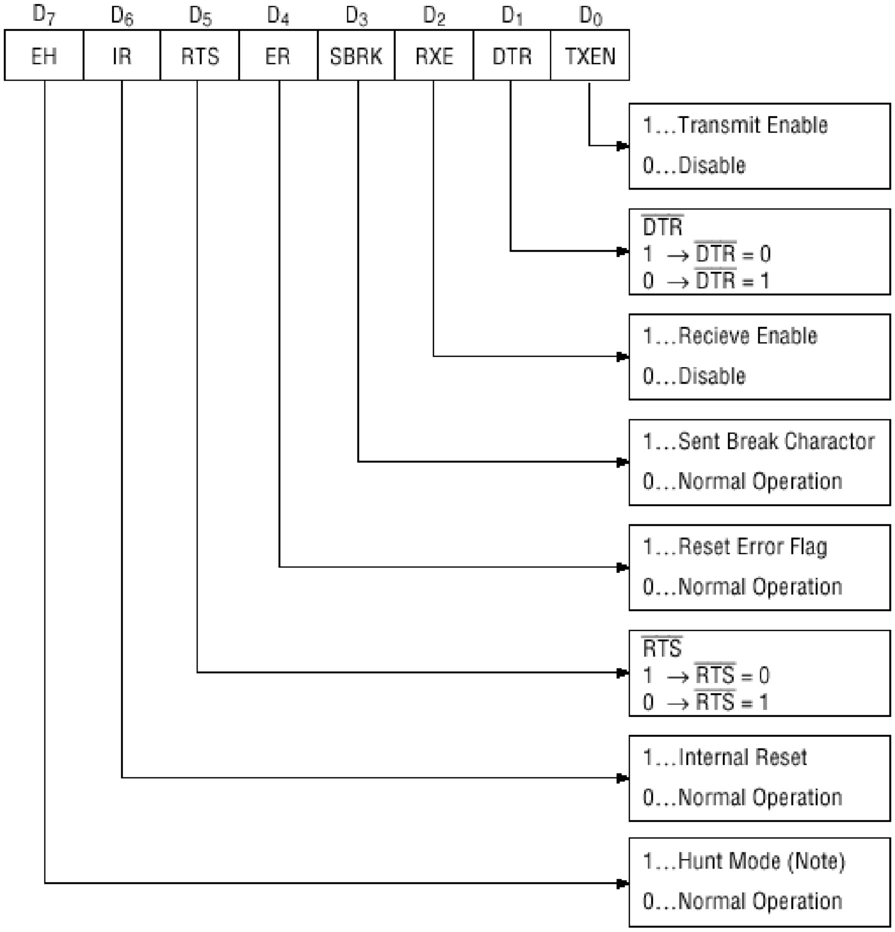
Fig. 3 Bit Configuration of Mode Instruction (Synchronous)

2) Command

Command is used for setting the operation of the 8251. It is possible to write a command whenever necessary after writing a mode instruction and sync characters.

Items to be set by command are as follows:

- Transmit Enable/Disable
- Receive Enable/Disable
- DTR, RTS Output of data.
- Resetting of error flag.
- Sending to break characters
- Internal resetting
- Hunt mode (synchronous mode)



Note: Seach mode for synchronous charactors in synchronous mode.

Fig. 4 Bit Configuration of Command

Status Word

It is possible to see the internal status of the 8251 by reading a status word. The bit configuration of status word is shown in Fig. 5.

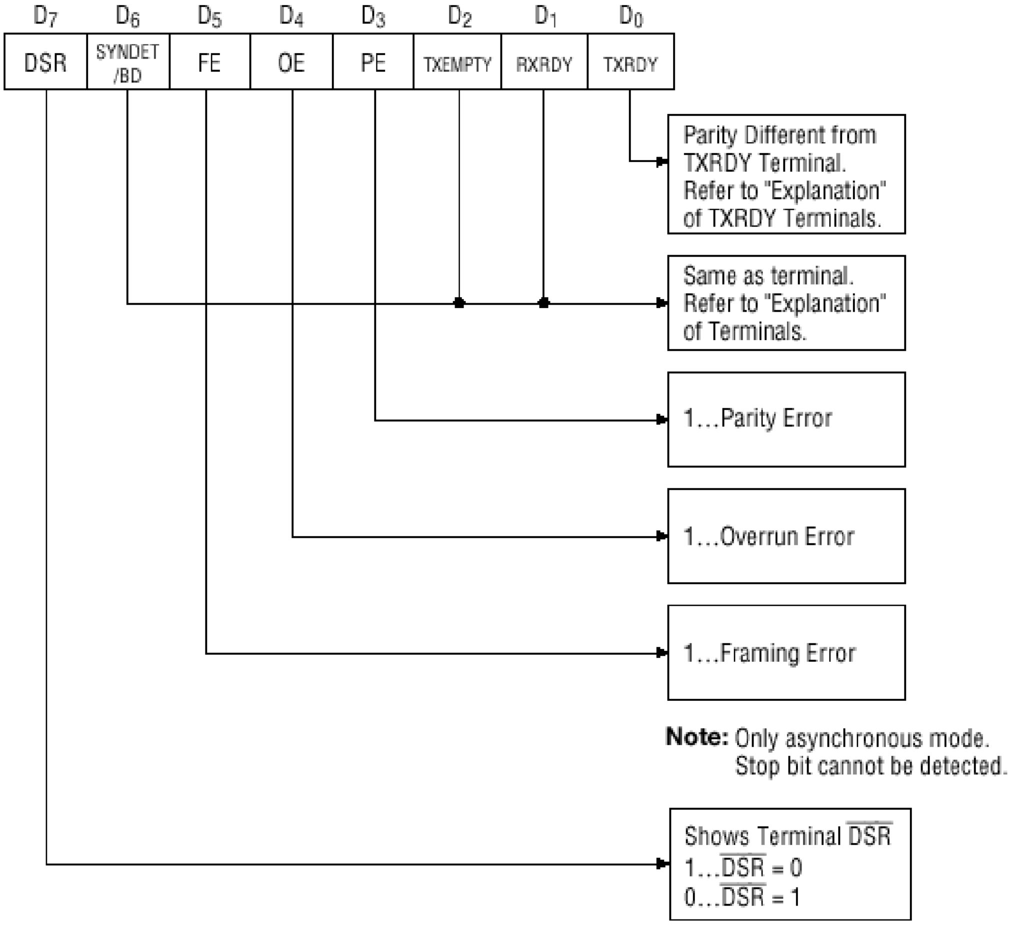

D7
D6
D5
D4
D3
D2
D1
D0
DSR
SYNDET /BD
FE
OE
PE
TXEMPTY
RXRDY
TXRDY
Parity Different from TXRDY Terminal. Refer to "Explanation" of TXRDY Terminals.
Same as terminal. Refer to "Explanation" of Terminals.
1…Parity Error
1…Overrun Error
1…Framing Error
Note: Only asynchronous mode. Stop bit cannot be detected.
Shows Terminal $\overline{DSR}$
1…$\overline{DSR}$ = 0
0…$\overline{DSR}$ = 1

Fig. 5 Bit Configuration of Status Word

D 0 to D 7 (l/O terminal)

This is bidirectional data bus which receive control words and transmits data from the CPU and sends status words and received data to CPU.

RESET (Input terminal)

A "High" on this input forces the 8251 into "reset status." The device waits for the writing of "mode instruction." The min. reset width is six clock inputs during the operating status of CLK.

CLK (Input terminal)

CLK signal is used to generate internal device timing. CLK signal is independent of RXC or TXC. However, the frequency of CLK must be greater than 30 times the RXC and TXC at Synchronous mode and Asynchronous "x1" mode, and must be greater than 5 times at Asynchronous "x16" and "x64" mode.

WR (Input terminal)

This is the "active low" input terminal which receives a signal for writing transmit data and control words from the CPU into the 8251.

RD (Input terminal)

This is the "active low" input terminal which receives a signal for reading receive data and status words from the 8251.

C/D (Input terminal)

This is an input terminal which receives a signal for selecting data or command words and status words when the 8251 is accessed by the CPU. If C/D = low, data will be accessed. If C/D = high, command word or status word will be accessed.

CS (Input terminal)

This is the "active low" input terminal which selects the 8251 at low level when the CPU accesses. Note: The device won't be in "standby status"; only setting CS = High.

TXD (output terminal)

This is an output terminal for transmitting data from which serial-converted data is sent out. The device is in "mark status" (high level) after resetting or during a status when transmit is disabled. It is also possible to set the device in "break status" (low level) by a command.

TXRDY (output terminal)

This is an output terminal which indicates that the 8251is ready to accept a transmitted data character. But the terminal is always at low level if CTS = high or the device was set in "TX disable status" by a command. Note: TXRDY status word indicates that transmit data character is receivable, regardless of CTS or command. If the CPU writes a data character, TXRDY will be reset by the leading edge or WR signal.

TXEMPTY (Output terminal)

This is an output terminal which indicates that the 8251 has transmitted all the characters and had no data character. In "synchronous mode," the terminal is at high level, if transmit data characters are no longer remaining and sync characters are automatically transmitted. If the CPU writes a data character, TXEMPTY will be reset by the leading edge of WR signal. Note : As the transmitter is disabled by setting CTS "High" or command, data written before disable will be sent out. Then TXD and TXEMPTY will be "High". Even if a data is written after disable, that data is not sent out and TXE will be "High".After the transmitter is enabled, it sent out. (Refer to Timing Chart of Transmitter Control and Flag Timing)

TXC (Input terminal)

This is a clock input signal which determines the transfer speed of transmitted data. In "synchronous mode," the baud rate will be the same as the frequency of TXC. In "asynchronous mode", it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16 or 1/64 the TXC. The falling edge of TXC sifts the serial data out of the 8251.

RXD (input terminal)

This is a terminal which receives serial data.

RXRDY (Output terminal)

This is a terminal which indicates that the 8251 contains a character that is ready to READ. If the CPU reads a data character, RXRDY will be reset by the leading edge of RD signal. Unless the CPU reads a data character before the next one is received completely, the preceding data will be lost. In such a case, an overrun error flag status word will be set.

RXC (Input terminal)

This is a clock input signal which determines the transfer speed of received data. In "synchronous mode," the baud rate is the same as the frequency of RXC. In "asynchronous mode," it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16, 1/64 the RXC.

SYNDET/BD (Input or output terminal)

This is a terminal whose function changes according to mode. In "internal synchronous mode." this terminal is at high level, if sync characters are received and synchronized. If a status word is read, the terminal will be reset. In "external synchronous mode, "this is an input terminal. A "High" on this input forces the 8251 to start receiving data characters.

In "asynchronous mode," this is an output terminal which generates "high level"output upon the detection of a "break" character if receiver data contains a "low-level" space between the stop bits of two continuous characters. The terminal will be reset, if RXD is at high level. After Reset is active, the terminal will be output at low level.

DSR (Input terminal)

This is an input port for MODEM interface. The input status of the terminal can be recognized by the CPU reading status words.

DTR (Output terminal)

This is an output port for MODEM interface. It is possible to set the status of DTR by a command.

CTS (Input terminal)

This is an input terminal for MODEM interface which is used for controlling a transmit circuit. The terminal controls data transmission if the device is set in "TX Enable" status by a command. Data is transmitable if the terminal is at low level.

RTS (Output terminal)

This is an output port for MODEM interface. It is possible to set the status RTS by a command.

***

Communication Media:

The purpose of this application note is to attempt to describe the main elements in Serial Communication. This application note attempts to cover enough technical details of RS232, RS422 and RS485.

RS422 Serial Communication

RS422 is a Standard interfaces approved by the Electronic Industries Association (EIA), and designed for greater distances and higher Baud rates than RS232. In its simplest form, a pair of converters from RS232 to RS422 (and back again) can be used to form an "RS232 extension cord." Data rates of up to 100K bits / second and distances up to 4000 Ft. can be accommodated with RS422. RS422 is also specified for multi-drop (party-line) applications where only one driver is connected to, and transmits on, a "bus" of up to 10 receivers. RS422 devices cannot be used to construct a truly multi-point network. A true multi-point network consists of multiple drivers and receivers connected on a single bus, where any node can transmit or receive data.

DCE and DTE Devices

DTE stands for Data Terminal Equipment, and DCE stands for Data Communications Equipment. These terms are used to indicate the pin-out for the connectors on a device and the direction of the signals on the pins. Your computer is a DTE device, while most other devices such as modem and other serial devices are usually DCE devices.

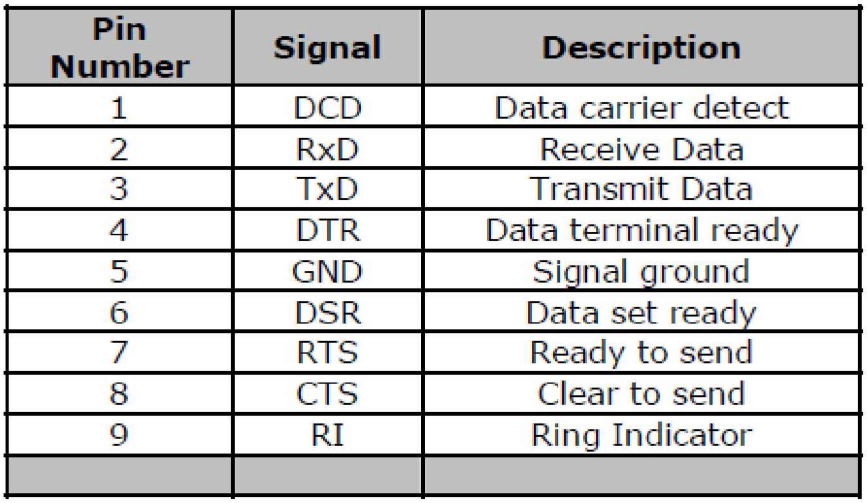
RS-232 has been around as a standard for decades as an electrical interface between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) such as modems or DSUs. It appears under different incarnations such as RS-232C, RS-232D, V.24, V.28 or V.10. RS-232 is used for asynchronous data transfer as well as synchronous links such as SDLC, HDLC, Frame Relay and X.25

RS232

RS-232 (Recommended standard-232) is a standard interface approved by the Electronic Industries Association (EIA) for connecting serial devices. In other words, RS-232 is a longestablished standard that describes the physical interface and protocol for relatively low-speed serial data communication between computers and related devices. An industry trade group, the Electronic Industries Association (EIA), defined it originally for teletypewriter devices. In 1987, the EIA released a new version of the standard and changed the name to EIA-232-D. Many people, however, still refer to the standard as RS-232C, or just RS- 232. RS-232 is the interface that your computer uses to talk to and exchange data with your modem and other serial devices. The serial ports on most computers use a subset of the RS- 232C standard.

RS232 on DB9 (9-pin D-type connector)

There is a standardized pinout for RS-232 on a DB9 connector, as shown below

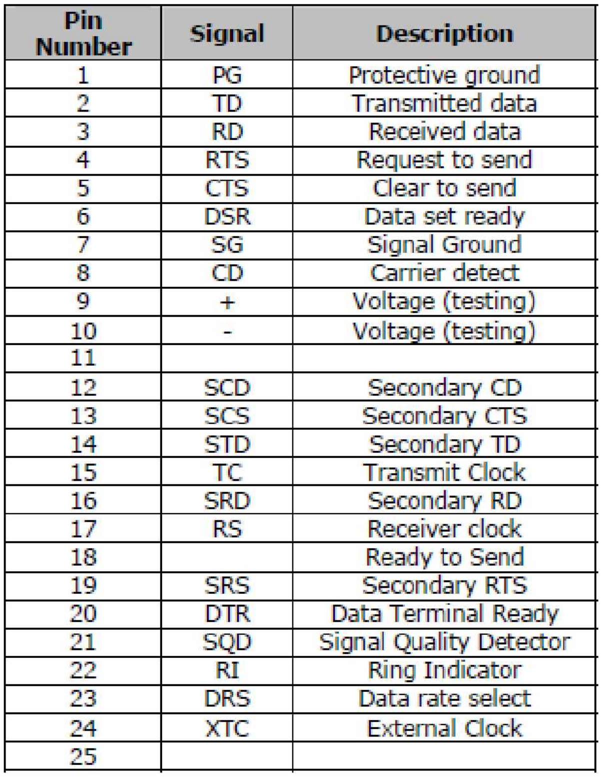


Pin Number	Signal	Description
1	DCD	Data carrier detect
2	RxD	Receive Data
3	TxD	Transmit Data
4	DTR	Data terminal ready
5	GND	Signal ground
6	DSR	Data set ready
7	RTS	Ready to send
8	CTS	Clear to send
9	RI	Ring Indicator

25-pin D-type connector Pin assignment

RS232 on DB25 (25-pin D-type connector)

In DB-25 connector most of the pins are not needed for normal PC communications, and indeed, most new PCs are equipped with male D type connectors having only 9 pins. Using a 25- pin DB-25 or 9-pin DB-9 connector, its normal cable limitation of 50 feet can be extended to several hundred feet with high-quality cable. RS-232 defines the purpose and signal timing for each of the 25 lines; however, many applications use less than a dozen. There is a standardized pinout for RS-232 on a DB25 connector, as shown below.



Pin Number	Signal	Description
1	PG	Protective ground
2	TD	Transmitted data
3	RD	Received data
4	RTS	Request to send
5	CTS	Clear to send
6	DSR	Data set ready
7	SG	Signal Ground
8	CD	Carrier detect
9	+	Voltage (testing)
10	-	Voltage (testing)
11		
12	SCD	Secondary CD
13	SCS	Secondary CTS
14	STD	Secondary TD
15	TC	Transmit Clock
16	SRD	Secondary RD
17	RS	Receiver clock
18		Ready to Send
19	SRS	Secondary RTS
20	DTR	Data Terminal Ready
21	SQD	Signal Quality Detector
22	RI	Ring Indicator
23	DRS	Data rate select
24	XTC	External Clock
25		

25-pin D-type connector Pin assignment

Signal Description

TxD: - This pin carries data from the computer to the serial device

RXD: - This pin carries data from the serial device to the computer

DTR signals: - DTR is used by the computer to signal that it is ready to communicate with the serial device like modem. In other words, DTR indicates to the Dataset (i.e., the modem or DSU/CSU) that the DTE (computer) is ON.

DSR: - Similarly to DTR, Data set ready (DSR) is an indication from the Dataset that it is ON.

DCD: - Data Carrier Detect (DCD) indicates that carrier for the transmit data is ON.

RTS: - This pin is used to request clearance to send data to a modem

CTS: - This pin is used by the serial device to acknowledge the computer's RTS Signal. In most situations, RTS and CTS are constantly on throughout the communication session.

Clock signals (TC, RC, and XTC): - The clock signals are only used for synchronous communications. The modem or DSU extracts the clock from the data stream and provides a steady clock signal to the DTE. Note that the transmit and receive clock signals do not have to be the same, or even at the same baud rate.

CD: - CD stands for Carrier Detect. Carrier Detect is used by a modem to signal that it has a made a connection with another modem, or has detected a carrier tone. In other words, this is used by the modem to signal that a carrier signal has been received from a remote modem.

RI: - RI stands for Ring Indicator. A modem toggles(keystroke) the state of this line when an incoming call rings your phone. In other words, this is used by an auto answer modem to signal the receipt of a telephone ring signal

The Carrier Detect (CD) and the Ring Indicator (RI) lines are only available in connections to a modem. Because most modems transmit status information to a PC when either a carrier signal is detected (i.e. when a connection is made to another modem) or when the line is ringing, these two lines are rarely used.

Limitations of RS-232

RS-232 has some serious shortcomings as an electrical interface.

Firstly, the interface presupposes a common ground between the DTE and DCE. This is a reasonable assumption where a short cable connects a DTE and DCE in the same room, but with longer lines and connections between devices that may be on different electrical busses, this may not be true. We have seen some spectacular electrical events causes by "uncommon grounds".

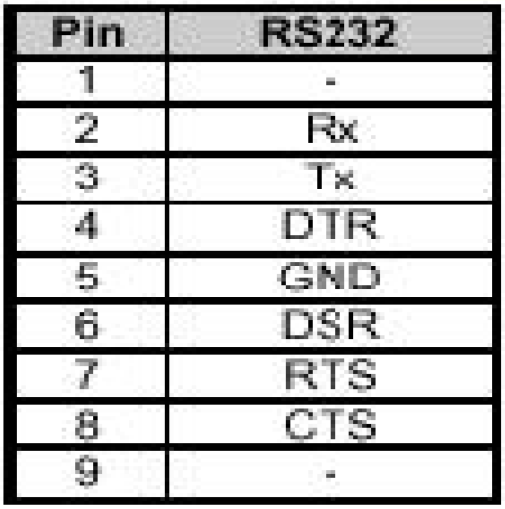
Secondly, a signal on a single line is impossible to screen effectively for noise. By screening the entire cable one can reduce the influence of outside noise, but internally generated noise remains a problem. As the baud rate and line length increase, the effect of capacitance between the cables introduces serious crosstalk until a point is reached where the data itself is unreadable.

Converters

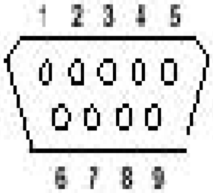
Converters in general can be used to change the electrical characteristic of one communications standard into another, to take advantage of the best properties of the alternate standard selected.

For example, an Automatic RS232<=>RS485 converter, could be connected to a computer's RS232, full-duplex port, and transform it into an RS485 half-duplex, multi-drop network at distances up to 4000ft. Converters in most instances, pass data through the interface without changing the timing and/or protocol. While the conversion is "transparent" the software must be able to communicate with the expanded network features. An "Automatic Converter" (RS232<=>RS485) will turn on the RS485 transmitter when data is detected on the RS232 port, and revert back into the receive mode after a character has been sent.

This avoids timing problems (and software changes) that are difficult to deal with in typical systems. When fullduplex is converted into half-duplex only one device at a time can transmit data. Automatic Converters take care of the timing problems and allow fast communications without software intervention.



Pin	RS232
1	-
2	Rx
3	Tx
4	DTR
5	GND
6	DSR
7	RTS
8	CTS
9	-



Pin assignment of the DB-9 connector

Pin layout of the DB-9 connector

CHAPTER-7

INTRODUCTION TO MICROCONTROLLERS

OVERVIEW OF 8051 MICRO CONTROLLER

Basically used for control actions. It is used to control the operation of machine using fixed program that is stored in ROM/EPROM and that does not change over the life time.The Intel 8051 (Official designation for 8051 family is MCS-51)is a Harvard architecture.A single chip microcontroller(µC) which was developed by Intel in 1980 for use in embedded systems. Intel's original versions were popular in the 1980s and early 1990s, but has today largely been superseded by a vast range of faster and/or functionally enhanced 8051-compatible devices manufactured by more than 20 independent manufacturers including Atmel, Infineon Technologies (formerly Siemens AG), Maxim Integrated Products (via its Dallas Semiconductor subsidiary), NXP (formerly Philips Semiconductor), Nuvoton (formerly Winbond), ST Microelectronics, Silicon Laboratories (formerly Cygnal), Texas Instruments and Cypress Semiconductor.

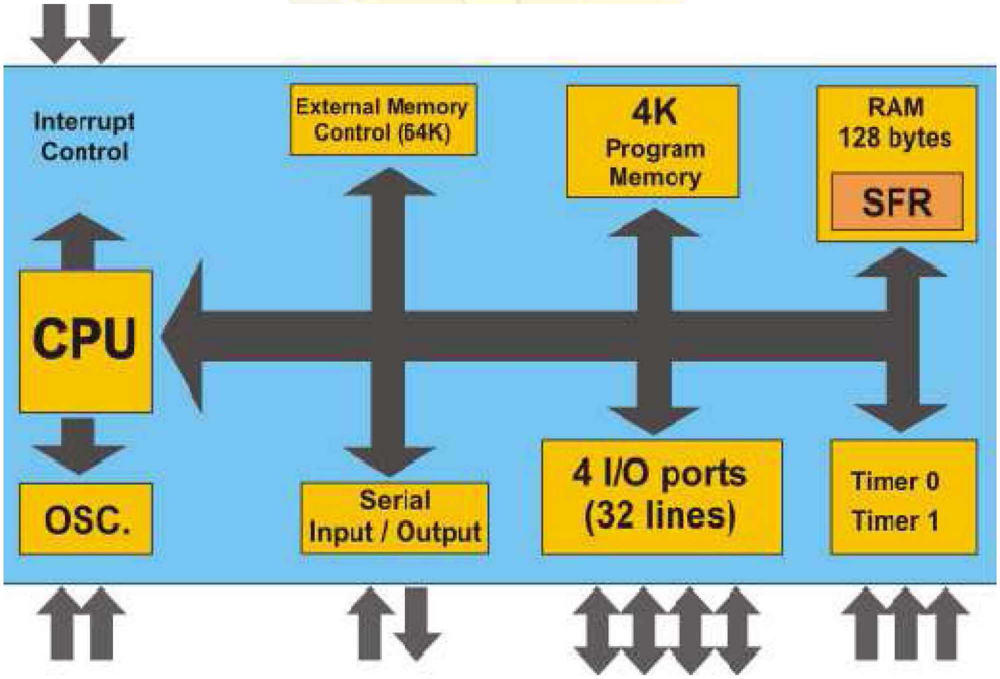
FEATURE OF 8051 MICRO CONTROLLER

- It provides many functions (CPU, RAM, ROM, I/O, interrupt logic, timer, etc.) in a single package.
- 8-bit data bus - It can access 8 bits of data in one operation (hence it is an 8-bit microcontroller).
- 16-bit address bus - 64 kB each of RAM and ROM.
- On-chip RAM - 128 bytes ("DATA Memory").
- On-chip ROM - 4 kB ("CODE (program) Memory").
- Four 8-bit bi-directional input/output ports.
- UART (serial port).
- Two 16-bit Counter/timers.
- Two-level interrupt priority.
- Power saving mod.
- Full duplex serial Port.
- 32 bits arranged as four,8 bit ports P0-P3.
- 16 bit PC and DPTR.
- 8 bit ALU.
- Control Registers are TCON,TMOD,SCON,PCON,IP,IE etc (SFR’s).
- Two External and three internal interrupt sources.
- 0-12 MHz clock.

- Ø 40 pin DIP package.
- Ø Powerful Instruction set.
- Ø Works in Power Down and Idle mod.

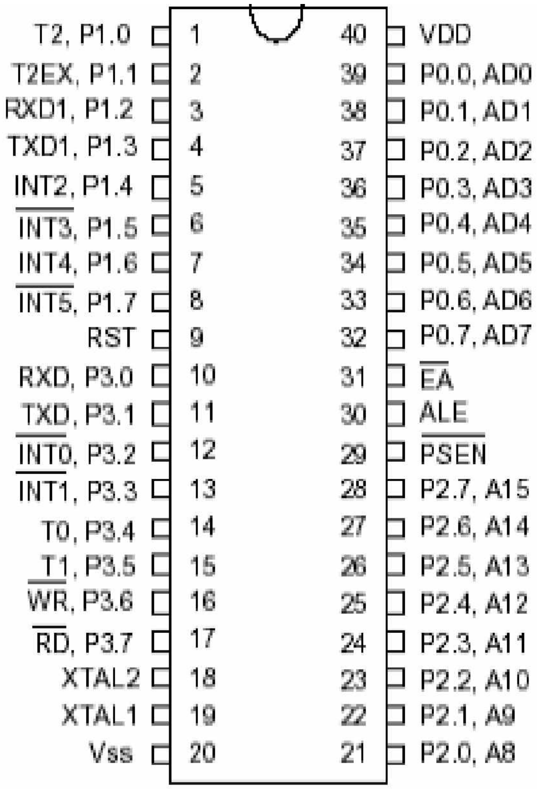
Microcontrollers' producers have been struggling for a long time for attracting more and more choosy customers. Every couple of days a new chip with a higher operating frequency, more memory and more high-quality A/D converters comes on the market.Nevertheless, by analyzing their structure it is concluded that most of them have the same (or at least very similar)architecture known in the product catalogs as "8051 compatible". What is all this about? The whole story began in the far 80s when Intel launched its series of the microcontrollers labeled with MCS 051.Although, several circuits belonging to this series had quite modest features in comparison to the new ones, they took over the world very fast and became a standard for what nowadays is meant by a word microcontroller.

The reason for success and such a big popularity is a skillfully chosen configuration which satisfies needs of a great number of the users allowing at the same time stable expanding (refers to the new types of the microcontrollers). Besides, since a great deal of software has been developed in the meantime, it simply was not profitable to change anything in the microcontroller's basic core. That is the reason for having a great number of various microcontrollers which actually are solely upgraded versions of the 8051 family. What is it what makes this microcontroller so special and universal so that almost all the world producers manufacture it today under different name ?



8051 MICRO CONTROLLER PINS

PIN DIAGRAM OF 8051 MICRO CONTROLLER



Pins 1-8: Port 1 Each of these pins can be configured as input or output.

Pin 9: RS Logical one on this pin stops microcontroller’s operating and erases the contents of most registers. By applying logical zero to this pin, the program starts execution from the beginning. In other words, a positive voltage pulse on this pin resets the microcontroller.

Pins10-17: Port 3 Similar to port 1, each of these pins can serve as universal input or output . Besides, all of them have alternative functions:

Pin 10: RXD Serial asynchronous communication input or Serial synchronous communication output.

Pin 11: TXD Serial asynchronous communication output or Serial synchronous communication clock output.

Pin 12: INT0 Interrupt 0 input

Pin 13: INT1 Interrupt 1 input

Pin 14: T0 Counter 0 clock input

Pin 15: T1 Counter 1 clock input

Pin 16: WR Signal for writing to external (additional) RAM

Pin 17: RD Signal for reading from external RAM

Pin 18, 19: X2, X1 Internal oscillator input and output. A quartz crystal which determines operating frequency is usually connected to these pins. Instead of quartz crystal, the miniature ceramics resonators can be also used for frequency stabilization. Later versions of the microcontrollers operate at a frequency of 0 Hz up to over 50 Hz.

Pin 20: GND Ground

Pin 21-28: Port 2 If there is no intention to use external memory then these port pins are configured as universal inputs/outputs. In case external memory is used then the higher address byte, i.e. addresses A8-A15 will appear on this port.It is important to know that even memory with capacity of 64Kb is not used (i.e. note all bits on port are used for memory addressing) the rest of bits are not available as inputs or outputs.

Pin 29: PSEN If external ROM is used for storing program then it has a logic-0 value every time the microcontroller reads a byte from memory.

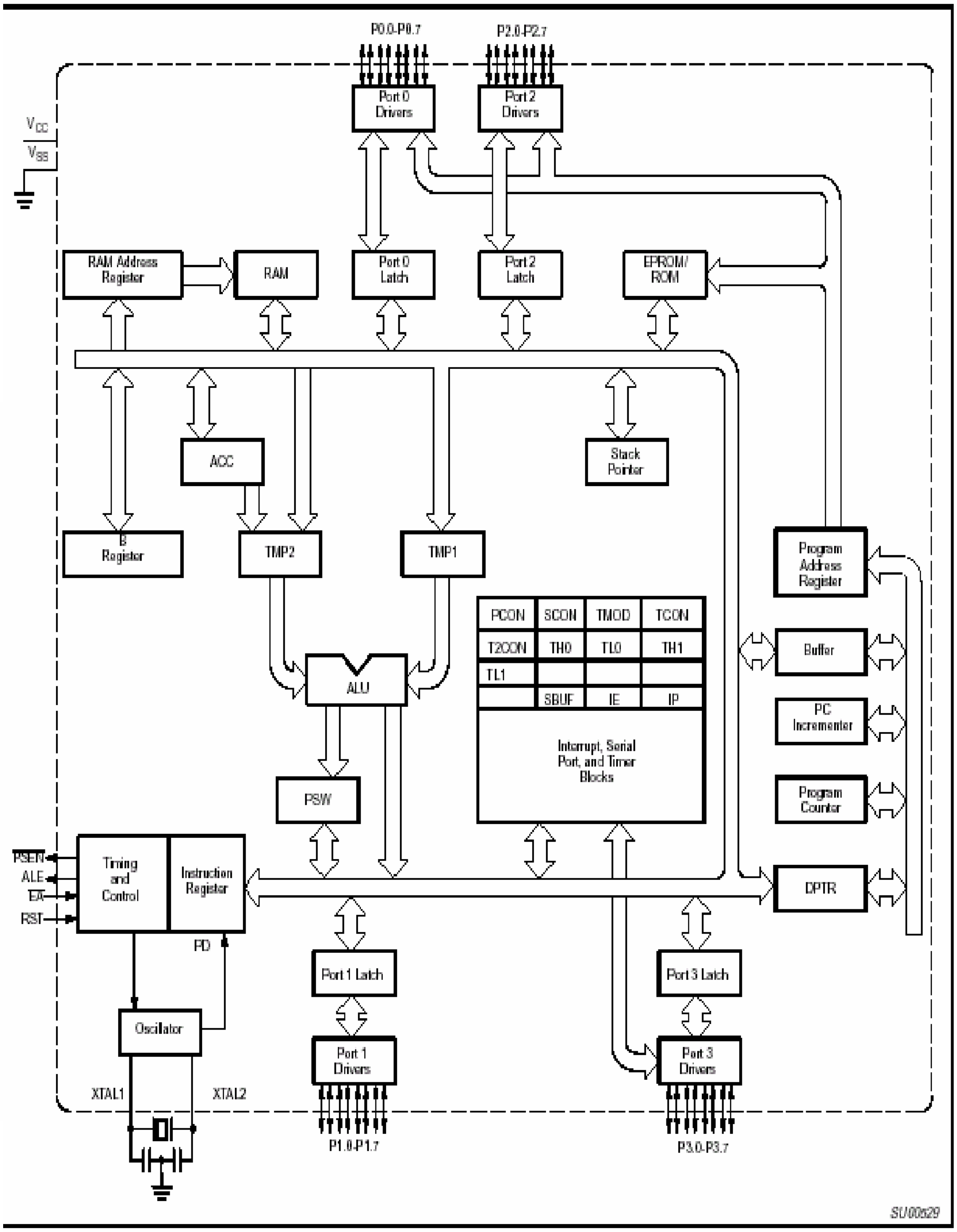
Pin 30: ALE Prior to each reading from external memory, the microcontroller will set the lower address byte (A0-A7) on P0 and immediately after that activates the output ALE. Upon receiving signal from the ALE pin, the external register (74HCT373 or 74HCT375 circuit is usually embedded) memorizes the State of P0 and uses it as an address for memory chip. In the second part of the microcontroller’s machine cycle, a signal on this pin stops being emitted and P0 is used now for data transmission (Data Bus). In this way, by means of only one additional (and cheap) integrated circuit, data multiplexing from the port is performed. This port at the same time used for data and address transmission.

Pin 31: EA By applying logic zero to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. That means that even there is a program written to the microcontroller, it will not be executed, the program written to external ROM will be used instead. Otherwise, by applying logic one to the EA pin, the microcontroller will use both memories, first internal and afterwards external (if it exists), up to End address space.

Pin 32-39: Port 0 Similar to port 2, if external memory is not used, these pins can be used as universal inputs or outputs.Otherwise, P0 is configured as address output (A0-A7) when the ALE pin is at high level (1) and as data output (Data Bus),when logic zero (0) is applied to the ALE pin.

Pin 40: VCC Power supply +5V nd of address space.

BLOCK DIAGRAM OF 8051 MICRO CONTROLLER



REGISTER SET OF 8051

The 8051 is a flexible microcontroller with a relatively large number of modes of operations. Your program may inspect and/or change the operating mode of the 8051 by manipulating the values of the 8051's Special Function Registers (SFRs)…..

SFRs are accessed as if they were normal Internal RAM. The only difference is that Internal RAM is from address 00h through 7Fh whereas SFR registers exist in the address range of 80h through FFh.

SP (Stack Pointer, Address 81h): This is the stack pointer of the microcontroller. This SFR indicates where the next value to be taken from the stack will be read from in Internal RAM. If you push a value onto the stack, the value will be written to the address of SP + 1. That is to say, if SP holds the value 07h, a PUSH instruction will push the value onto the stack at address 08h. This SFR is modified by all instructions which modify the stack, such as PUSH, POP, LCALL, RET, RETI, and whenever interrupts are provoked by the microcontroller.

DPL/DPH (Data Pointer Low/High, Addresses 82h/83h): The SFRs DPL and DPH work together to represent a 16-bit value called the *Data Pointer*. The data pointer is used in operations regarding external RAM and some instructions involving code memory. Since it is an unsigned two-byte integer value, it can represent values from 0000h to FFFFh (0 through 65,535 decimal).

PCON (Power Control, Addresses 87h): The Power Control SFR is used to control the 8051's power control modes. Certain operation modes of the 8051 allow the 8051 to go into a type of "sleep" mode which requires much less power. These modes of operation are controlled through PCON. Additionally, one of the bits in PCON is used to double the effective baud rate of the 8051's serial port.

PCON – Power Control Register

D7	D6	D5	D4	D3	D2	D1	D0
SMOD	X	X	X	GF1	GF0	PD	IDL

If SMOD = 0 then N = 384. If SMOD = 1 then N = 192. TH1 is the high byte of timer 1 when it is in 8-bit autoreload mode.

GF1 and GF0 are General purpose flags not implemented on the standard device

PD is the power down bit. Not implemented on the standard device

IDL activate the idle mode to save power. Not implemented on the standard device

TR1 – Timer 1 run control bit

TF0 – Timer 0 overflow flag

TR0 – Timer 0 run control bit

IE1 – External interrupt 1 edge flag. Set to 1 when edge detected.

IT1 – Edge control bit for external interrupt 1. 1 = edge, 0 = level

TCON (Timer Control, Addresses 88h, Bit-Addressable): The Timer Control SFR is used to configure and modify the way in which the 8051's two timers operate. This SFR controls whether each of the two timers is running or stopped and contains a flag to indicate that each timer has overflowed. Additionally, some non-timer related bits are located in the TCON SFR. These bits are used to configure the way in which the external interrupts are activated and also contain the external interrupt flags which are set when an external interrupt has occured.

TCON – Timer Control Register

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 – Timer 1 overflow flag

TR1 – Timer 1 run control bit

TF0 – Timer 0 overflow flag

TR0 – Timer 0 run control bit

IE1 – External interrupt 1 edge flag. Set to 1 when edge detected.

IT1 – Edge control bit for external interrupt 1. 1 = edge, 0 = level

IE0 – External interrupt 0 edge flag. Set to 1 when edge detectd

IT0 – Edge control bit for external interrupt 0. 1 = edge, 0 = level

TMOD (Timer Mode, Addresses 89h): The Timer Mode SFR is used to configure the mode of operation of each of the two timers. Using this SFR your program may configure each timer to be a 16-bit timer, an 8-bit autoreload timer, a 13-bit timer, or two separate timers. Additionally, you may configure the timers to only count when an external pin is activated or to count "events" that are indicated on an external pin.

TL0/TH0 (Timer 0 Low/High, Addresses 8Ah/8Ch): These two SFRs, taken together, represent timer 0. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value.

TL1/TH1 (Timer 1 Low/High, Addresses 8Bh/8Dh): These two SFRs, taken together, represent timer 1. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value.

SCON (Serial Control, Addresses 98h, Bit-Addressable): The Serial Control SFR is used to configure the behavior of the 8051's on-board serial port. This SFR controls the baud rate of the serial port, whether the serial port is activated to receive data, and also contains flags that are set when a byte is successfully sent or received.

SCON – Serial Control Register

D7	D6	D5	D4	D3	D2	D1	D0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	**SM1**	**Operation**	**Baud rate**
0	0	Shift register	Osc/12
0	1	8-bit UART	Set by timer
1	0	9-bit UART	Osc/12 or Osc/64
1	1	9-bit UART	Set by timer

SM2 – Enables multiprocessor communication in modes 2 and 3.

REN – Receiver enable

TB8 – Transmit bit 8. This is the 9th bit transmitted in modes 2 and 3.

RB8 – Receive bit 8. This is the 9th bit received in modes 2 and 3.

TI – Transmit interrupt flag. Set at end of character transmission. Cleared in software.

RI – Receive interrupt flag. Set at end of character reception. Cleared in software.

SBUF (Serial Control, Addresses 99h): The Serial Buffer SFR is used to send and receive data via the on-board serial port. Any value written to SBUF will be sent out the serial port's TXD pin. Likewise, any value which the 8051 receives via the serial port's RXD pin will be delivered to the user program via SBUF. In other words, SBUF serves as the output port when written to and as an input port when read from.

IE (Interrupt Enable, Addresses A8h): The Interrupt Enable SFR is used to enable and disable specific interrupts. The low 7 bits of the SFR are used to enable/disable the specific interrupts, where as the highest bit is used to enable or disable ALL interrupts. Thus, if the high bit of IE is 0 all interrupts are disabled regardless of whether an individual interrupt is enabled by setting a lower bit.

D7	D6	D5	D4	D3	D2	D1	D0
EA	X	ET2	ES	ET1	EX1	ET0	EX0

EA – Global interrupt enable

x – not defined

ET2 – Timer 2 interrupt enable

ES – Serial port interrupt enable

ET1 – Timer 1 interrupt enable

EX1 – External interrupt 1 enable

ET0 – Timer 0 interrupt enable

EX0 – External interrupt 0 enable

IP (Interrupt Priority, Addresses B8h, Bit-Addressable): The Interrupt Priority SFR is used to specify the relative priority of each interrupt. On the 8051, an interrupt may either be of low (0) priority or high (1) priority. An interrupt may only interrupt interrupts of lower priority. For example, if we configure the 8051 so that all interrupts are of low priority except the serial interrupt, the serial interrupt will always be able to interrupt the system, even if another interrupt is currently executing. However, if a serial interrupt is executing no other interrupt will be able to interrupt the serial interrupt routine since the serial interrupt routine has the highest priority.

D7	D6	D5	D4	D3	D2	D1	D0
X	X	PT2	PS	PT1	PX1	PT0	PX0

x – not defined

PT2 – Priority for timer 2 interrupt

PS – Priority for serial port interrupt

PT1 – Priority for timer 1 interrupt

PX1 – Proiority for external interrupt 1

PT0 – Priority for timer 0 interrupt

PX0 – Priority for external interrupt 0

ACC (Accumulator, Addresses E0h, Bit-Addressable): The Accumulator is one of the most-used SFRs on the 8051 since it is involved in so many instructions. The Accumulator resides as an SFR at E0h, which means the instruction **MOV A,#20h** is really the same as **MOV E0h,#20h**. However, it is a good idea to use the first method since it only requires two bytes whereas the second option requires three bytes.

B (B Register, Addresses F0h, Bit-Addressable): The "B" register is used in two instructions: the multiply and divide operations. The B register is also commonly used by programmers as an auxiliary register to temporarily store values

PSW (Program Status Word, Addresses D0h, Bit-Addressable):

The **register PSW** *(Program Status Word)* contains information on the status of the CPU. Contains indicators or *flags* to use conditional statements to make decisions. These indicators are changed automatically when any of the instructions shown in the following table is executed. The Program Status Word is used to store a number of important bits that are set and cleared by 8051 instructions. The PSW SFR contains the carry flag, the auxiliary carry flag, the overflow flag, and the parity flag. Additionally, the PSW register contains the register bank select flags which are used to select which of the "R" register banks are currently selected.

D7 D6 D5 D4 D3 D2 D1
D0

C	AC	FO	RS1	RS0	OV	X	P

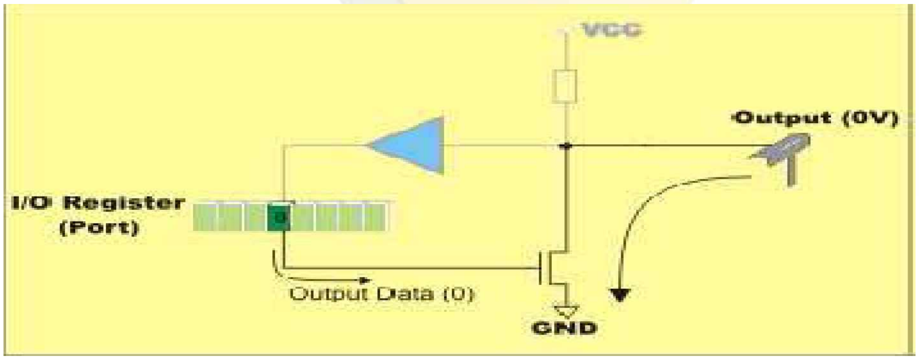
Indicators or *flags*

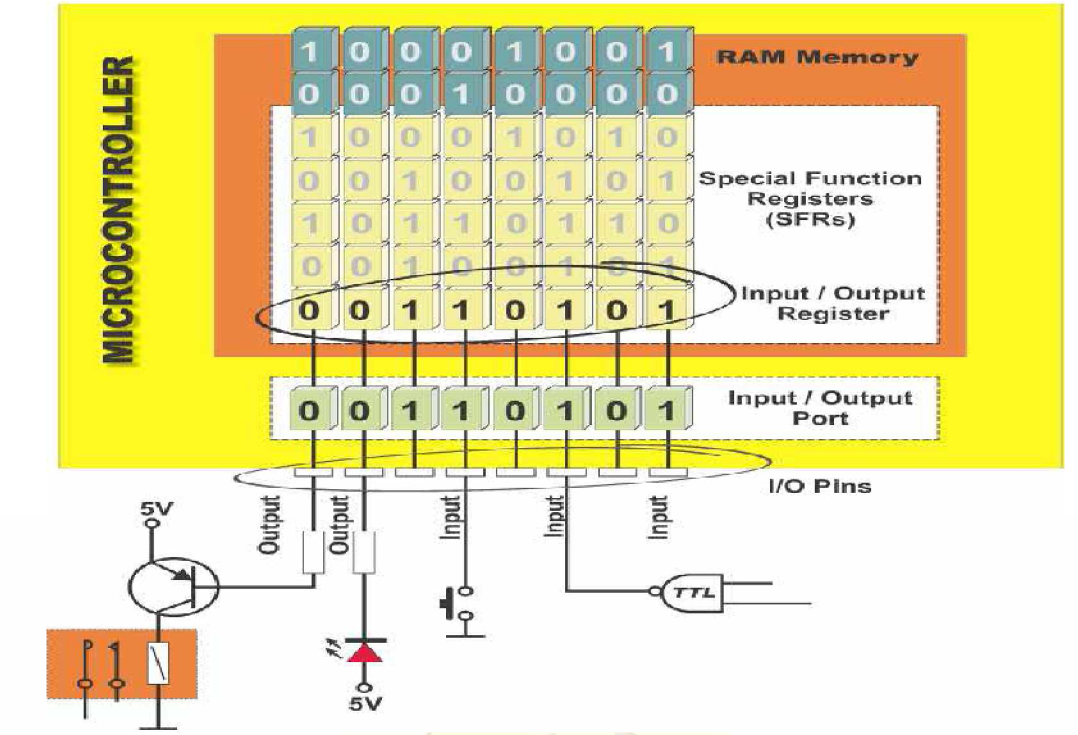
- **C:** Carry Flag.
- **AC:** Auxiliary Carry Flag indicates the carry from bit 3, is used for BCD operations.
- **F0:** User Indicator or general purpose.
- **Ov:** overflow indicator, when a drift in the 6 th and 7 th bit at a time.
- **P:** parity indicator indicates 1 when the number is odd about the Acc.
- **Rs $_0$** and **Rs $_1$:** Selection of bank records.

Rs1	Rs0	Bank	Address
0	0	0	00H to 07H
0	1	1	08H to 0FH
1	0	2	10H 17H
1	1	3	18H to 1FH

INPUT/OUTPUT PORTS OF 8051

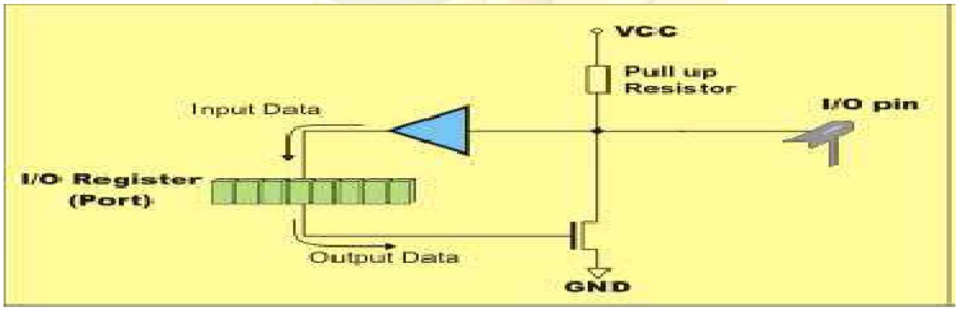
All 8051 microcontrollers have 4 I/O ports, each consisting of 8 bits which can be configured as inputs or outputs. This means that the user has on disposal in total of 32 input/output lines connecting the microcontroller to peripheral devices.A logic state on a pin determines whether it Is configured as input or output: 0=output, 1=input. If a pin on the microcontroller needs to be configured as output, then a logic zero (0) should be applied to the appropriate bit on I/O port.In his way, a voltage level on the appropriate pin will be 0.Similar to that, if a pin needs to be configured as input, then a logic one (1) should be applied to the appropriate port. In this way, as a side effect a voltage level on the appropriate pin will be 5V (as it is case with any TTL input). This may sound a bit confusing but everything becomes clear after studying a simplified electronic circuit connected to one I/O pin.





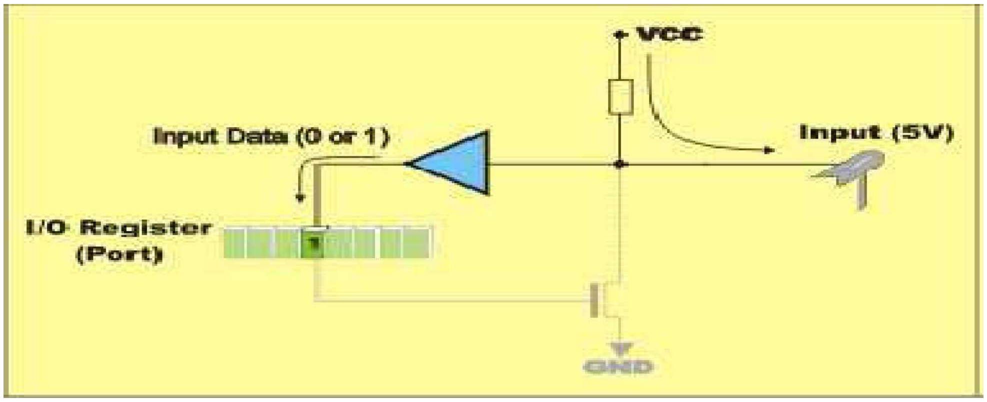
Input/Output (I/O) pin

This is a simplified overview of what is connected to a pin inside the microcontroller. It concerns all pins except those included in P0 which do not have embedded pullup resistor.



Input Output pin

A logic zero (0) is applied to a bit in the P register. By turning output FE transistor on, the appropriate pin is directly connected to ground.



INPUT PIN

A logic one (1) is applied to a bit in the P register.Output FE transistor is turned off. The appropriate pin remains connected to voltage power supply through a pull-up resistor of high resistance.

PORT0

It is specific to this port to have a double purpose. If external memory is used then the lower address byte (addresses A0-A7) is applied on it. Otherwise, all bits on this port are configured as inputs or outputs.Another characteristic is expressed when it is configured as output. Namely, unlike other ports consisting of pins with embedded pull-up resistor (connected by its end to 5 V power supply), this resistor is left out here. This, apparently little change has its consequences:

If any pin on this port is configured as input then it performs as if it “floats”. Such input has unlimited input resistance and has no voltage coming from “inside”.

When the pin is configured as output, it performs as “open drain”,meaning that by writing 0 to some port’s bit, the appropriate pin will be connected to ground (0V). By writing 1, the external output will keep on “floating”. In order to apply 1 (5V) on this output, an external pull-up resistor must be embedded.

PORT1

This is a true I/O port, because there are no role assigning as it is the case with P0. Since it has embedded pull-up resistors it is completely compatible with TTL circuits.

PORT 2

Similar to P0, when using external memory, lines on this port occupy addresses intended for external memory chip. This time it is the higher address byte with addresses A8-A15. When there is no additional memory, this port can be used as universal input-output port similar by its features to the port 1.

PORT3

Even though all pins on this port can be used as universal I/O port, they also have an alternative function. Since each of these functions use inputs, then the appropriate pins have to be configured like that. In other words, prior to using some of reserve port functions, a logical one (1) must be written to the appropriate bit in the P3 register. From hardware’s perspective , this port is also similar to P0, with the difference that its outputs have a pull-up resistor embedded.

CURRENT LIMITATIONS OF PINS

When configured as outputs (logic zero (0)), single port pins can "receive" current of 10mA. If all 8 bits on a port are active, total current must be limited to 15mA (port P0: 26mA). If all ports (32 bits) are active, total maximal current must be limited to 71mA. When configured as inputs (logic 1), embedded pull-up resistor provides very weak current, but strong enough to activate up to 4 TTL inputs from LS series.

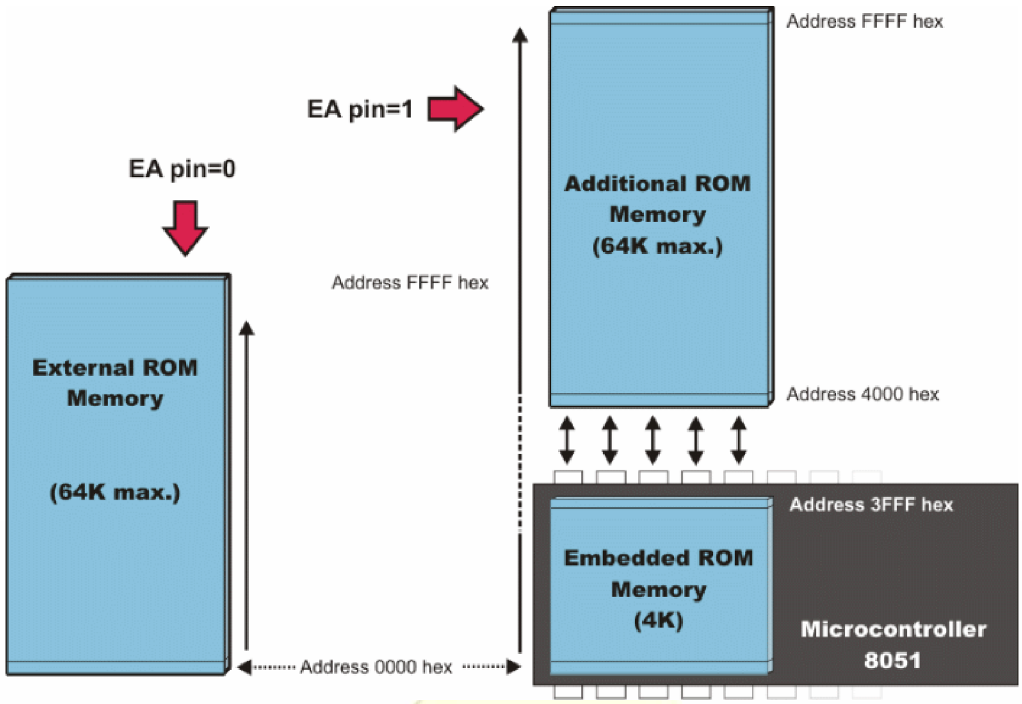
8051 MICRO CONTROLLER MEMORY ORGANIZATION

The microcontroller memory is divided into Program Memory and Data Memory. Program Memory (ROM) is used for permanent saving program being executed, while Data Memory (RAM) is used for temporarily storing and keeping intermediate results and variables. Depending on the model in use (still referring to the whole 8051 microcontroller family)at most a few Kb of ROM and 128 or 256 bytes of RAM can be used. However…All 8051 microcontrollers have 16-bit addressing bus and can address 64 kb memory. It is neither a mistake nor a big ambition of engineers who were working on basic core development. It is a matter of very clever memory organization which makes these controllers a real “ programmers” Tibbited.

PROGRAM MEMORY (ROM)

Program Memory (ROM) is used for permanent saving program (CODE) being executed. The memory is read only. Depending on the settings made in compiler, program memory may also used to store a constant variables. The 8051 executes programs stored in program memory only. code memory type specifier is used to refer to program memory.

8051 memory organization alows external program memory to be added. How does the microcontroller handle external memory depends on the pin EA logical state.



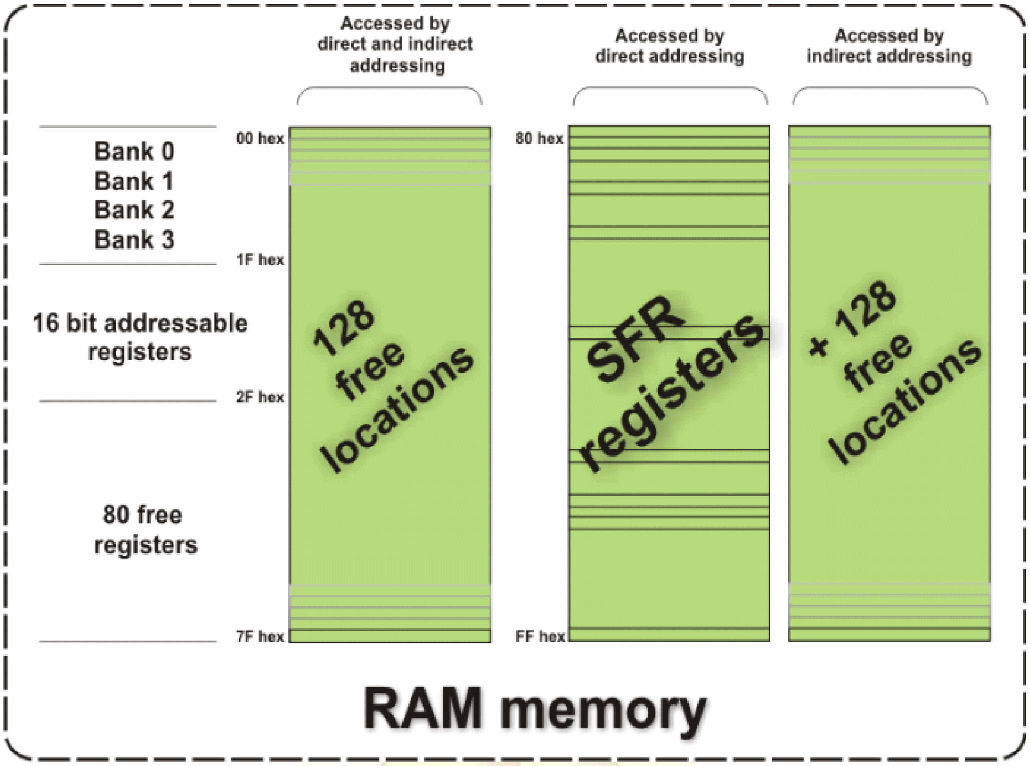
DATA MEMORY

Internal Data Memory

Up to 256 bytes of internal data memory are available depending on the 8051 derivative. Locations available to the user occupy addressing space from 0 to 7Fh, i.e. first 128 registers and this part of RAM is divided in several blocks. The first 128 bytes of internal data memory are both directly and indirectly addressable. The upper 128 bytes of data memory (from 0x80 to 0xFF) can be addressed only indirectly.

Since internal data memory is used for CALL stack also and there is only 256 bytes splited over few different memory areas fine utilizing of this memory is crucial for fast and compact code.

The first block consists of 4 banks each including 8 registers designated as R0 to R7. Prior to access them, a bank containing that register must be selected. Next memory block (in the range of 20h to 2Fh) is bit- addressable, which means that each bit being there has its own address from 0 to 7Fh. Since there are 16 such registers, this block contains in total of 128 bits with separate addresses (The 0th bit of the 20h byte has the bit address 0 and the 7th bit of th 2Fh byte has the bit address 7Fh). The third group of registers occupy addresses 2Fh-7Fh (in total of 80 locations) and does not have any special purpose or feature.



External Data Memory

Access to external memory is slower than access to internal data memory. There may be up to 64K Bytes of external data memory. Several 8051 devices provide on-chip XRAM space that is accessed with the same instructions as the traditional external data space. This XRAM space is typically enabled via proper setting of SFR register and overlaps the external memory space. Setting of that register must be manualy done in code, before any access to external memory or XRAM space is made.

SFR Memory

The 8051 provides 128 bytes of memory for Special Function Registers (SFRs). SFRs are bit, byte, or word-sized registers that are used to control timers, counters, serial I/O, port I/O, and peripherals.

ADDRESSING MODES OF 8051

While operating, processor processes data according to the program instructions. Each nstruction consists of two parts.One part describes what should be done and another part indicates what to use to do it. This later part can be data (binary number) or address where the data is stored. All 8051 microcontrollers use two ways of addressing depending on which part of memory should be accessed:

DIRECT ADDRESSING

In direct addressing, you specify the address to operate in absolute terms. For the family of 8051 microcontrollers are available direct address 256, corresponding to (internal + Ram Records SFR). The *OpCode* is followed by a byte representing the address.

On direct addressing, a value is obtained from a memory location while the address of that location is specified in instruction. Only after that, the instruction can process data (howdepends on the type of instruction: addition, subtraction,copy…). Obviously, a number being changed during operating a variable can reside at that specified address. For example:Since the address is only one byte in size (the greatest number is 255), this is how only the first 255 locations in RAM can be accessed in this case the first half of the basic RAM is intended to be used freely, while another half is reserved for the SFRs.

MOV A,33h; Means: move a number from address 33 hex. to accumulator

INDIRECT ADDRESSING

On indirect addressing, a register which contains address of another register is specified in the instruction. A value used in operating process resides in that another register. For example:

Only RAM locations available for use are accessed by indirect addressing (never in the SFRs). For all latest versions of the microcontrollers with additional memory block (those 128 locations in Data Memory), this is the only way of accessing them. Simply, when during operating, the instruction including “@” sign is encountered and if the specified address is higher than 128 (7F hex.), the processor knows that indirect addressing is used and jumps over memory space Reserved for the SFRs.

MOV A,@R0; Means: Store the value from the register whose address is in the R0 register into accumulator

THE 8051 INSTRUCTION SETS

Writing program for the microcontroller mainly consists of giving instructions (commands) in that order in which they should be executed later in order to carry out specific task. As Electronics can not “understand” what for example instruction “if the push button is pressed- turn the light on” means, then a certain number of more simpler and precisely defined orders that decoder can recognise must be used. All commands are known as INSTRUCTION SET. All microcontrollers compatibile with the 8051 have in total of 255 instructions, i.e. 255 different words available for program writing.

TYPES OF INSTUCTION SETS

Depending on operation they perform, all instructions are divided in several groups:

- Arithmetic Instructions
- Branch Instructions
- Data Transfer Instructions
- Logical Instructions
- Logical Instructions with bits

ARITHMETIC INSTRUCTIONS

These instructions perform several basic operations (addition, subtraction, division, multiplication etc.) After execution,the result is stored in the first operand.

For example:

ADD A,R1 - The result of addition (A+R1) will be stored in the accumulator.

Examples

ADD A,Rn	Add R Register to accumulator
ADD A,Rx	Add directly addressed Rx Register to accumulator
ADD A,@Ri	Add indirectly addressed Register to accumulator
ADD A,#X	Add number X to accumulator

BRANCH INSTRUCTIONS

There are two kinds of these instructions:Unconditional jump instructions: after their execution a jump to a new location from where the program continues execution is executed.Conditional Jump instructions: if some condition is met - a jump is executed. Otherwise, the program normally proceeds with the next instruction.

Examples

ACALL adr11 Call subroutine located at addreess within 2 K byte Program Memory space LCALL adr16 Call subroutine located at any address within 64 K byte Program Memory space RET Return from subroutine

RETI Return from interrupt routine

DATA TRANSFER INSTRUCTIONS

These instructions move the content of one register to another one. The register which content is moved remains unchanged. If they have the suffix “X” (MOVX), the data is exchanged with external memory.

Examples

MOV A,Rn	Move R register to accumulator
MOV A,Rx	Move directly addressed Rx register to accumulator
MOV A,@Ri	Move indirectly addressed register to accumulator
MOV A,#X	Move number X to accumulator
MOV Rn,A	Move accumulator to R register
MOV Rn,Rx	Move directly addressed Rx register to R register

LOGICAL INSTRUCTIONS

These instructions perform logical operations between corresponding bits of two registers. After execution, the result is stored in the first operand.

Examples

ANL A,Rn	Logical AND between accumulator and R register
ANL A,Rx	Logical AND between accumulator and directly addressed register Rx
ANL A,@Ri	Logical AND between accumulator and indirectly addressed register
ANL A,#X	Logical AND between accumulator and number X
ANL Rx,A	Logical AND between accumulator and directly addressed register Rx
ANL Rx,#X	Logical AND between directly addressed register Rx and number X
ORL A,Rn	Logical OR between accumulator

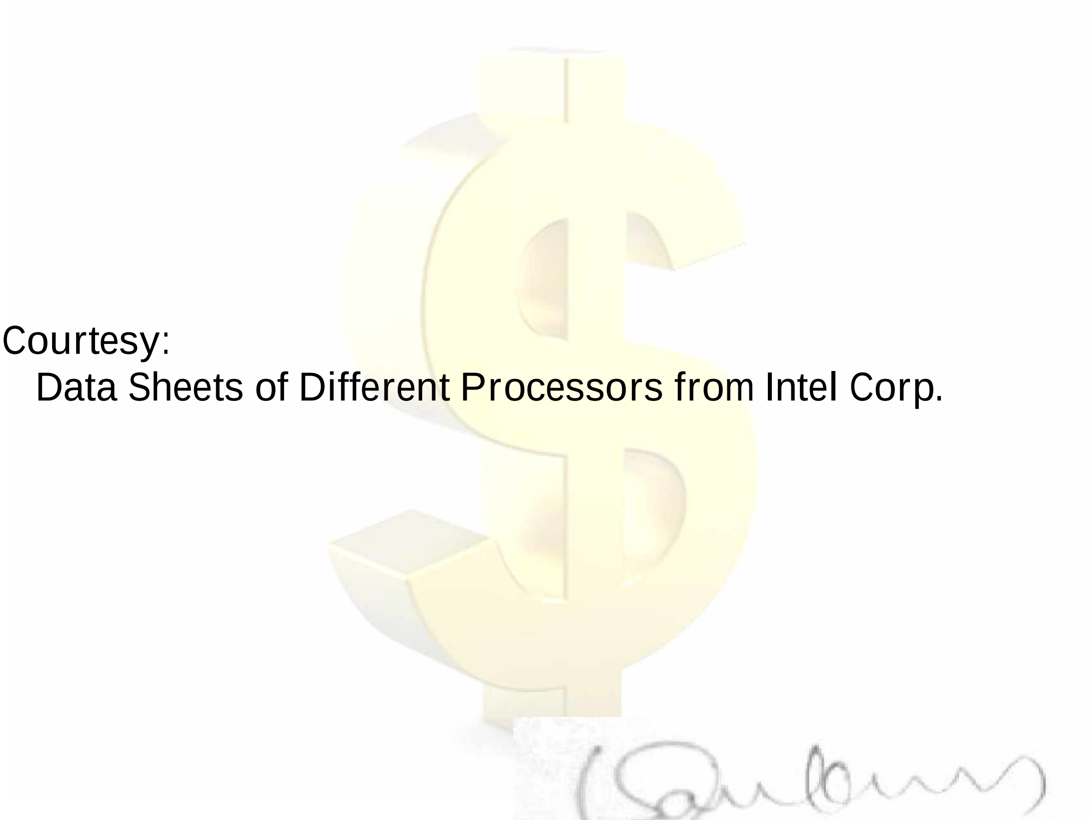
LOGICAL OPERATIONS ON BITS

Similar to logical instructions, these instructions perform logical operations. The difference is that these operations are performed on single bits.

Examples

CLR C	Clear Carry bit
CLR bit	Clear directly addressed bit
SETB C	Set Carry bit
SETB bit	Set directly addressed bit
CPL C	Complement Carry bit

CPL bit	Complement directly addressed bit
ANL C,	bit Logical AND between Carry bit and directly addressed bit
ANL C,/bit	Logical AND between Carry bit and inverted directly addressed bit
ORL C,bit	Logical OR between Carry bit and directly addressed bit



Courtesy:

Data Sheets of Different Processors from Intel Corp.

(Umsankar.ch)