NARSIMHA REDDY ENGINEERING COLLEGE
UGC AUTONOMOUS INSTITUTION
Maisammaguda (V), Kompally - 500100, Secunderabad, Telangana State, India
Your roots to success....
UGC - Autonomous Institute
Accredited by NBA & NAAC with 'A' Grade
Approved by AICTE
Permanently affiliated to JNTUH

MICROCONTROLLERS (23EC501)

Mrs. K V Siva Nagalakshmi
Asst Professor
Dept of ECE

23EC501: MICROCONTROLLERS

B.Tech. III Year I Semester

L	T	P	C
3	1	0	4

Prerequisite: Nil

Course Objectives:

1. To familiarize the architecture of Microprocessors and Microcontroller
2. To provide the knowledge about interfacing techniques of bus & memory.
3. To understand the concepts of ARM architecture.
4. To study the basic concepts of Advanced ARM processors,

Course Outcomes (CO's)

1. Understands the internal architecture, organization and assembly language programming of8086
2. Understands the internal architecture, organization and assembly language programming of8051/controllers
3. Understands the interfacing techniques to 8086 and 8051 based systems.
4. Understands the internal architecture of ARM processors
5. Understands the basic concepts of advanced ARM processors.

Course	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11
CO1	3	3	-	2	-	-	-	-	-	-	-
CO2	3	3	-	2	-	-	-	-	-	-	-
CO3	3	3		2	-	-	-	-	-	-	-
CO4	3	3	3	2	-	-	-	-	-	-	-
CO5	3	3	3	2	-	-	-	-	-	-	-

UNIT -1

8086 Architecture

8086 Architecture-Functional diagram, Register Organization, Memory Segmentation, Programming Model, Memory addresses, Physical Memory Organization, Architecture of 8086, Signal descriptions of 8086, interrupts of 8086. Instruction Set and Assembly Language Programming of 8086: Instruction formats, Addressing modes, Instruction Set, Assembler Directives, Macros, and Simple Programs involving Logical, Branch and Call Instructions, Sorting, String Manipulations.

UNIT-II

Introduction to Microcontrollers

Overview of 8051 Microcontroller, Architecture, I/O Ports, Memory Organization, Addressing Modes and Instruction set of 8051. 8051 Real Time Control: Programming Timer Interrupts. Programming External Hardware Interrupts, Programming the Serial Communication Interrupts, Programming 8051 Timers and Counters.

UNIT-III

I/O and Memory Interface LCD, Keyboard, External Memory RAM, ROM Interface, ADC,DAC Interface to 8051. Serial Communication and Bus Interface: Serial Communication Standards, Serial Data Transfer Scheme, On board Communication Interfaces-12C Bus, SPI Bus, UART: External Communication Interfaces-RS232, USB.

UNIT-IV

ARM Architecture

ARM Processor fundamentals, ARM Architecture - Register, CPSR, Pipeline, exceptions and interrupts interrupt vector table, ARM instruction set - Data processing. Branch instructions, load store instructions, Software interrupt instructions, Program status register instructions, loading constants, Conditional execution, Introduction to Thumb instructions.

UNIT -V

Advanced ARM Processors

Introduction to CORTEX Processor and its architecture, OMAP Processor and its Architecture.

TEXT BOOKS:

1. A. K. Ray and K. M. Bhurchandani -Advanced Microprocessors and Peripherals, TMH, 2nd Edition 2006.
2. Andrew N SLOSS, Dominic SYMES, Chris WRIGHT -ARM System Developers guide, Elsevier, 2012

REFERENCE BOOKS:

1. Kenneth. J. Ayala-The 8051 Microcontroller, Cengage Learning, 3rd Ed, 2004.
2. D. V. Hall -Microprocessors and Interfacing, TMGH, 2nd Edition, 2006.
3. K. Uma Rao, Andhe Pallavi-The 8051 Microcontrollers, Architecture and Programming and Applications, Pearson, 2009.
4. Donald Reay-Digital Signal Processing and Applications with the OMAP- L138 Experimenter, WILEY 2012.

NAGALAKSHMI, Asst Prof

UNIT -1
8086 MIROPROCESSOR

Features of 8086

- It is a 16-bit Microprocessor (µp).It's ALU, internal registers works with 16bit binary word.
- 8086 has a 20 bit address bus can access up to 2^{20}= 1 MB memory locations.
- 8086 has a 16bit data bus. It can read or write data to a memory/port either 16bits or 8 bit at a time.
- It can support up to 64K I/O ports.
- It provides 14, 16 -bit registers.
- Frequency range of 8086 is 6-10 MHz
- 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.
- It can prefetch 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.
- 8086 is designed to operate in two modes, Minimum mode and Maximum mode.
    - 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 / MX# input pin. This is a multi micro processors configuration.

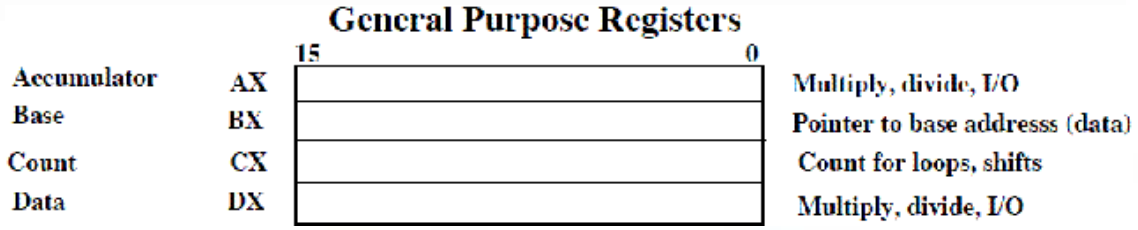
Register Organization of 8086

General purpose registers

The 8086 microprocessor has a total of fourteen registers that are accessible to the programmer. It is divided into four groups. They are:

- Four General purpose registers
- Four Index/Pointer registers
- Four Segment registers
- Two Other registers

General purpose registers:



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.

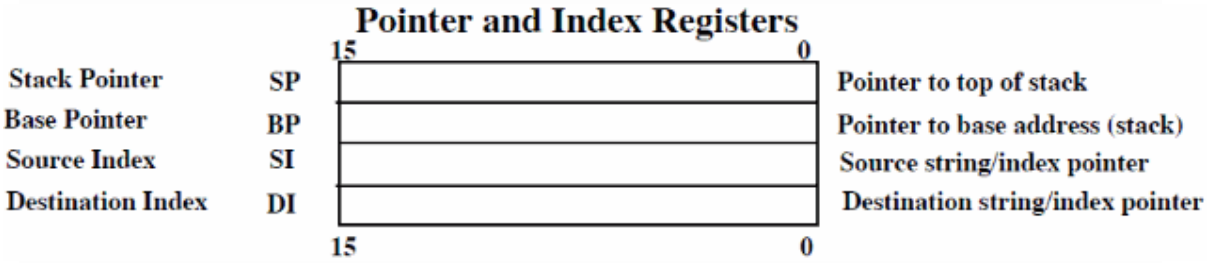
Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register 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.

Index or Pointer Registers

These registers can also be called as Special Purpose registers.



Stack Pointer (SP) is a 16-bit register pointing to program stack, i.e. it is used to hold the address of the top of stack. The stack is maintained as a LIFO with its bottom at the start of the stack segment (specified by the SS segment register).Unlike the SP register, the BP can be used to specify the offset of other program segments.

Base Pointer (BP) is a 16-bit register pointing to data in stack segment. It is usually used by subroutines to locate variables that were passed on the stack by a calling program. BP register is usually used for based, based indexed or register indirect addressing.

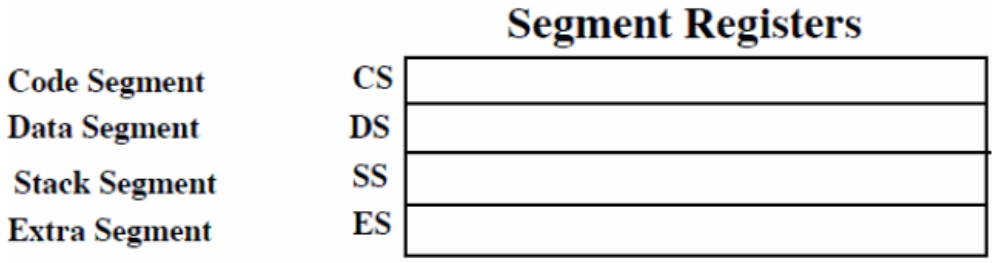
Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions. Used in conjunction with the DS register to point to data locations in the data segment.

Destination Index (DI) is a 16-bit register. Used in conjunction with the ES register in string operations. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions. In short, Destination Index and SI Source Index registers are used to hold address.

Segment Registers

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.



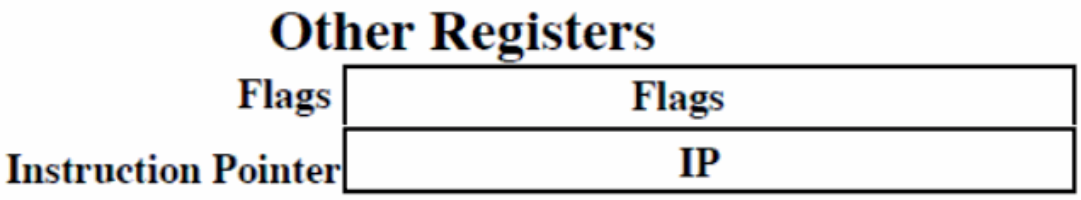
Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during 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.

Extra segment (ES) used to hold the starting address of Extra segment. Extra segment is provided for programs that need to access a second data segment. Segment registers cannot be used in arithmetic operations.

Other registers of 8086



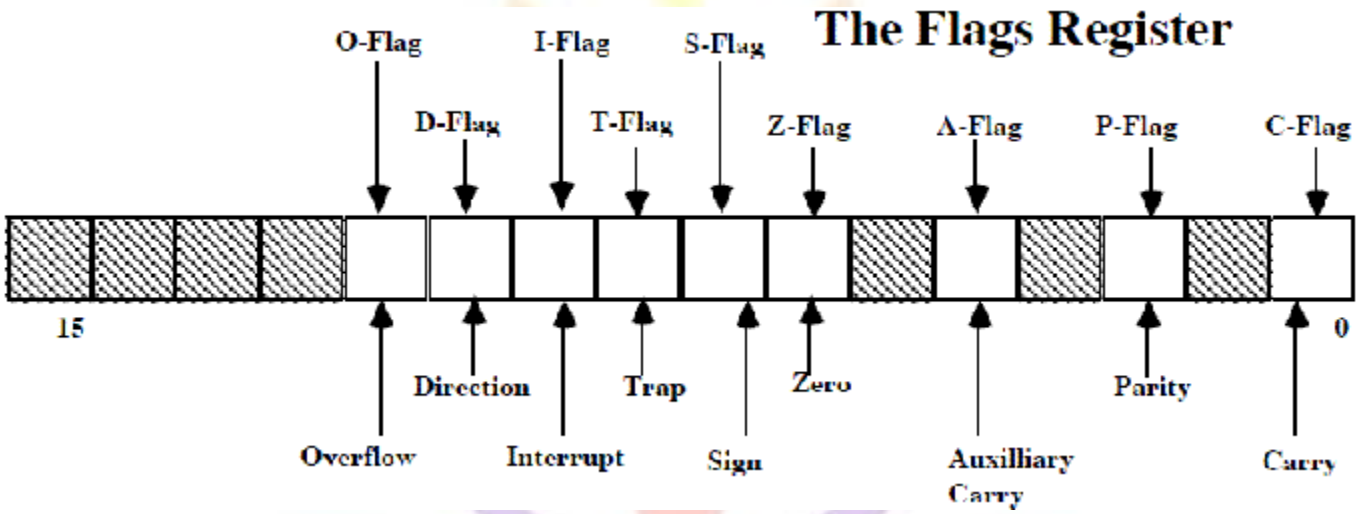
Instruction Pointer (IP) is a 16-bit register. This is a crucially important register which is used to control which instruction the CPU executes. The IP, or program counter, is used to store the memory location of the next instruction to be executed. The CPU checks the program counter to ascertain which instruction to carry out next. It then updates the program counter to point to the next instruction. Thus the program counter will always point to the next instruction to be executed.

Flag Register contains a group of status bits called flags that indicate the status of the CPU or the result of arithmetic operations. There are two types of flags:

1. The **status flags** which reflect the result of executing an instruction. The programmer cannot set/reset these flags directly.
2. The **control flags** enable or disable certain CPU operations. The programmer can set/reset

these bits to control the CPU's operation.

Nine individual bits of the status register are used as control flags (3 of them) and status flags (6 of them).The remaining 7 are not used.
A flag can only take on the values 0 and 1. We say a flag is set if it has the value 1.The status flags are used to record specific characteristics of arithmetic and of logical instructions.



Control Flags: There are three control flags

1. **The Direction Flag (D):** Affects the direction of moving data blocks by such instructions as MOVS, CMPS and SCAS. The flag values are 0 = up and 1 = down and can be set/reset by the STD (set D) and CLD (clear D) instructions.

2. **The Interrupt Flag (I):** Dictates whether or not system interrupts can occur. Interrupts are actions initiated by hardware block such as input devices that will interrupt the normal execution of programs. The flag values are 0 = disable interrupts or 1 = enable interrupts and can be manipulated by the CLI (clear I) and STI (set I) instructions.

3. **The Trap Flag (T):** Determines whether or not the CPU is halted after the execution of each instruction. When this flag is set (i.e. = 1), the programmer can single step through his program to debug any errors. When this flag = 0 this feature is off. This flag can be set by the INT 3 instruction.

Status Flags: There are six status flags

1. **The Carry Flag (C):** This flag is set when the result of an unsigned arithmetic operation is too large to fit in the destination register. This happens when there is an end carry in an addition operation or there an end borrows in a subtraction operation. A value of 1 = carry and 0 = no carry.

2. **The Overflow Flag (O):** This flag is set when the result of a signed arithmetic operation is too large to fit in the destination register (i.e. when an overflow occurs). Overflow can occur when adding two numbers with the same sign (i.e. both positive or both negative). A value of 1 =

overflow and 0 = no overflow.

3. **The Sign Flag (S):** This flag is set when the result of an arithmetic or logic operation is negative. This flag is a copy of the MSB of the result (i.e. the sign bit). A value of 1 means negative and 0 = positive.

4. **The Zero Flag (Z):** This flag is set when the result of an arithmetic or logic operation is equal to zero. A value of 1 means the result is zero and a value of 0 means the result is not zero.

5. **The Auxiliary Carry Flag (A):** This flag is set when an operation causes a carry from bit 3 to bit 4 (or a borrow from bit 4 to bit 3) of an operand. A value of 1 = carry and 0 = no carry.

6. **The Parity Flag (P):** This flags reflects the number of 1s in the result of an operation. If the number of 1s is even its value = 1 and if the number of 1s is odd then its value = 0.

Architecture of 8086 or Functional Block diagram of 8086

- 8086 has two blocks Bus Interface Unit (BIU) and Execution Unit (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.

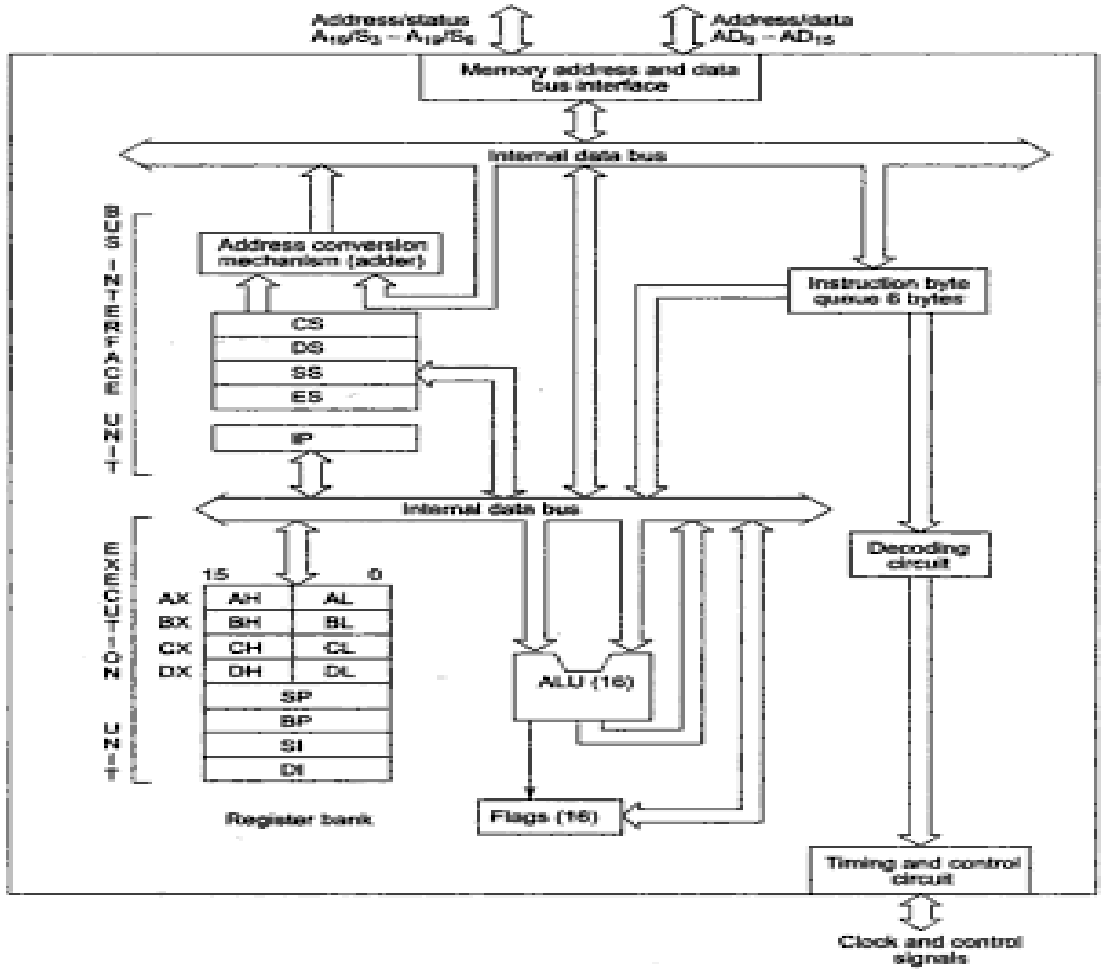


Figure: 8086

Architecture Explanation of Architecture of 8086

Bus Interface 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.

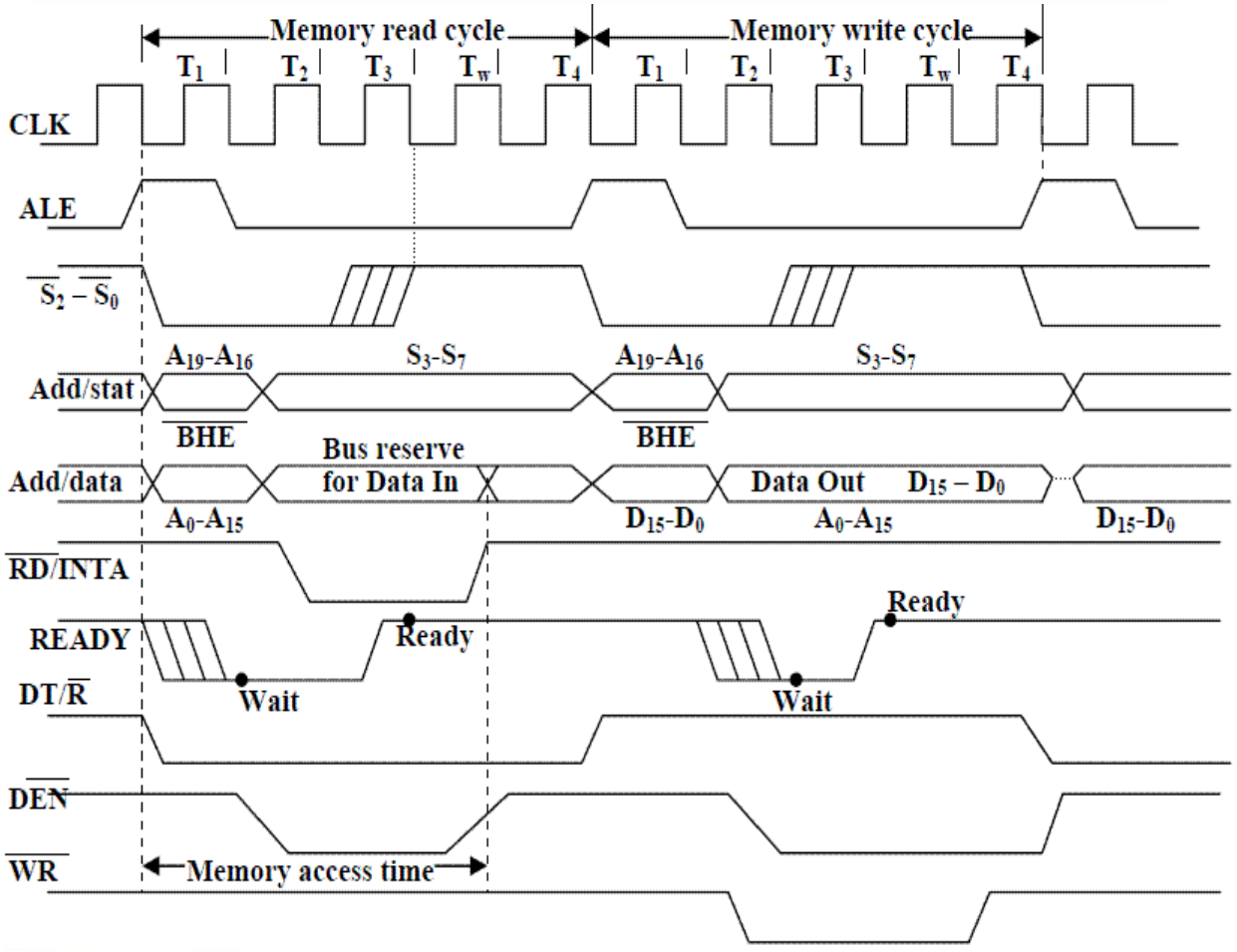
- These prefetching instructions are held in its FIFO queue. With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle.
- After a byte is loaded at the input end of the queue, it automatically shifts up through the FIFO to the empty location nearest the output.
- The EU 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.

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 bus 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 automatically resets the queue and then begins to fetch instructions from this new location to refill the queue.

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 utilization of processor pins and it facilitates the use of 40 pin standard DIP package.
- The bus can be demultiplexed using a few latches and transceivers, when ever required.
- Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T1, T2, T3, and 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.



Maximum mode

- 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.

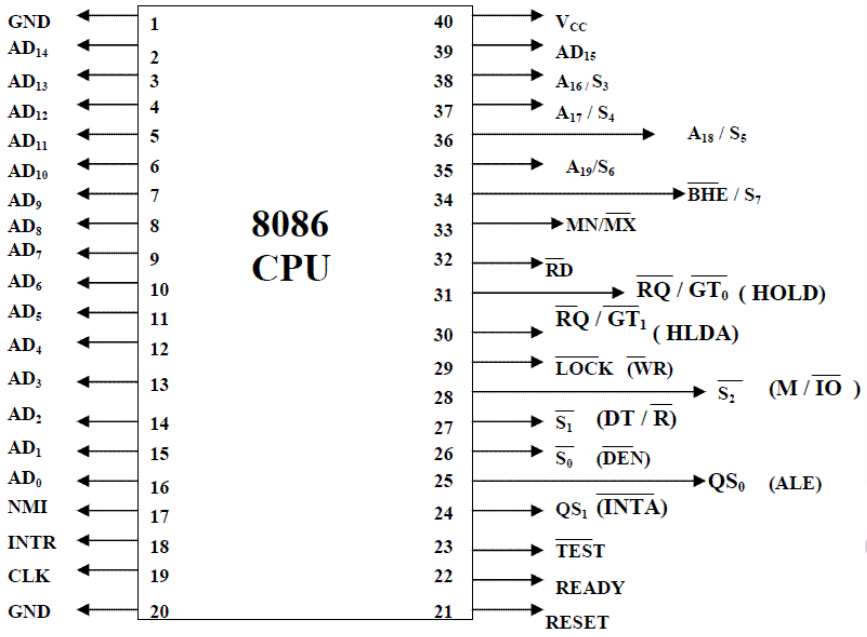
Minimum mode

- 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.

Pin Diagram of 8086 and Pin description of 8086

Figure shows the Pin diagram of 8086. The description follows it.

Pin Diagram 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

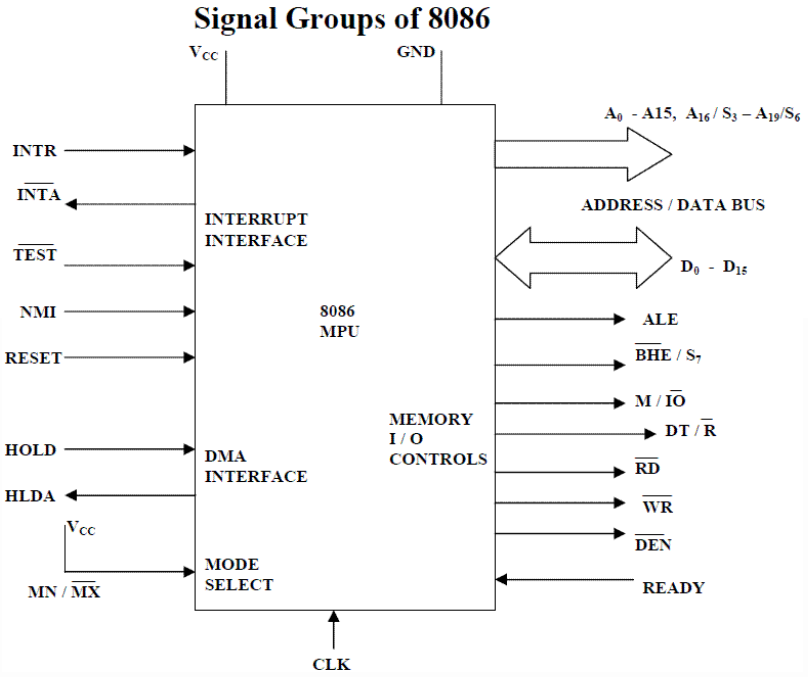
mode) and other function in maximum mode configuration (multiprocessor mode).

- 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
    - The 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, and 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 combine indicate which segment registers 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 is 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
0	0	Whole word
0	1	Upper byte from or to even address
1	0	Lower byte from or to even address

- **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 acknowledge cycle.
- **RD – Read:** This signal on low indicates the peripheral that the processor is performing 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. It's an asymmetric square wave with 33% duty cycle.

Figure shows the Pin functions of 8086.



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 ".
- **INTA – Interrupt 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 transceivers (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 transceivers (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 functions are applicable for maximum mode operation of 8086.

- **S2, S1, and 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.
- **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. By prefetching the instruction, there is a considerable speeding up in instruction execution in 8086. This is known as **instruction pipelining**.

S2	S1	So	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

- 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 and 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. 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

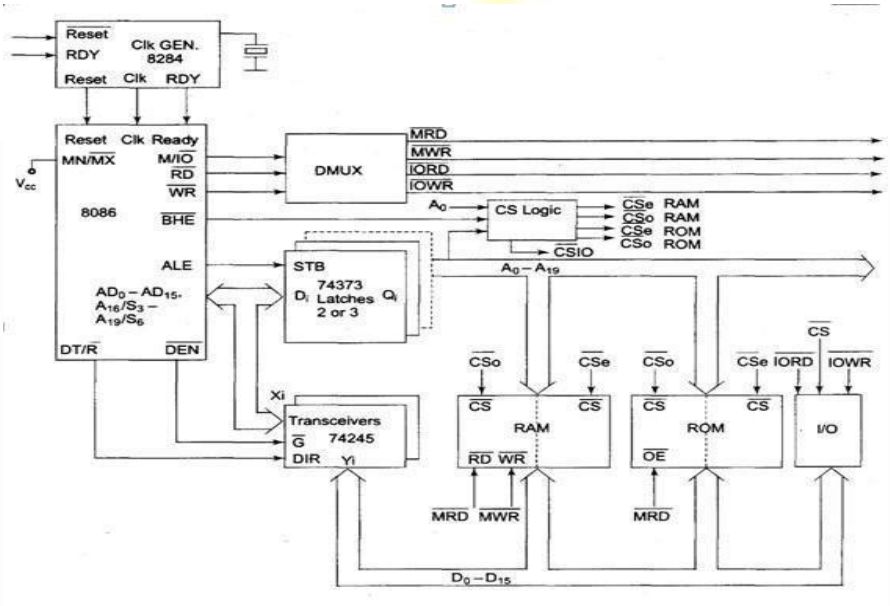
- **RQ/GT0, RQ/GT1 – Request/Grant:** These pins are used by the other local bus master in maximum mode, to force the processor to release the local bus at the end of the processor current bus cycle.

  - 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 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.

Minimum Mode 8086 System

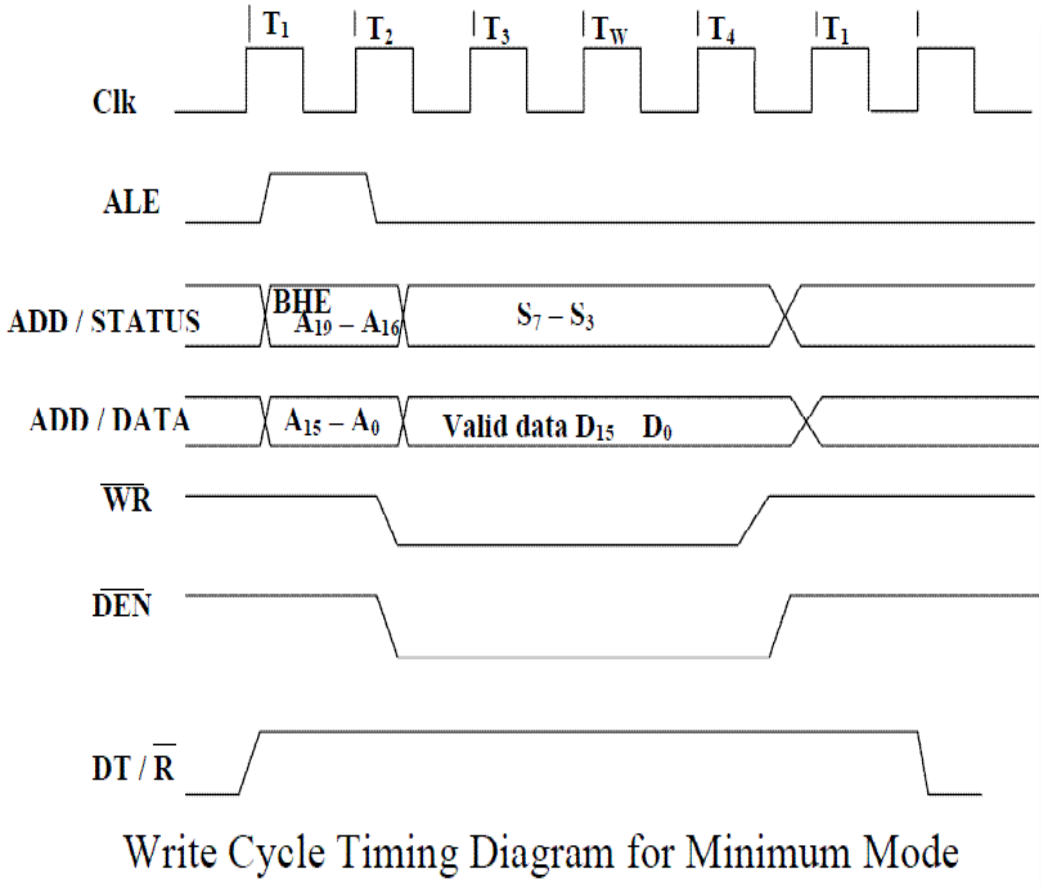


Minimum mode 8086 system

- 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, transceivers, 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.
- Transceivers are the bidirectional buffers and sometimes 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 is used for monitor storage, while RAM for users program storage. A system may contain I/O devices.

Write Cycle Timing Diagram for Minimum Mode

- 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.

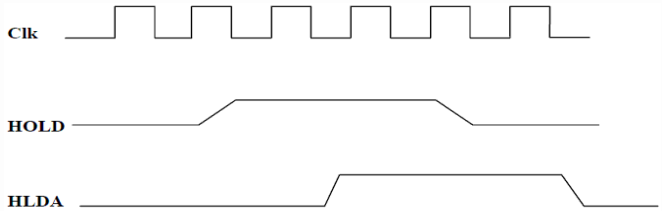


Write Cycle Timing Diagram for Minimum Mode

- 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.

Bus Request and Bus Grant Timings in Minimum Mode System of 8086



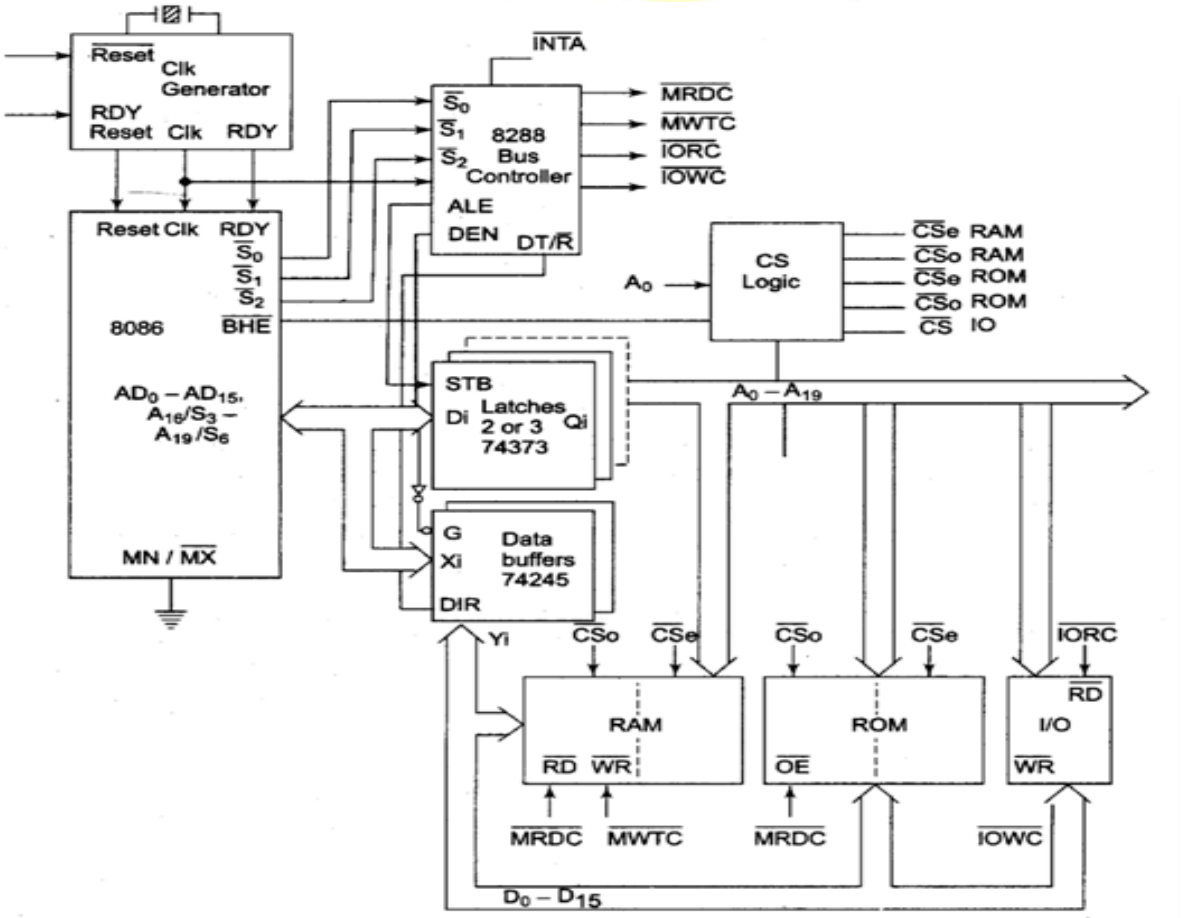
Bus Request and
Bus Grant Timings in Minimum Mode System

- 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.

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.

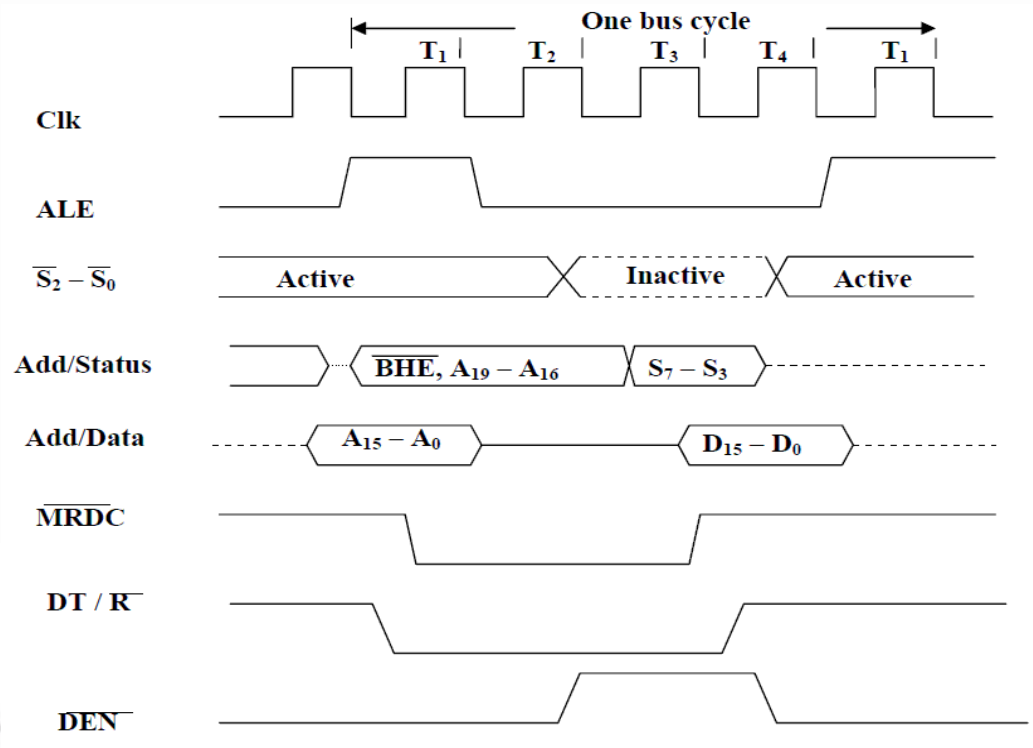
- The basic function of the bus controller chip IC8288 is to derive control signals like RD and WR (for memory and I/O devices), DEN, DT/R, ALE etc. using the information by the processor on the status lines.
- The bus controller chip has input lines S2, S1, S0 and CLK. These inputs to 8288 are driven by CPU.



- It derives the outputs ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC, IOWC and AIOWC. The AEN, IOB and CEN pins are especially 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.

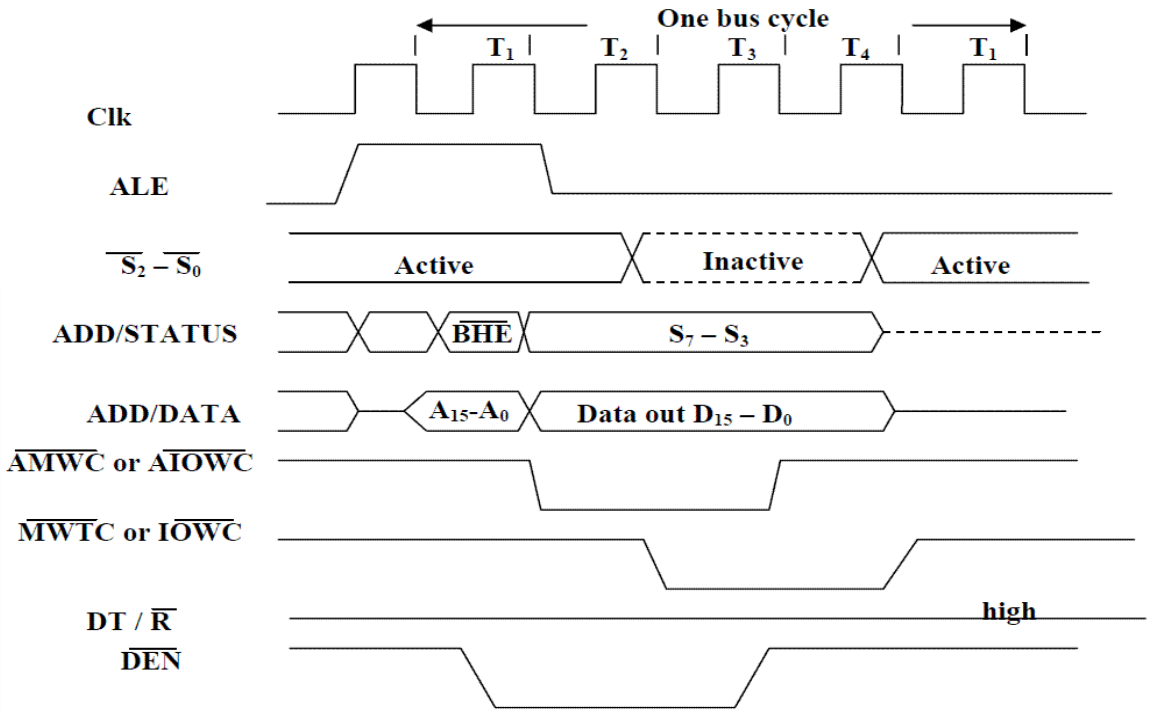
- The MRDC, MWTC are memory read command and memory write command signals respectively and may be used as memory read or write signals.
- All these command signals instructs the memory to accept or send data from or to the bus.
- For both of these write command signals, the advanced signals namely AIOWC 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.
- 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.

Memory Read Timing Diagram in Maximum Mode of 8086



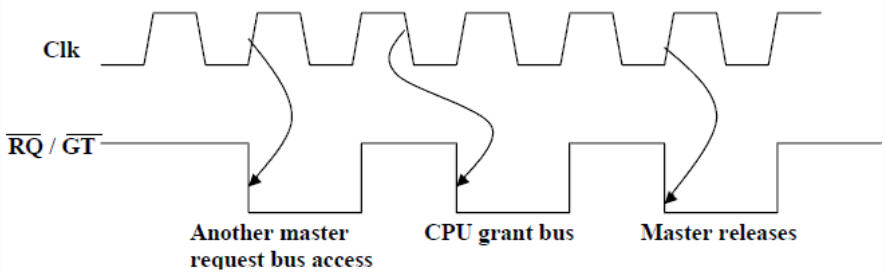
Memory Read Timing in Maximum Mode

Memory Write Timing in Maximum mode of 8086



Memory Write Timing in Maximum mode.

RQ/GT Timings in Maximum Mode



RQ/GT Timings in Maximum Mode

- 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 is 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 :

1. **Address/data bus**
2. **Status**
3. **Control**
4. **Interrupt and**
5. **DMA**.

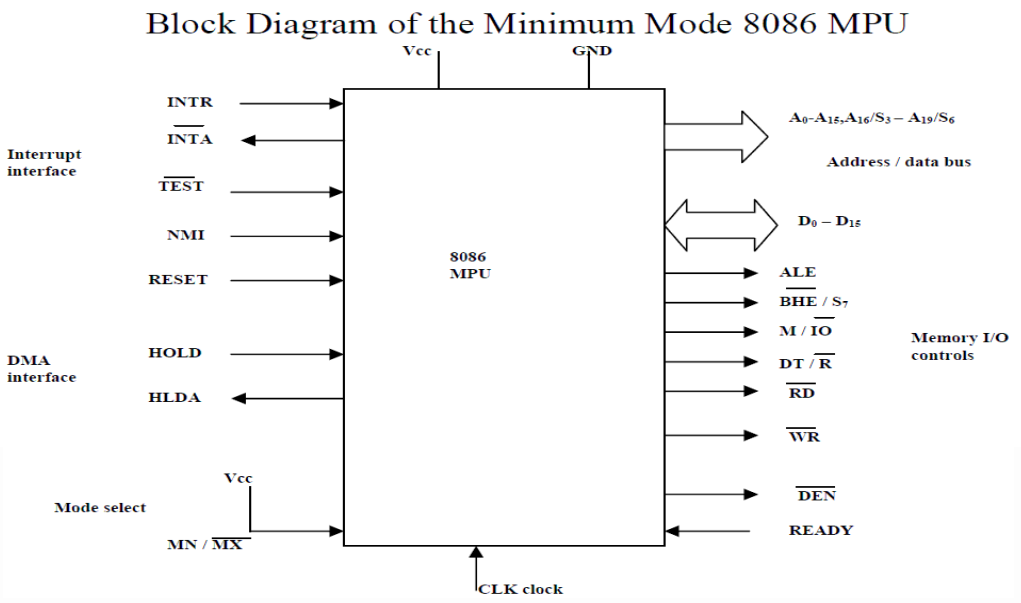
Each and every group is explained clearly.

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. Moreover 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 is 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.

S4	S3	Segment Register
0	0	Extra
0	1	Stack
1	0	Code / none
1	1	Data

Memory segment status codes

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 serve a second function, which is as the S7 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 signals 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 non maskable interrupt NMI and the reset interrupt RESET.
- On the 0-to-1 transition of NMI control is passed to a non maskable 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:

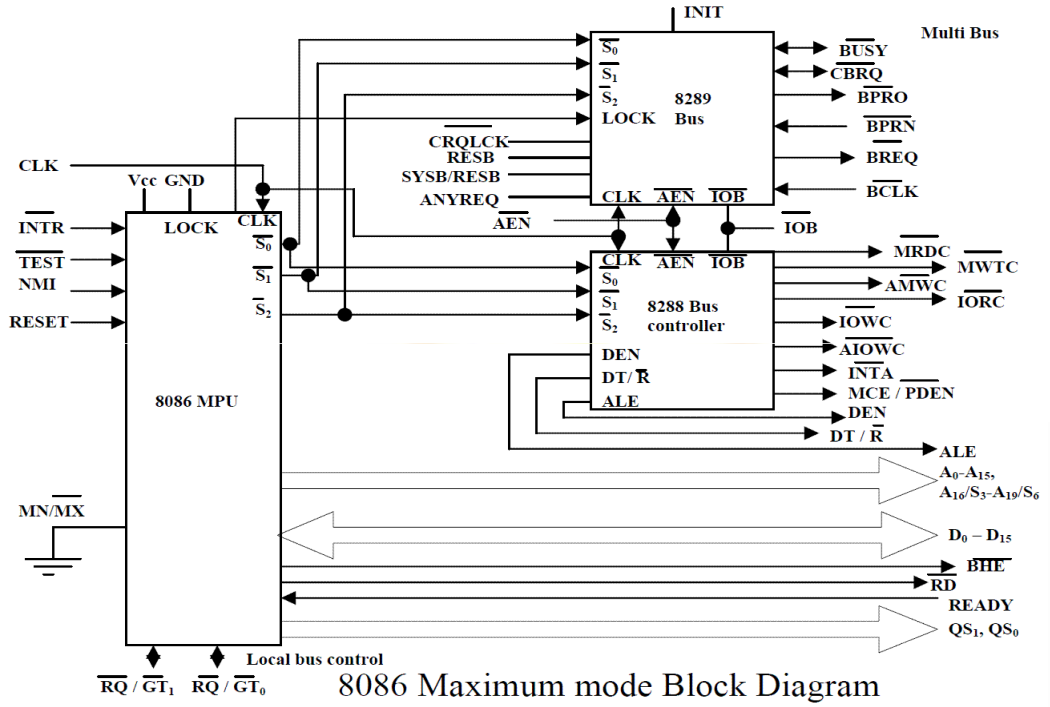
- The direct memory access DMA interface of the 8086 minimum mode consist of the HOLD and HLDA signals.
- When an external device wants to take control of the system bus, it signals to the 8086 by switching HOLD to the logic 1 level. At the completion of the current bus cycle, the 8086 enters the hold 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 these two processors 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.



8086 Maximum mode Block Diagram

- 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.
- 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.

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Indication	8288 Command
0	0	0	Interrupt Acknowledge	$\overline{INTA}$
0	0	1	Read I/O port	$\overline{IORC}$
0	1	0	Write I/O port	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	Halt	None
1	0	0	Instruction Fetch	$\overline{MRDC}$
1	0	1	Read Memory	$\overline{MRDC}$
1	1	0	Write Memory	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	Passive	None

- 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 Multi bus 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 Multi bus 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.
- Local Bus Control Signal – Request / Grant Signals: In a maximum mode configuration, the minimum mode HOLD, HLDA interface is also changed

QS1	QS0	Queue Status
0 (Low)	0	Queue Empty. The queue has been reinitiated as a result of the execution of a transfer instruction.
0	1	First Byte. The byte taken from the queue was the first byte of the instruction.
1	0	Queue Empty. The queue has been reinitiated as a result of the execution of a transfer instruction.
1	1 (High)	Subsequent Byte. The byte taken from the queue was the subsequent byte of the instruction.

- . 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.

Interrupts

Definition: The meaning of 'interrupts' is to break the sequence of operation. While the CPU is executing a program, on 'interrupt' breaks the normal sequence of execution of instructions, diverts its execution to some other program called Interrupt Service Routine (ISR).After executing ISR , the control is transferred back again to the main program. Interrupt processing is an alternative to polling.

Need for Interrupt: Interrupts are particularly useful when interfacing I/O devices that provide or require data at relatively low data transfer rate.

Types of Interrupts: There are two types of Interrupts in 8086. They are: (i)Hardware **interrupts** (External Interrupts). The Intel microprocessors support hardware interrupts through:

- Two pins that allow interrupt requests, INTR and NMI
- One pin that acknowledges, INTA, the interrupt requested on

INTR. INTR and NMI

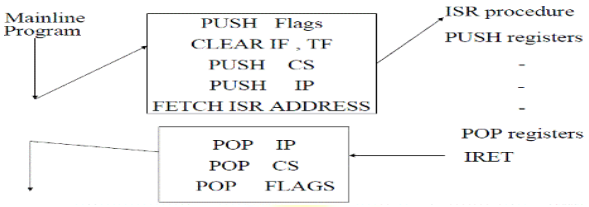
- INTR is a maskable hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POPF instruction.
- When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 * <interrupt type>. Interrupt processing routine should return with the IRET instruction.
- NMI is a non-maskable interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority than the maskable interrupt.
- – Ex: NMI, INTR.

(ii) **Software Interrupts** (Internal Interrupts and Instructions) .Software interrupts can be caused by:

- INT instruction - breakpoint interrupt. This is a type 3 interrupt.
- INT <interrupt number> instruction - any one interrupt from available 256 interrupts.
- INTO instruction - interrupt on overflow
- Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When the CPU processes this interrupt it clears TF flag before calling the interrupt processing routine.
- Processor exceptions: Divide Error (Type 0), Unused Opcode (type 6) and Escape opcode (type 7).
- Software interrupt processing is the same as for the hardware interrupts.

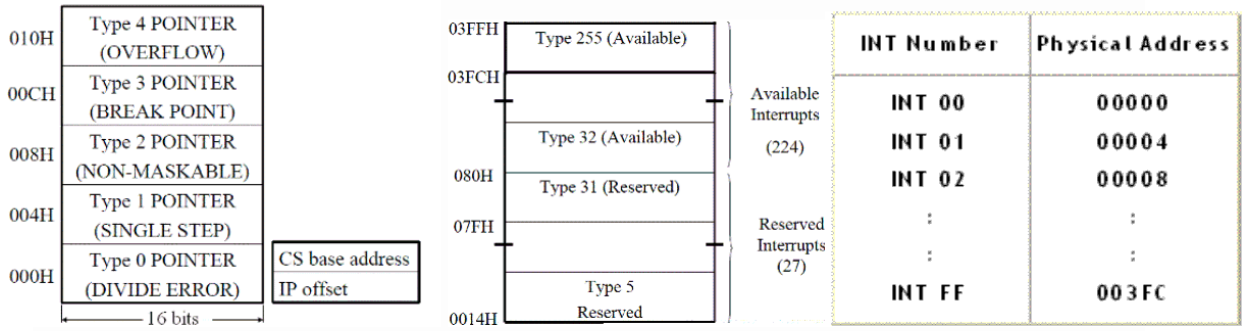
- - Ex: INT n (Software Instructions)
- Control is provided through:
    - IF and TF flag bits
    - IRET and IRETD

Performance of Software Interrupts



1. It decrements SP by 2 and pushes the flag register on the stack.
2. Disables INTR by clearing the IF.
3. It resets the TF in the flag Register.
5. It decrements SP by 2 and pushes CS on the stack.
6. It decrements SP by 2 and pushes IP on the stack.
6. Fetch the ISR address from the interrupt vector table.

Interrupt Vector Table



Functions associated with INT00 to INT04

INT 00 (divide error)

- INT00 is invoked by the microprocessor whenever there is an attempt to divide a number by zero.
- ISR is responsible for displaying the message “Divide Error” on the screen

INT 01

- For single stepping the trap flag must be 1
- After execution of each instruction, 8086 automatically jumps to 00004H to fetch 4 bytes for CS: IP of the ISR.
- The job of ISR is to dump the registers on to the screen

INT 02 (Non maskable Interrupt)

- When ever NMI pin of the 8086 is activated by a high signal (5v), the CPU Jumps to physical memory location 00008 to fetch CS:IP of the ISR associated with NMI.

INT 03 (break point)

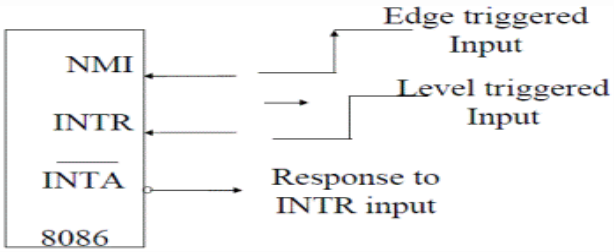
- A break point is used to examine the CPU and memory after the execution of a group of Instructions.
- It is one byte instruction whereas other instructions of the form "INT nn" are 2 byte instructions.

INT 04 (Signed number overflow)

- There is an instruction associated with this INT 0 (interrupt on overflow).
- If INT 0 is placed after a signed number arithmetic as IMUL or ADD the CPU will activate INT 04 if 0F = 1.
- In case where 0F = 0, the INT 0 is not executed but is bypassed and acts as a NOP.

Performance of Hardware Interrupts

- NMI : Non maskable interrupts - TYPE 2 Interrupt
- INTR : Interrupt request - Between 20H and FFH



Interrupt Priority Structure

Interrupt	Priority
Divide Error, INT(n),INTO	Highest
NMI	↓
INTR	↓
Single Step	Lowest

ASSEMBLY LANGUAGE PROGRAMMING

2.1. Addressing modes of 8086

Various addressing modes of 8086/8088

1) Register Addressing mode
2) Immediate Addressing mode
3) Register indirect addressing mode
4) Direct Addressing mode
5) Indexed Addressing mode
6) Base Relative Addressing mode
7) Base Indexed Addressing mode

Register Addressing Mode

Data transfer using registers is called register addressing mode. Here operand value is present in register. For example

MOV AL,BL;

MOV AX,BX;

Immediate Addressing mode

When data is stored in code segment instead of data segment immediate addressing mode is used. Here operand value is present in the instruction. For example

MOV AX, 12345;

Direct Addressing mode

When direct memory address is supplied as part of the instruction is called direct addressing mode. Operand offset value with respect to data segment is given in instruction. For example

MOV AX, [1234];

ADD AX, [1234];

Register indirect addressing mode: Here operand offset is given in a cpu register. Register used are BX, SI(source index), DI(destination index), or BP(base pointer). BP holds offset w.r.t Stack segment, but SI,DI and BX refer to data segment. For example

MOV [BX],AX;

ADD AX, [SI];

Indexed Addressing mode

Here operand offset is given by a sum of a value held in either SI, or DI register and a constant displacement specified as an operand. For example

Lets take arrays as an example. This is very efficient way of accessing arrays.

My_array DB '1', '2', '3','4,'5';

MOV SI, 3;

MOV AL, My_array[3];

So AL holds value 4.

Base Relative addressing mode

Operand offset given by a sum of a value held either in BP, or BX and a constant offset specified as an operand. For example

MOV AX,[BP+1];

JMP [BX+1];

Base Indexed

Here operand offset is given by sum of either BX or BP with either SI or DI. For example

MOV AX, [BX+SI]

JMP [BP+DI]

2.2. 8086 Assembler directives

ASSUME Directive - The ASSUME directive isused to tell the assembler that the name of the logical segmentshould be used for a specified segment. The 8086 works directly with only 4 physical segments: a Code segment, a data segment, a stack segment, and an extra segment.

Example:

ASUME CS: CODE; This tells the assembler that the logical segment named CODE contains the ; instruction statements for the program and should be treated as a code segment.

ASUME DS: DATA; This tells the assembler that for any instruction which refers to a data in the ;data segment, data will found in the logical segment DATA.

DB - DB directive is used to declare a bytetype variable or to store a byte in memory location.

Example:

1. PRICE DB 49h, 98h, 29h ;Declare an array of 3 bytes, named as PRICE and initialize.

2. NAME DB 'ABCDEF' ;Declare an array of 6 bytes and initialize with ASCII code for letters

3. TEMP DB 100 DUP(?) ;Set 100 bytes of storage in memory and give it the name as TEMP, ;but leave the 100 bytes uninitialized. Program instructions will load values into these locations

DW - The DW directive is used to define a variable of type word or to reserve storage location of type word in memory.

Example:

MULTIPLIER DW437Ah ; this declares a variable of type word and named it as MULTIPLIER. ;This variable is initialized with the value 437Ah when it is loaded into memory to run.

EXP1 DW 1234h, 3456h, 5678h ; this declares an array of 3 words and initialized with specified ;values.

STOR1 DW 100 DUP(0); Reserve an array of 100 words of memory and initialize all words with

;0000.Array is named as STOR1

END - END directive is placed after the last statement of a program to tell the assembler that this is the end of the program module. The assembler will ignore any statement after an END directive. Carriage return is required after the END directive.

ENDP - ENDP directive is used along with the name of the procedure to indicate the end of a procedure to the assembler

Example:

SQUARE_NUM PROCE ; It start the procedure Some steps to find the square root of a number

ENDS - This ENDS directive is used with name of the segment to indicate the end of that logic segment.

Example:

CODE SEGMENT ;Hear it Start the logic segment containing code

; Some instructions statements to perform the logical

;operation

CODE ENDS ;End of segment named as CODE

EQU - This EQU directive is used to give a name to some value or to a symbol. Each time the assembler finds the name in the program, it will replace the name with the value or symbol you given to that name.

Example:

FACTOR EQU 03H ; you has to write this statement at the starting of your program and later in ;the program you can use this as follows

ADD AL, FACTOR; When it codes this instruction the assembler will code it as ADDAL, 03H ;The advantage of using EQU in this manner is, if FACTOR is used many no of times in a ;program and you want to change the value, all you had to do is change the EQU statement at ;beginning, it will changes the rest of all.

EVEN - This EVEN directive instructs the assembler to increment the location of the counter to the next even address if it is not already in the even address. If the word is at even address 8086 can read a memory in 1 bus cycle. If the word starts at an odd address, the 8086 will take 2 bus cycles to get the data. A series of words can be read much more quickly if they are at even address. When EVEN is used the location counter will simply incremented to next address and NOP instruction is inserted in that incremented location

GROUP - The GROUP directive is used to group the logical segments named after the directive into one logical group segment.

INCLUDE - This INCLUDE directive is used to insert a block of source code from the named file into the current source module.

PROC - The PROC directive is used to identify the start of a procedure. The term near or far is used to specify the type of the procedure.

Example:

SMART PROC FAR; This identifies that the start of a procedure named as SMART and ;instructs the assembler that the procedure is far .

SMART ENDP;This PROC is used with ENDP to indicate the break of the procedure.

PTR - This PTR operator is used to assign a specific type of a variable or to a label.

Example:

INC [BX] ; This instruction will not know whether to increment the byte pointed to by BX or a ;word pointed to by BX.

INC BYTE PTR [BX] ;increment the byte pointed to by BX

This PTR operator can also be used to override the declared type of variable. If we want to access the a byte in an

Array WORDS DW 437Ah, 0B97h,

MOV AL, BYTE PTR WORDS

PUBLIC - The PUBLIC directive is used to instruct the assembler that a specified name or label will be accessed from other modules.

Example:

PUBLIC DIVISOR, DIVIDEND; these two variables are public so these are available to all ;modules. If an instruction in a module refers to a variable in another assembly module, we can ;access that module by declaring as EXTRN directive.

TYPE - TYPE operator instructs the assembler to determine the type of a variable and determines the number of bytes specified to that variable.

Example:

Byte type variable – assembler will give a value 1

Word type variable – assembler will give a value 2

Double word type variable – assembler will give a value 4

ADD BX, TYPE WORD_ ARRAY; hear we want to increment BX to point to next word in an array of words.

2.3. Instruction set of 8086

DATA TRANSFER INSTRUCTIONS

GENERAL – PURPOSE BYTE OR WORD TRANSFER INSTRUCTIONS:

MOV

PUSH

POP

XCHG

XLAT

SIMPLE INPUT AND OUTPUT PORT TRANSFER INSTRUCTIONS

IN

OUT

SPECIAL ADDRESS TRANSFER INSTRUCTIONS

LEA

LDS

LES

FLAG TRANSFER INSTRUCTIONS:

LAHF

SAHF

PUSHF

POPF

ADITION INSTRUCTIONS:

ADD

ADC

INC

AAA

DAA

SUBTSUBTRACTION INSTRUCTIONS:

SUB

SBB

DEC

NEG

CMP

AAS

DAS

MULTIPLICATION INSTRUCTIONS:

MUL

IMUL

AAM

DIVISION INSTRUCTIONS:

DIV

IDIV

AAD

CBW

CWD

LOGICAL INSTRUCTIONS:

NOT

AND

OR

XOR

TEST

SHIFT INSTRUCTIONS:

SHL / SAL

SHR

SAR

PROGRAM EXECUTION TRANSFER INSTRUCTIONS

UNCONDITIONAL TRANSFER INSTRUCTIONS:

CALL

RET

JMP

CONDITIONAL TRANSFER INSTRUCTIONS:

JA / JNBE

JAE / JNB

JB / JNAE

JBE / JNA

JC/ JNC

JE / JZ

JG / JNLE

JGE / JNL

JL / JNGE

JLE / JNG

JNE / JNZ

JNO

JNP / JPO

JNS

JO

ITERATION CONTROL INSTRUCTIONS:

LOOP

LOOPE / LOOPZ

LOOPNE / LOOPNZ

JCXZ

INTERRUPT INSTRUCTIONS:

INT

INTO

IRET

PROCESS CONTROL INSTRUCTIONS

FLAG SET / CLEAR INSTRUCTIONS:

STC

CLC

CMC

STD

CLD

STI

CLI

EXTERNAL HARDWARE SYNCHRONIZATION INSTRUCTIONS:

HLT

WAIT

ESC

LOCK

NOP

AAA Instruction - ASCII Adjust after Addition

AAD Instruction - ASCII adjust before Division

AAM Instruction - ASCII adjust after Multiplication

AAS Instruction - ASCII Adjust for Subtraction

AAA Instruction -AAA converts the result of the addition of two valid unpacked BCD digits to a

valid 2-digit BCD number and takes the AL register as its implicit operand. Two operands of the addition must have its lower 4 bits contain a number in the range from 0-9.The AAA instruction then adjust AL so that it contains a correct BCD digit. If the addition produce carry (AF=1), the AH register is incremented and the carry CF and auxiliary carry AF flags are set to 1. If the addition did not produce a decimal carry, CF and AF are cleared to 0 and AH are not altered. In both cases the higher 4 bits of AL are cleared to 0. AAA will adjust the result of the two ASCII characters that were in the range from 30h ("0") to 39h("9").This is because the lower 4 bits of those character fall in the range of 0-9.The result of addition is not a ASCII character but it is a BCD digit.

AAM Instruction - AAM converts the result of the multiplication of two valid unpacked BCD digits into a valid 2-digit unpacked BCD number and takes AX as an implicit operand. To give a valid result the digits that have been multiplied must be in the range of 0 – 9 and the result should have been placed in the AX register. Because both operands of multiply are required to be 9 or less, the result must be less than 81 and thus is completely contained in AL. AAM unpacks the result by dividing AX by 10, placing the quotient (MSD) in AH and the remainder (LSD) in AL.

AAS Instruction - AAS converts the result of the subtraction of two valid unpacked BCD digits to a single valid BCD number and takes the AL register as an implicit operand. The two operands of the subtraction must have its lower 4 bit contain number in the range from 0 to 9 .The AAS instruction then adjust AL so that it contain a correct BCD digit.

2.4. Assembly language programs

1. Addition of two 16-bit numbers-signed & unsigned

Registers used: AX,DS

Flags affected: AF,CF,OF,PF,SF,ZF

Program:

```
ASSUME CS:CODE,DS:DATA

DATA SEGMENT

OPR1 DW 4269H

OPR2 DW 1000H

RES DW ?

DATA ENDS
```

```
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPR1
ADD AX,OPR2
MOV RES,AX
MOV AH,4CH
INT 21H
CODE ENDS
END  START
END
```

Result:

INPUT: OPR1 = 4269H

OPR2 = 1000H

OUTPUT: RES = 5269H

2. Multiplication of two 16-bit unsigned numbers

Registers used: AX,DS

Flags affected: OF,CF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
OPR1 DW 2000H
OPR2 DW 4000H
```

```
RESLW DW ?
RESHW DW ?
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPR1
MUL OPR2
MOV RESLW,AX
MOV RESHW,DX
MOV AH,4CH
INT 21H
CODE ENDS
END  START
END
```

Result:

INPUT: OPR1 = 2000H

OPR2 = 4000H

OUTPUT: RESLW = 0000H (AX)

RESHW = 0800H (DX)

3. division of unsigned numbers

Registers used: AX,DS

Flags affected: IF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
OPR1 DW 2C58H
OPR2 DB 56H
RESQ DB ?
RESR DB ?
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPR1
DIV OPR2
MOV RESQ,AL
MOV RESR,AH
MOV AH,4CH
INT 21H
CODE ENDS
END  START
END
```

Result:

INPUT: OPR1 = 2C58H (DIVIDEND)

OPR2 = 56H (DIVISOR)

OUTPUT: RESQ = 84H (AL)

RESR = 00H (AH)

4. Logical and operation

Registers used: AX,DS

Flags affected: PF,SF,ZF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
OPR1 DW 6493H
OPR2 DW 1936H
RES DW ?
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPR1
AND AX,OPR2
MOV RES,AX
MOV AH,4CH
INT 21H
CODE ENDS
END  START
END
```

Result:

INPUT: OPR1 = 6493H

OPR2 = 1936H

OUTPUT: RES = 0012H

5. Shift arithmetic / logical left operation

Registers used: AX,DS

Flags affected: SF,ZF,PF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
OPR1 DW 1639H
RES DW ?
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPR1
SAL AX,01H ←(or)→ SHL AX,01H
MOV RES,AX
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Result:

INPUT: OPR1 = 1639H

OUTPUT: RES = 2C72H

6. Rotate right with carry

Registers used: AX,DS

Flags affected: CF,OF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
OPR1 DW 1639H
RES DW ?
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPR1
RCR AX,01H
MOV RES,AX
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

END

Result:

INPUT: OPR1 = 1639H

OUTPUT: RES = 0B1CH

7. Ascending order

Registers used: AX,DS,ES,SI,DI

Flags affected: AX,DS,SI,CX,DX

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
LIST DW 05H,04H,01H,03H,02H
COUNT EQU 05H
DATA ENDS
CODE SEGMENT
START:MOV AX,DATA
MOV DS,AX
MOV DX,COUNT-1
BACK:MOV CX,DX
MOV SI,OFFSET LIST
AGAIN:MOV AX,[SI]
CMP AX,[SI+2]
JC GO
XCHG AX,[SI+2]
XCHG AX,[SI]
```

```
GO:INC SI
INC SI
LOOP AGAIN
DEC DX
JNZ BACK
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Result:

INPUT: LIST (DS: 0000H) = 05H,04H,01H,03H,02H

OUTPUT: LIST (DS: 0000H) = 01H,02H,03H,04H,05H

8. Packed bcd to unpacked bcd Registers used:

AX,DS,BL,CL **Flags affected:**

PF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
BCD DB 49H
COUNT DB 04H
UBCD1 DB ?
UBCD2 DB ?
DATA ENDS
```

```
CODE SEGMENT
START:MOV AX,DATA
MOV DS,AX
MOV AL,BCD
MOV BL,AL
AND AL,0FH
MOV UBCD1,AL
MOV AL,BL
AND AL,0F0H
MOV CL,COUNT
ROR AL,CL
MOV UBCD2,AL
MOV AH,4CH
INT 21H
CODE ENDS
END  START
END
```

Result:

INPUT: BCD = 49

OUTPUT: UBCD1 = 09

UBCD2 = 04

9. Reverse string

Registers used: AX,DS,SI,DI,CL

Flags affected: ZF,PF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
STR DB 01H,02H,03H,04H
COUNT EQU 02H
DATA ENDS
CODE SEGMENT
START:MOV AX,DATA
MOV DS,AX
MOV CL,COUNT
MOV SI,OFFSET STR
MOV DI,0003H
BACK:MOV AL,[SI]
XCHG [DI],AL
MOV [SI],AL
INC SI
DEC DI
DEC CL
JNZ BACK
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Result:

INPUT: STR (DS:0000H) = 01H,02H,03H,04H

OUTPUT: STR (DS:0000H) = 04H,03H,02H,01H

10. Length of the string

Registers used: AX,DS,SI,CL

Flags affected: ZF,PF,SF,AF,CF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
STR DB 01H,03H,08H,09H,05H,07H,02H
LENGTH DB ?
DATA ENDS
CODE SEGMENT
START:MOV AX,DATA
MOV DS,AX
MOV AL,00H
MOV CL,00H
MOV SI,OFFSET STR
BACK:CMP AL,[SI]
JNC GO
INC CL
INC  SI
JNZ BACK
GO:MOV LENGTH,CL
MOV AH,4CH
```

```
INT 21H
CODE ENDS
END START
END
```

Result:

INPUT: STR (DS:0000H) = 01H, 03H,08H,09H,05H,07H,02H

OUTPUT: LENGTH = 07H (CL)

11. String comparison

Registers used: AX,DS,SI,DI,CL

Flags affected: ZF,CF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
STR DB 04H,05H,07H,08H
COUNT EQU 04H
ORG 0010H
STR1 DB 04H,06H,07H,09H
DATA ENDS
CODE SEGMENT
START:MOV AX,DATA
MOV DS,AX
MOV SI,OFFSET STR
MOV DI,OFFSET STR1
```

```
MOV CL,COUNT
CLD
REP CMPSB
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Result:

INPUT: STR (DS:0000H) = 04H,05H,07H,08H

STR1 (DS:0010H) = 04H,06H,07H,09H

OUTPUT: **I):** IF STR = STR1 THEN ZF = 1 &

II): IF STR ≠ STR1 THEN ZF = 0

12. Display the string

Registers used: AX,DS,DX

Flags affected: No flags are affected

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
MSG DB 0DH,0AH,"MICROPROCESSORS ",0DH,0AH,"$"
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
```

```
MOV DS,AX
MOV AH,09H
MOV DX,OFFSET MSG
INT 21H
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Result:

MICROPROCESSOR