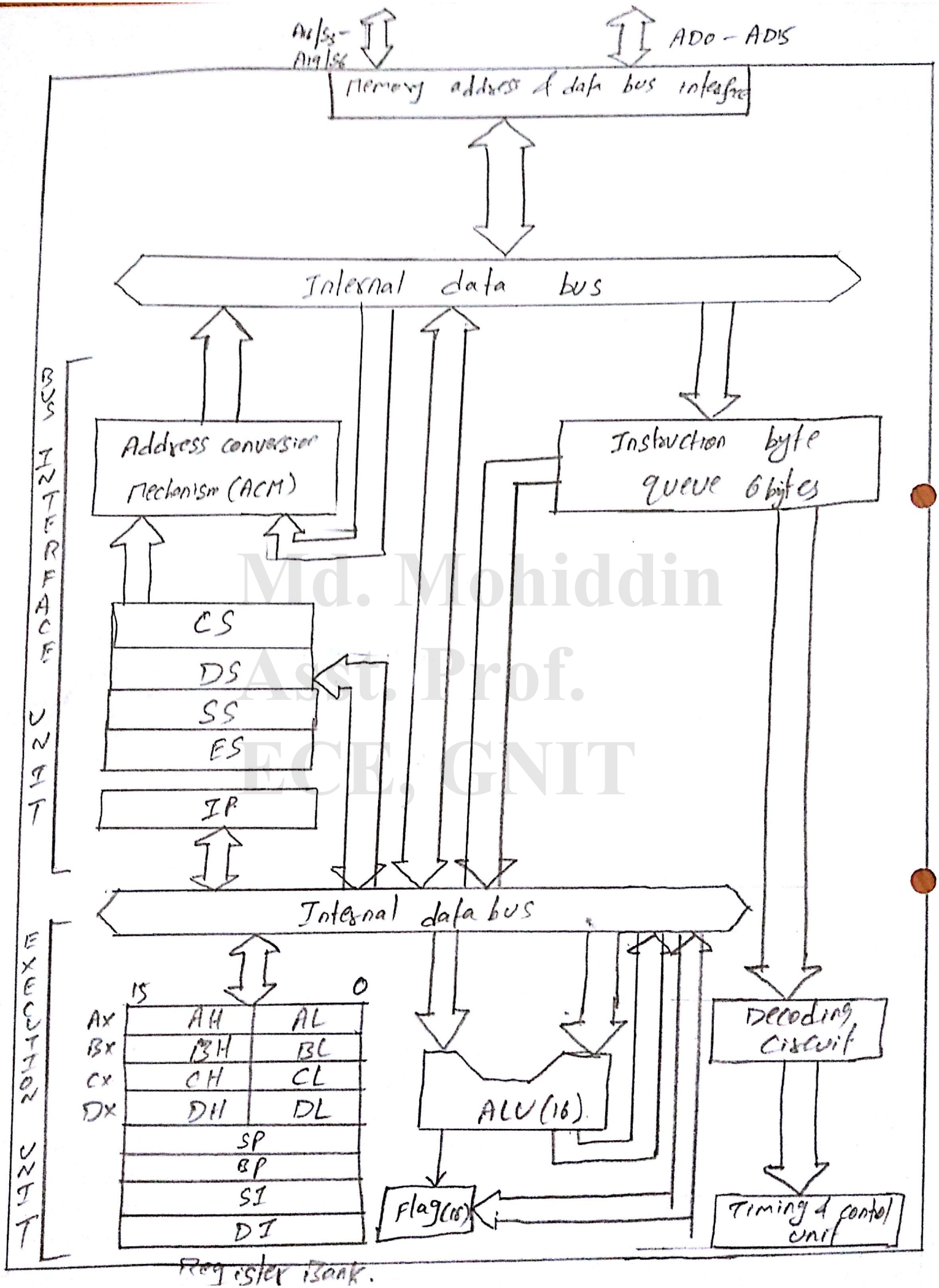


Features of 8086 microprocessor:

1. Number of data lines are 16

   data Range : 0000h to FFFFh.

2. ALU size is 16 bit, so called 16 bit micro processor for general purpose.

3. Number of Address lines are 20.

   so processor can handle 2^{20} = 1Mb memory.

   Addres Range : 00000h to FFFFFh.

4. To identify memory 20 bit address is require.

5. To identify IO device 16 bit address is require.

6. No. of Instructions are 246.

7. It has 2 interrupts like INTR & NMI.

8. This processor is not an accumulator based.

9. clock frequency is 5MHz.

10. It has 40 pin IC Package.

11. It operates with +5V dc supply.

12. Pipelining is introduced in 8086 µP.

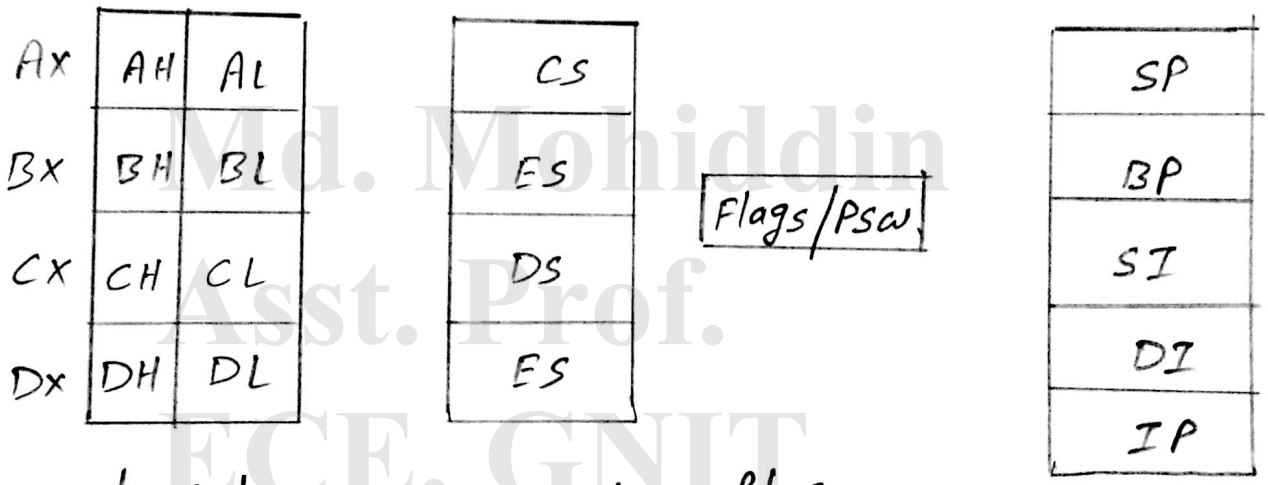
The complete architecture of 8086 can be divided into two parts (a) Bus Interface unit (BIU) and (b) Execution unit (EU).

The BIU contains the circuit for physical address calculations and a pre decoding instruction byte queue. This unit is responsible for establishing communications with external devices and peripherals including memory via the bus. This unit is also responsible for generating the complete physical address which is 20 bits long is generated using segment and offset registers, each 16-bits long.

The execution unit contains the register set of 8086 except segment registers and IP. It has a 16-bit ALU, able to perform arithmetic and logic operations. The 16-bit flag register reflects the results of execution by the ALU. The decoding unit decodes the opcodes bytes issued from the instruction byte queue. The timing and control unit derives the necessary control signals to execute the instruction opcode received from the queue.

Register Organisation of 8086

8086 has a powerful set of registers known as general purpose and special purpose registers. All of them are 16-bit registers. The general purpose registers, can be used as either 8-bit registers or 16-bit registers.



General data Registers | Segment Registers | flag Registers | Pointer and index Regis

General Data Registers :

Usually the letters L and H specify th lower and higher bytes of a particular register.

The Registers AX, BX, CX and DX are the general purpose 16-bit registers.

Ax is used as 16-bit accumulator, with the lower 8-bits of Ax designated AL and higher 8-bits as AH. AL can be used as an 8-bit accumulator for 8-bit operations.

The register BX is used as an offset storage for forming Physical address in case of certain addressing modes.

The register CX is also used as a default counter in case of string and loop instructions.

Dx register is a general purpose register which may be used as an implicit operand or destination in case of a few instructions.

Segment Registers:

The code segment register is used for addressing a memory location in the code segment, where executable Prog. stored.

The Data segment register points to the data segment of Memory, where the data is resided.

The extra segment register points to the extra segment of Memory, where the data is resided.

The stack segment register is used for addressing stack segment of Memory. i.e which is used to store stack data.

Pointers and Index Registers

The pointers contain offset within the particular segments. the pointers IP, BP and SP usually contain offsets with in the code (IP), and stack (BP & SP) segments. the index registers are used as general purpose registers as well as for offset storage in case of indexed, based indexed and relative based indexed addressing modes. the register SI is generally used to store the offset of source data in data segment while the register DI is used to store the offset of destination in data or extra segment. the index registers are particularly useful for string manipulations.

8086 flag Register & fuctions of 8086 flags.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	O	D	I	T	S	Z	X	A	X	P	X	C

8086 has a 16-bit flag registes which is divided into two parts, eg: a, conditional code (or) status flag and b, Machine contsol flags.

Conditional code flag Registers :

The conditional code flag Registes is the lowes byte of the 16-bit flag registes along with the owerflow flag. These flags reflects the result of the operations perfosmed by ALU. The functions of these flags are:

C - cassy flag : This flag is set when there is a cassy out of MSB in case of addition or a bossow in case of subtraction

P - Pasity flag : this flag is set to 1 if the lowes byte of the result contains even numbes of 1s.

A - Auxiliasy cassy Flag : this is set if there is a cassy from lowes nibble to uppes nibble.

Z - zeso flag : This flag is set to 1 if the result of the computation (or) compasision perfosmed by the previous instruction is zeso.

S - sign Flag : This flag is set when the result of any computation is negative. For signed computations, the sign flag equals the MSB of the result.

O - overflow Flag : This flag is set if an overflow occurs that is if the result of a signed operation is large enough to be accomodated in a destination register.

control flag Register.

* the control flag Register is the higher byte of flag register
* It consists of direction flag, Interrupt flag, & Trap flag.

Directional flag - D : this is used by string manipulation instruction.

* If the flag bit is '0', the string is processed beginning from the lowest address to the highest address.

Interrupt flag - I : If the flag is set the Maskable interrupts are recognised by the CPUs, otherwise they are ignored.

Trap flag - T : If the flag is set, the processor enters the single step execution mode. the processor executed the current instruction and the control is transferred to the trap interrupt service routine.

Memory Addresses (or)

Generation of Physical Address (or) 20-bit address from segment and offset addresses (two - 16-bit address)

For generating a physical address from contents of segment and offset registers: the content of a segment register also called as segment address is shifted left bit-wise four times and to this result, content of an offset register also called as offset address is added, to produce a 20-bit physical address.

For example, if the segment address is 1000H and the offset is 5555H, then the physical address is:

segment address : 1000H : 0001 0000 0000 0000b

seg. add. shifted by 4-bit wise : 10000H : 0001 0000 0000 0000 0000b

offset address : 5555H : 0101 0101 0101 0101b

0001 0101 0101 0101 0101b

Physical address : 15555H.

(or)

effective address

(or)

20-bit address.

---

formula : Effective address : 10H * segment address + offset address

---

Memory segmentation :

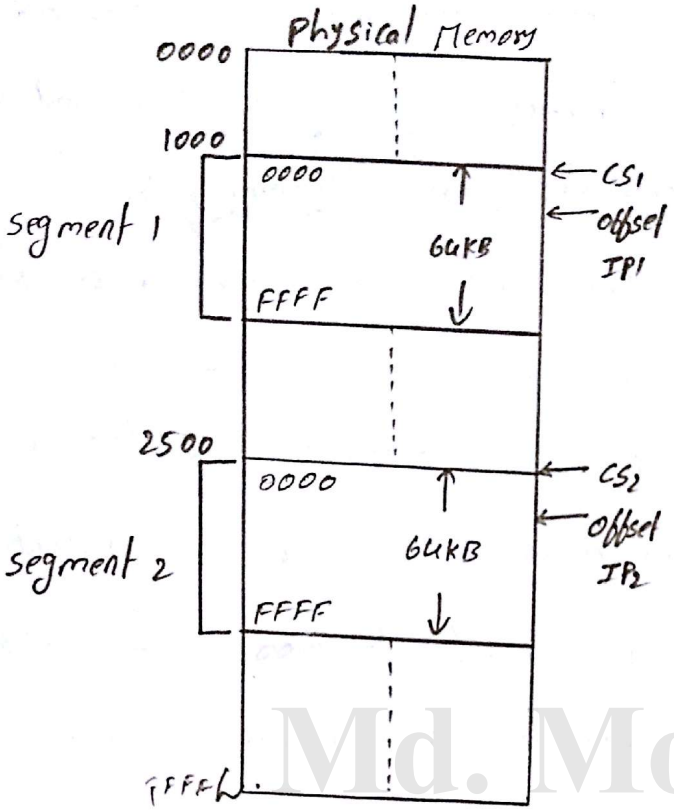
unlike 8085, the 8086 address a segmented memory, the complete 1Mbyte memory, which the 8086 addresses, is divided in to 16 logical segments. Each segment thus contain 64kbytes of memory. there are four segments ~~registers~~, viz. Code segment ~~Register~~ (CS), Data segment ~~Register~~ (DS), Extra segment ~~Register~~ (ES) and stack segment ~~Register~~ (SS). each of ~~four~~ 64kbytes of memory. the 16-bit contents of the segment register actually point to the starting location of a particular segment. To address a specific memory location within a segment, we need an offset address. the offset address is also 16-bit long so that the maximum offset value can be FFFFH, and the maximum size of any segment is thus 64 kbytes locations.

the addresses of the segments may be assigned as 0000H to FFFFH ~~resp~~ called logical address (or) segment address. the offset address values are from 0000H to FFFFH so that the Physical address range from 00000H to FFFFFH. In the above said case, the segments are called non-overlapping segments. In some cases, however, the segments may be overlapping. Suppose a segment starts at a particular address and its max. size up to 64kbytes. But, if another segment starts before this 64kbytes of first segment, then the two segment are overlapped.

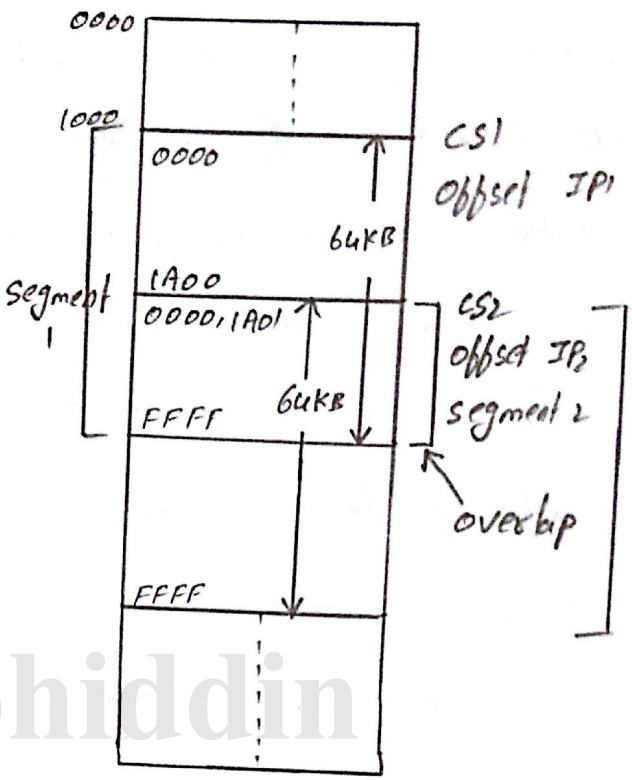
the area of memory called overlapped segment area.

⊛ the overlapped ~~segm~~ Area location Physical address = $CS_1 + IP_1 = CS_2 + IP_2$

+ : ACM



Non overlapping segments



overlapping segments

Advantages

* Allows the memory capacity to be 1Mbytes although the actual addresses to be handled are of 16-bit size.
* Allow the placing of code, data and stack portions of the same program in different parts of memory, for data and code protection.
* Permits a program and/or its data to be put in to different areas of memory each time the program is executed, i.e provision for relocation is done.

Physical Memory Organisation :-

In an 8086 based system, the 1M bytes memory is physically organised as an odd bank and an even bank, each of 512 kbytes, addressed in parallel by the Processor. Byte data with even address is transferred on $D_7 - D_0$, while the byte data with an odd address is transferred on $D_{15} - D_8$ bus lines. The Processor Provides two enable signals, $\overline{BHE}$ and A_0 for selection of either even or both the banks. If the Processor fetches a word (consecutive two bytes) from memory. there are different possibilities, like:

1. Both the bytes may be data operands.
2. Both the bytes may contain opcode bits.
3. One of the bytes may be opcode while the other may be data.

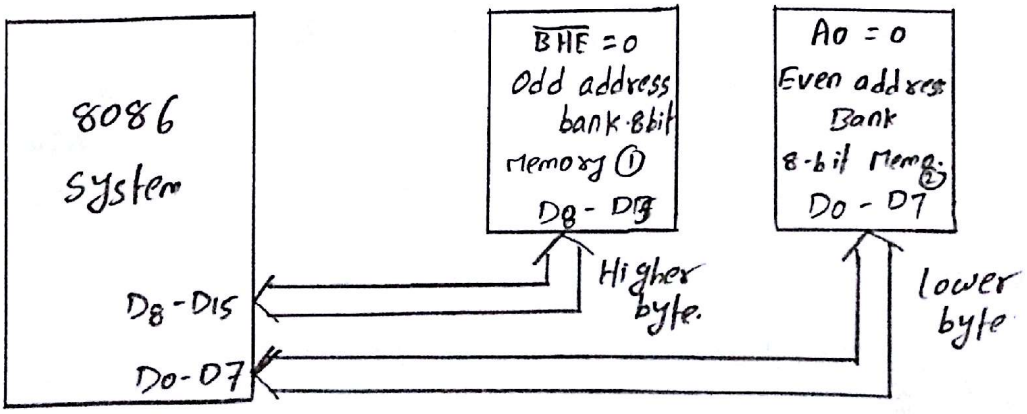


fig: Physical Memory Organization.

while referring to word data, the BIU requires one or two memory cycles, depending upon whether the starting byte is located at an even or odd address.

To read or write a complete word from/to memory, if it is located at an even address, only one read or write cycle is required. If the word is located at an odd address, the first read or write cycle is required for accessing the lower byte while the second one is required for accessing the upper byte. Thus, two bus cycles are required.

A map of an 8086 memory system starts at 00000H and ends at FFFFFH. 8086 being a 16-bit processor is expected to access 16-bit data to/from 8-bit commercially available memory chips in parallel, as shown below.

$\overline{BHE}$	$\overline{A_0}$	operation
0	0	complete word.
0	1	upper byte
1	0	lower byte.
1	1	none.

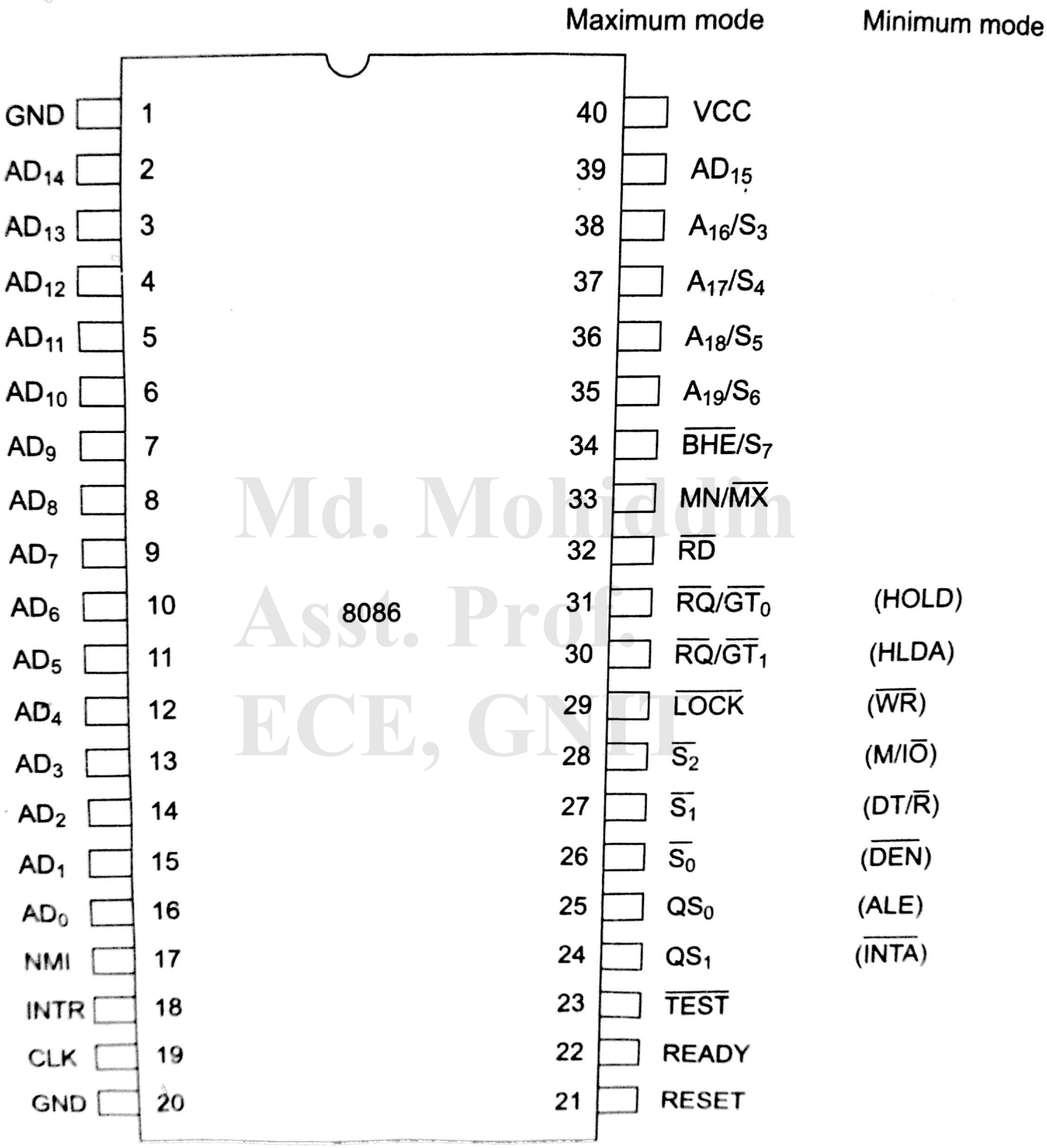


Fig. 1.5 *Pin Configuration of 8086*

8086 µP pins are divided into 3 types

1. Common mode pins
2. Minimum mode pins
3. Maximum mode pins

1. Common mode pins :-

$AD_0 - AD_{14}$:- (pin 2 to 16)

These are multiplexed Address & data lines. At 'T_1' clk cycle processor acess the address and then at the remaining clk cycles it processes

$AD_{15} / A_{16}/S_3 - A_{19}/S_6$:-

AD_{15} is a multiplexed Address & data line. $A_{16} - A_{19}$ are the Address lines

status lines gives the status of the processor

S_4	S_3	Memory access
0	0	Extra
0	1	stack
1	0	Code(or) None
1	1	data

S_5 - gives the status of interupt enable flag.

S_6 - it is always in the state of logical zero.

NMI :- (non-maskable interupt)

It activates for edge triggering. It is non-maskable and is given highest priority.

INTR :-

It erables for level triggering. It is maskable and is given lowest priority than NMI

HLDA :- (HOLD Acknowledgment)

It is the Response for HOLD signal

$\overline{WR}$:-

0 - MP performs writting operation

1 - no writting.

DT/$\overline{R}$:- (Data transmission/Receiver)

If DT/$\overline{R}$ is logic '1', data transfers to external devices (writting operation) otherwise Read.

DEN :- (Data enable)

If DEN is logic '1', valid data is availabe on data bus.

M/$\overline{IO}$:- (Memory/input output).

1 : Memory operation

0 : Io devices.

ALE :- (Asthematic latch enable)

1 : $A_0 - A_{19}$ (T_1)

0 : $D_0 - D_{15}$ & $S_3 - S_6$ $(T_2, T_3, \cdots T_N)$

$\overline{INTA}$:- (interrupt Acknowledgment)

It is a response to INTR signal.

3. Maximum mode pins :-

RQ/$\overline{GT_0}$ & RQ/$\overline{GT_1}$:-

Request for system bus in max mode by the external devices (DMA, others) sends through RQ/$\overline{GT_0}$ & RQ/$\overline{GT_1}$ & it sends grant single through same pins.

Clk:-

It is used to synchronizes all the external devices connected to it.

$\overline{BHE}/S_7$:-

$\overline{BHE}$ with the combination of A_0 gives the status of physical memory accessing

$\overline{BHE}$	A_0	Physical Memory
0	0	whole word
0	1	odd bank
1	0	Even bank
1	1	None

S_7: Not used

RESET:-

It clears all data and comes to the starting pt.

Ready:-

Slower devices synchronize using Ready pin. If it is active then only processor works on slower devices otherwise it enters into wait state

$\overline{TEST}$:-

This TEST signal examines the wait instruction. If test signal is active (0) then processor executes or execution will continue otherwise it will enter into ideal state

MN/$\overline{MX}$:-

1 : Minimum Mode

0 : Maximum Mode

2. Minimum Mode pins:-

HOLD:-

DMA makes HOLD high, for the Request of system bus.

$\overline{S_2}, \overline{S_1}, \overline{S_0}$:-

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	operation
0	0	0	Int. Acknowledgement
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

$\overline{LOCK}$:-

0 : System bus locked

1 : Accepts request of system bus (unlocked)

QS_0, QS_1 :-

QS_1	QS_0	operation
0	0	No operation
0	1	1st byte of opcode from queue
1	0	Empty queue
1	1	Subsequent byte from queue

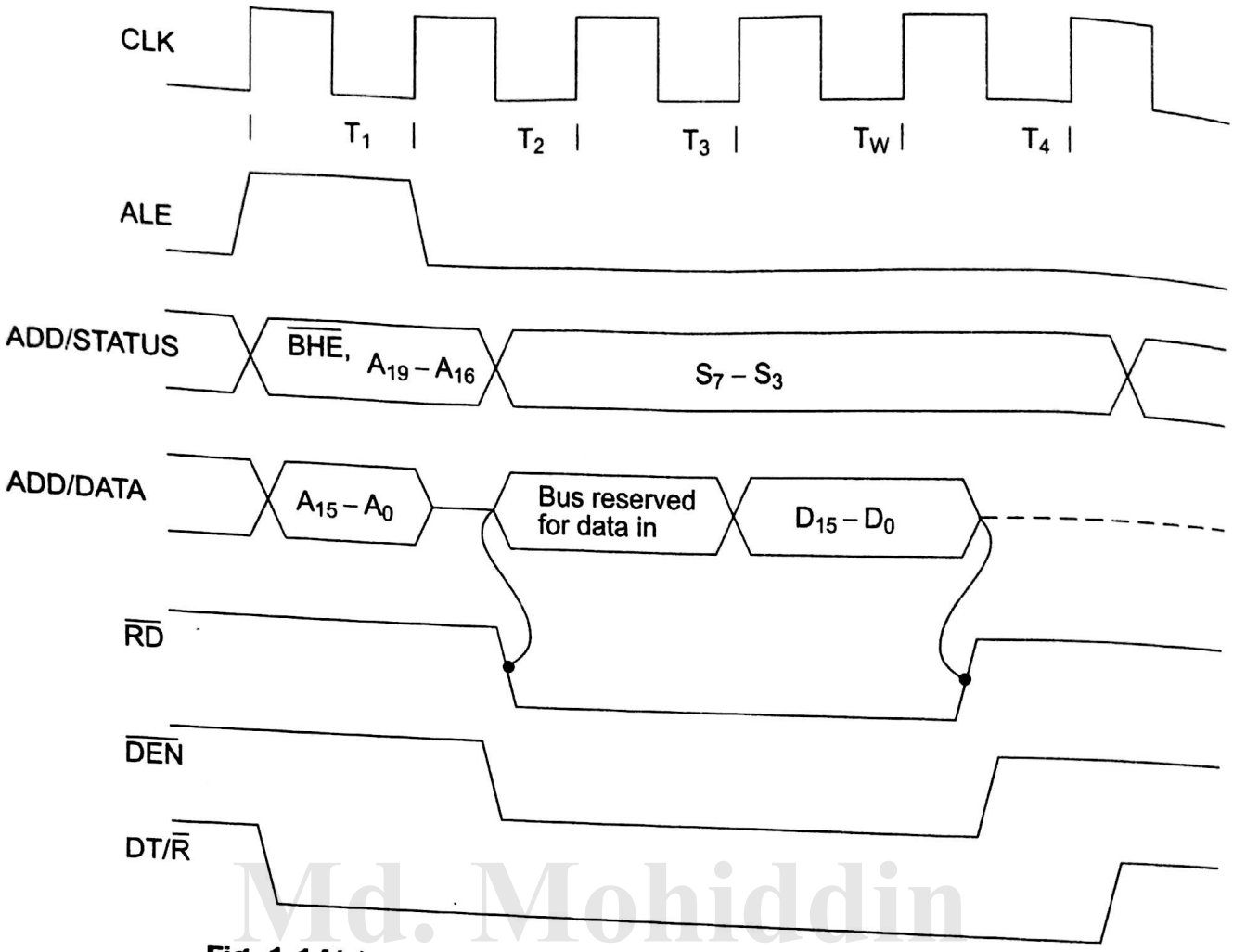


Fig. 1.14(a) *Read Cycle Timing Diagram for Minimum Mode*

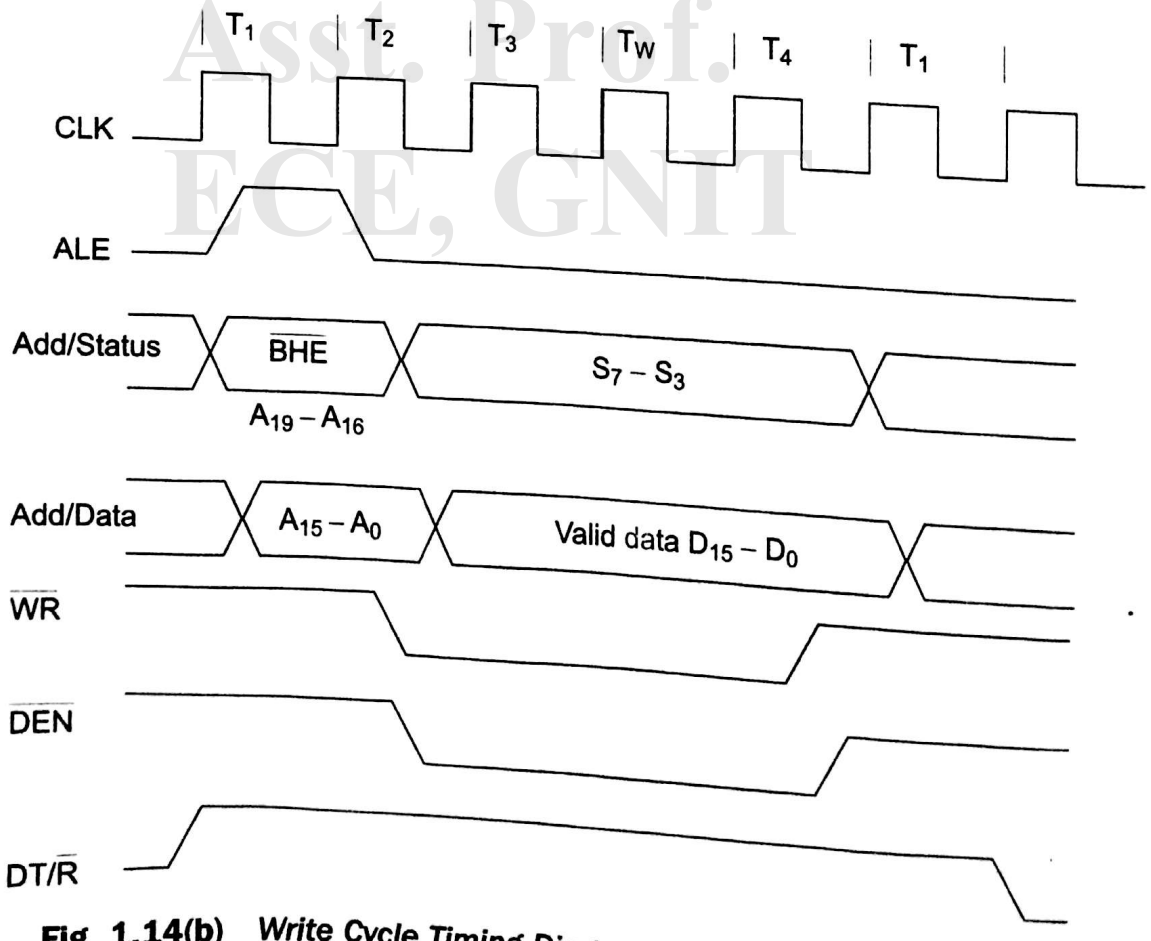


Fig. 1.14(b) *Write Cycle Timing Diagram for Minimum Mode Operation*

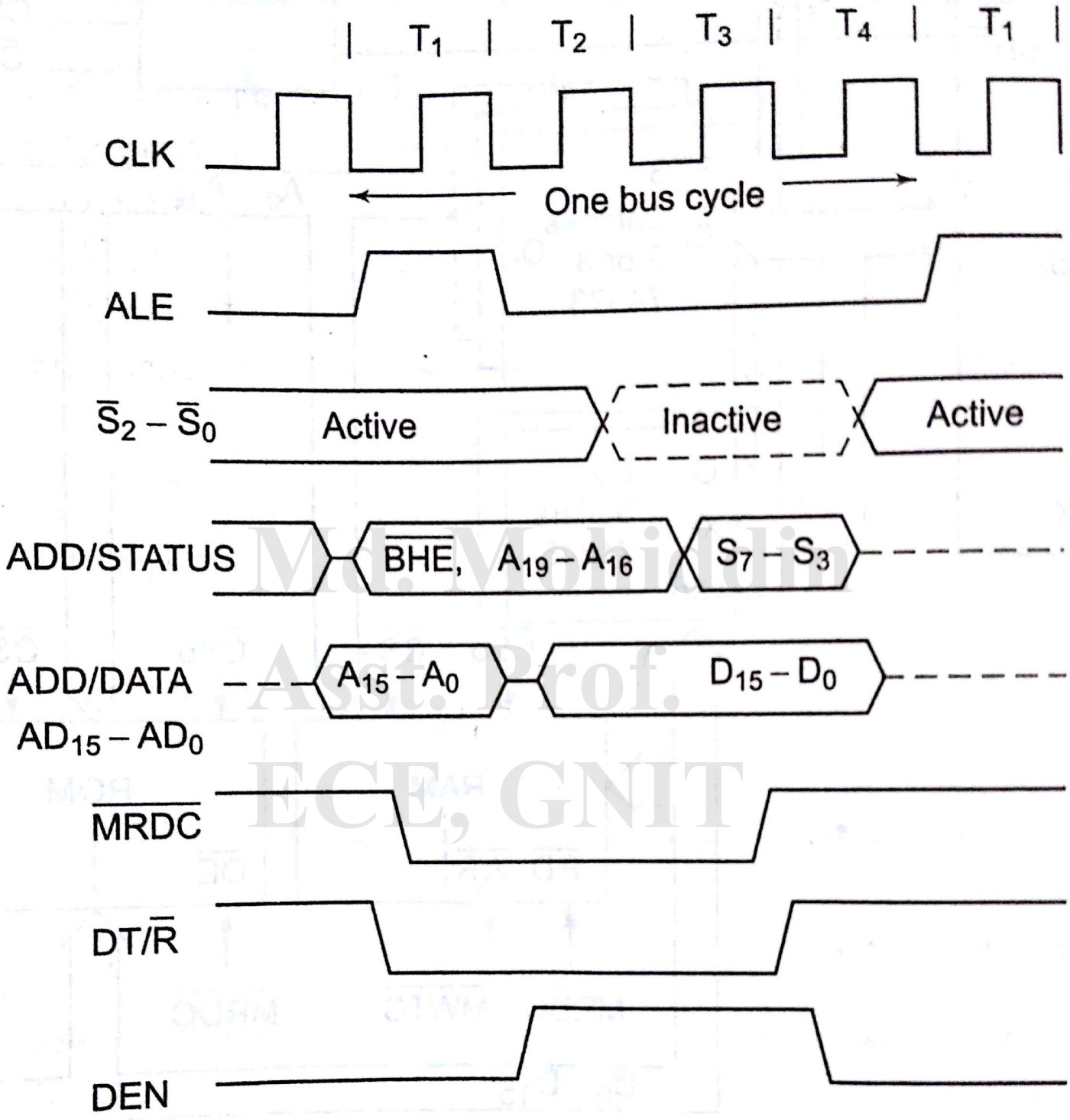


Fig. 1.16 (a) *Memory Read Timing in Maximum Mode*

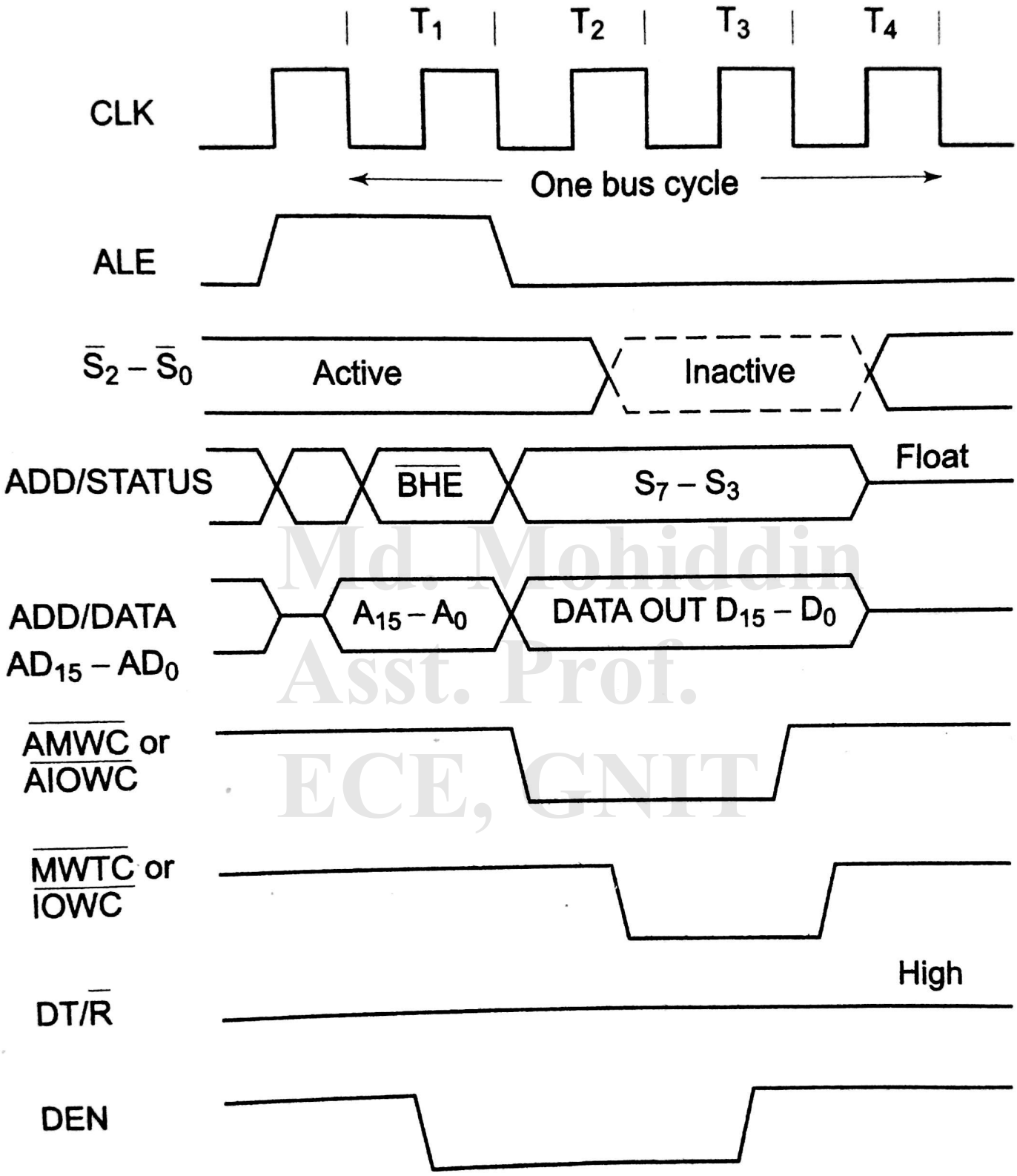


Fig. 1.16(b) ***Memory Write Timing in Maximum Mode***

Interrupts of 8086:

* The dictionary meaning of the word 'interrupt' is to break the sequence of operation. while the CPU is executing a program, an '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 transfessed back again to the main program which was being executed at the time of interruption.

* Broadly, these are two types of interrupts:
  1, External interrupt and 2, internal interrupt.

External interrupt: An external device or a signal, interrupts the processor from outside. eg: a keyboard interrupt

Internal Interrupt: Is generated internally by the Precessor circuit, or by the execution of an interrupt instruction.
eg: divide by zero interrupt, overflow interrupt etc.,

* In 8086, there are two interrupt pins: NMI and INTR.
* The NMI is a nonmaskable interrupt input pin which means that any interrupt request at NMI input cannot be Masked or disabled by any means.
* The INTR interrupt, however, may be masked using the Interrupt Flag (IF).

Programming Model

To run a program, a micro computer must have the program stored in binary form in successive memory locations,

label field	opcode field	operand field	comment field.
NEXT:	ADD	AL,07H	; add correction factor

There are three language levels that can be used to write a program for a microcomputer

Machine language

The binary form of the program is refessed to as machine language because it is the form required by the machine. However, it is difficult, if not impossible, for a programmer to memorize the thousands of binary instruction codes for a CPU such as the 8086. Also, it is very easy for an error to occur when working with long series of 1's & 0's.

Assembly language

To make programming easier, many programmers write programs in Assembly language. They then translate the assembly language program to machine language so that it can be loaded into memory and run.

Assembly language uses two-, three-, or four-letter mnemonics to represent each instruction type.

Assembly language statements are usually written in a standard form that has four fields.

label field	opcode field	operand field	comment field.
NEXT:	ADD	AL, 07H	; Add correction factor

A label is a symbol used to represent an address which is not specifically known at the time the statement is written. Labels are usually followed by a colon.

The opcode field of the instruction contains the mnemonic for the instruction to be performed. eg: Add

The operand field of the statement contains the data, the memory address, the port address, or the name of the register on which the instruction is to be performed.

Comment field, starts with a semicolon. Comments do not become part of the machine language program, You write comments in a program to remind you of the function that an instruction performs.

High-Level Languages : Another way of writing a program for a microcomputer is with a high-level language, such as BASIC, Pascal or C. These languages use program statements which are even more english like than those of assembly language. An interpreter program or a compiler program is used to translate higher-level language statements to machine codes which can be loaded into memory & executed.