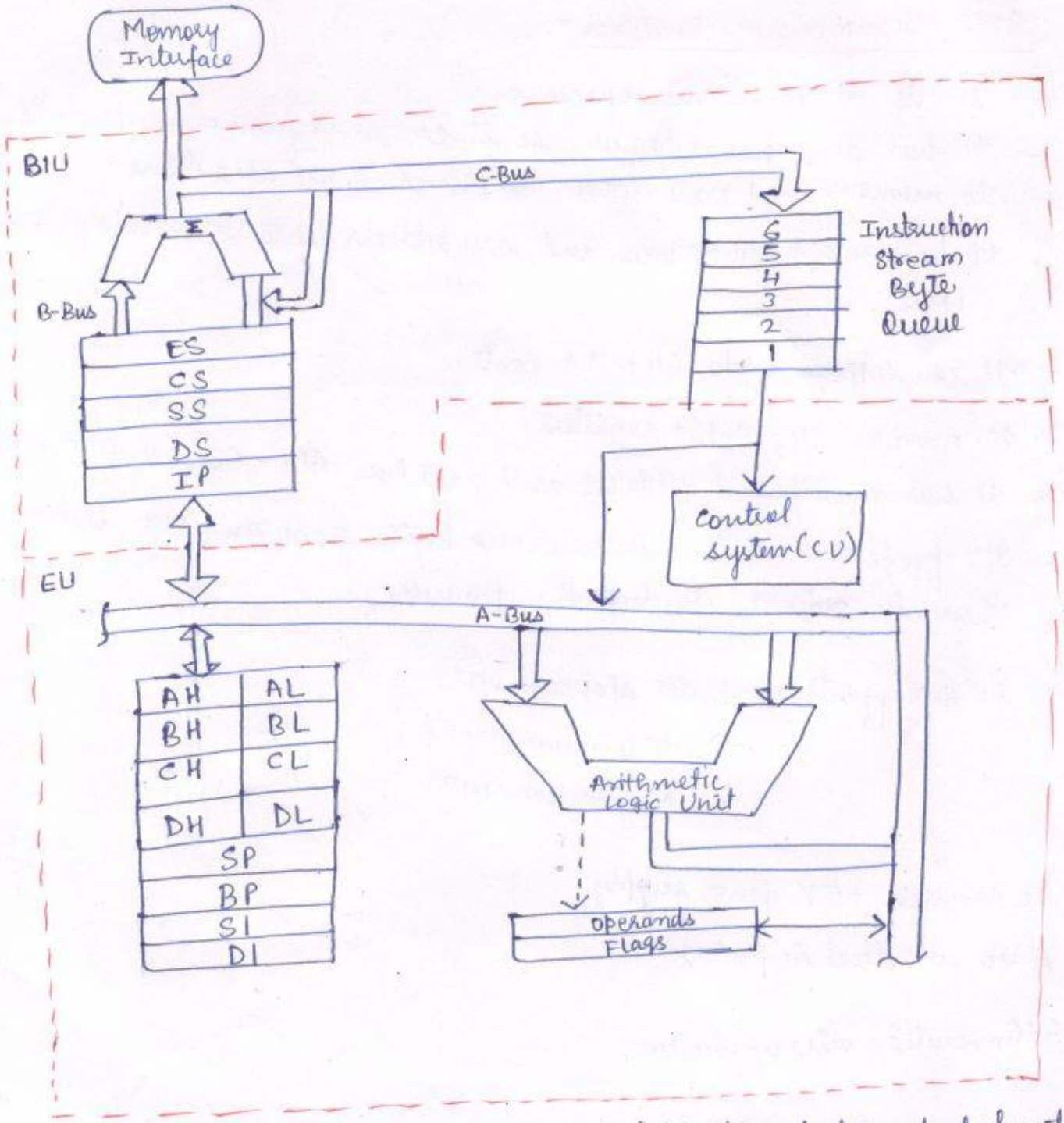
Minakshi

Unit-3

8086 Microprocessor Features:

→ It is a 16-bit microprocessor.
→ It has a 16-bit data bus, so it can read from or write data to memory and ports either 16-bit or 8-bit at a time.
→ It has 20 bit address bus and can address upto 2^{20} memory location (1MB).
→ It can supports upto 64K I/o ports.
→ It provides 14, 16-bit registers.
→ It has multiplexed address and data bus $AD_0 - AD_{15}$ and A_{16} & A_{19}
→ It prefetches upto 6 instructions bytes from memory and queue them in order to speedup the processing.
→ 8086 supports 2 modes of operation
  → Minimum mode
  → Maximum mode.
→ It requires +5V power supply.
→ A 40 pin dual in package.
→ Third Generation microprocessor.

Architecture of 8086 microprocessor:



The architecture of 8086 can be divided into two independent functional units

- Bus Interface Unit
- Execution unit

These two functional units can work simultaneously to increase the speed and hence the throughput.

Throughput ⇒ instruction executed per time unit.

Execution Unit:

- The execution unit of 8086 tells the BIU (Bus interface unit) to where to fetch the instructions or data from, decodes instructions, and execute instructions.
- EU contains control circuitry, which directs internal operations.
- A decoder in the EU translates instructions fetched from memory into a series of actions, which the EU carries out.
- The EU has a 16-bit ALU which can add, subtract, AND, OR X-OR, increment, decrement, complement or shift binary ~~operations~~ number

→ The main functions of EU are:

- Decoding of instructions
- Execution of instructions

steps
- EU extracts instructions from top of queue in BIU
- Decode the instructions
- generates operands if necessary.
- passes operands to BIU and requests it to perform read or write bus cycles to memory or I/O
- perform the operations specified by instruction on operands.

Bus Interface Unit:

- The bus interface unit provides interfacing facility to the outside world.
- It provides 16-bit data bus and 20-bit address bus.

Functions of BIU:

- It sends address of the memory or I/O
- It fetches instruction from memory.
- It reads data from port/memory

- It writes into port/memory.
- It supports instruction queuing.

Instruction Queue:

- To speedup program execution, the BIU fetches six instruction bytes ahead of time from the memory
- These prefetched ~~bytes~~ instruction bytes held for the execution unit in group of registers called **Queue**.
- With the help of queue it is possible to fetch next instruction when current instruction is in execution.
- Queue operates in FIFO principle.
- The EU (Execution Unit) fetches instruction codes from the queue.

Register Organization:

- 8086 has a powerful set of register known as ~~register~~ general purpose and special purpose registers.
- All registers are 16-bit registers.

→ **General purpose registers:**
- These registers can be used as either 8-bit registers or 16-bit registers
- These registers may be either used for holding data, variables and intermediate results temporarily or for other purposes like a counter or for storing offset address for some particular addressing modes etc.

Special Purpose:

→ These registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes.

→ The 8086 registers are classified into 4 groups:

- → General data registers
- → Segment registers
- → Pointers and index registers
- → Flag registers

General Data registers:

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

- The AX, BX, CX and DX are the general purpose 16-bit registers.
- AX is used as 16-bit accumulator. The lower 8-bit designated as AL and higher 8-bit is designated as AH. AL can be used as an 8-bit accumulator for 8-bit operation.

→ All data register can be used as either 16 bit or 8-bit.

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

→ The register CX is used as default counter in case of string or loop instructions.

→ DX register may be used as an implicit operand or destination in case of a few instructions.

Segment Registers:

There are 4 segments registers
- Code segment Register (CS)
- Data segment Register (DS)
- Extra segment register (ES)
- Stack Segment register (SS)

The 8086 architecture uses the concept of segmented memory. 8086 is able to address a memory capacity of 1 megabyte and it is byte organized. This 1MB memory is divided into 16 logical segments. Each segment contains 64KB of memory.

Code segment register (CS) is used for addressing memory locations in the code segment of the memory, where the executable program is stored.

CS value identifies the starting address of 64KB segment known as code segment

Data segment register: points to the data segment of memory where data is stored.

Extra Segment register: refers or points to the another data segment in the memory.

Stack segment register: is used for addressing stack segment of the memory. The stack segment is used to store stack data.

while addressing any location in the memory, the physical address is calculated from two parts: physical address = segment address + offset address.

Pointers and index registers:

The index and pointer registers are:

IP - instruction pointer - stores memory location of next instruction to be executed
BP - Base Pointer
SP - Stack Pointer
SI - Source Index
DI - Destination Index

→ The pointer register IP contains offset within the code segment.
→ The pointer register BP contains offset within the data segment
→ The pointer register SP contains offset within the stack segment.

→ Index registers are used as general purpose registers as well as for offset storage in case of indexed, base indexed and relative indexed addressing mode

The register SI is used to store offset of source data in data segment
The register DI is used to store offset of destination in the data or extra segment.

→ The index registers are particularly useful for string manipulation.

The 8086 flag register

The 8086 flag register contents indicate the result of computation in the ALU. It also contains some flag bits to control the CPU operations.

→ A 16-bit flag register is used in 8086.
→ Flag register is divided into two parts — [condition code or status flags
machine control flags.

condition code flag register:

~~i. 16 bit flag register~~

→ condition code flags

- → CF (carry Flag)
- → PF (Parity Flag)
- → AF (Auxiliary Carry Flag)
- → ZF (Zero Flag)
- → SF (Sign Flag)
- → OF (Overflow Flag)

carry Flag: This flag is set, when there is a carry out of MSB in case of addition ~~and~~ or borrow in case of subtraction.

Parity Flag: This flag is set to 1, if the lower byte of the result contains even number of 1's.

Auxiliary carry flag: it is set when, there is a carry from the lowest nibble.

Overflow flag: This flag is set if an overflow occurs, i.e if the result of a signed operation is large enough to accommodate in the destination register.

Machine Control flags:

These flags are used to control certain operations in microprocessor.

- → TF (Trap flag)
- → IF (Interrupt Flag)
- → DF (Direction Flag)

Trap flag: if this flag is set, the processor enters in the single step execution mode. The processor executes the current instruction and the control is transferred to the trap ~~calls~~ interrupt service routine.

Intersept flag: if this flag is set, the maskable intersepts are recognized by the microprocessor, otherwise they are ignored.

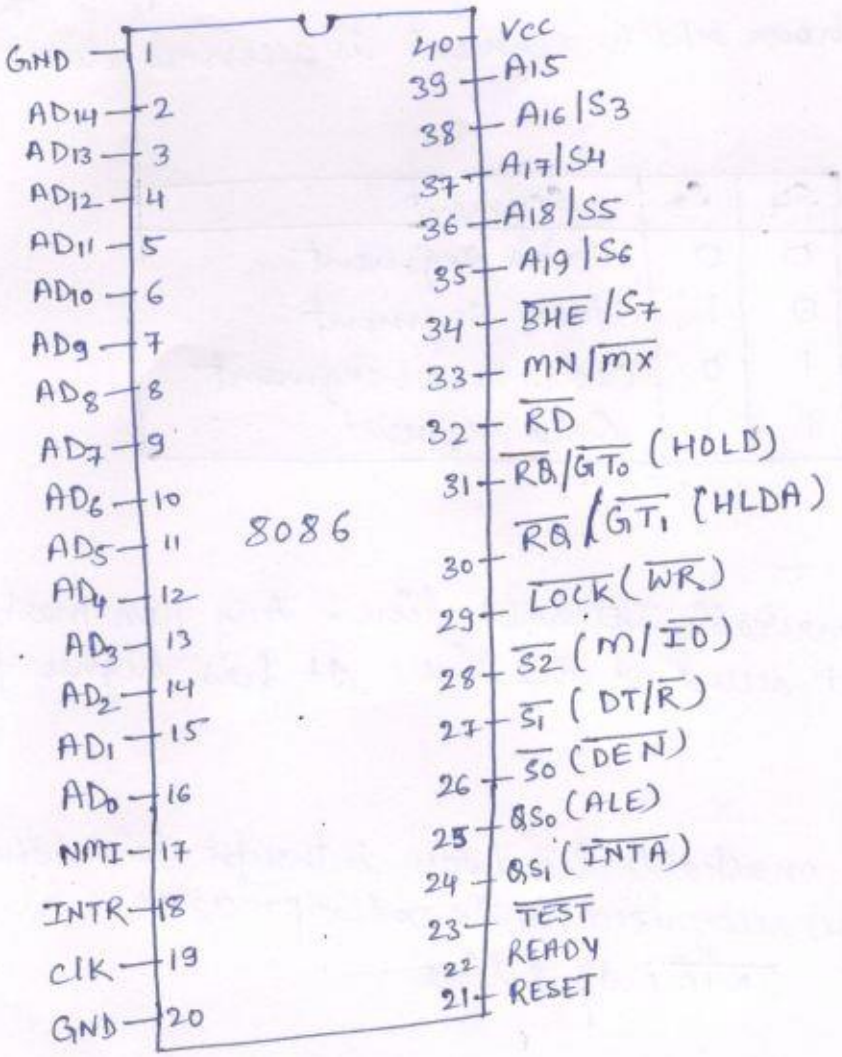
Direction Flag: This is used by the string manipulation instructions. If this flag bit is 0, the string is processed from the beginning (from lowestaddress to the highest address i.e auto-increment mode. Otherwise string is processed from the highest address towards the lowest address, i.e, auto decrementing mode.

Flag Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

U ⇒ means unspecified.

Pin Diagram of 8086



$AD_{15} - AD_0$ ⇒ Multiplexed Address and Data bus. It works 16 bit address bus during first machine cycle and as a 16-bit data bus during next machine cycles.

$A_{19}/S_6 - A_{16}/S_3$ ⇒ The address/status bus are multiplexed to provide address signal A_{19}-A_{16} and also status bits S_6-S_3.

* 8086 has 20 bit address bus and can address 2^{20} memory locations (1MB memory).

status signals: (S_6 S_5 S_4 S_3)

→ These status signals are multiplexed with address bits A_{19}, A_{18}, A_{17} A_{16}.

→ S_6 always remains at a logic 0.
S_5 indicates the condition of IF (interrupt flag)
S_4 and S_3 shows which segment is accessed during current bus cycle.

S_4	S_3	Segment
0	0	Extra Segment
0	1	Stack Segment
1	0	Code or no segment
1	1	Data segment

NMI → Non maskable Interrupt line.. Any non-maskable interrupt request occurs at this line. It has higher priority than INTR.

INTR: It is a maskable hardware interrupt line. When an interrupt request is recognized by the microprocessor, it sends logic 0 (zero) at $\overline{\text{INTA}}$ output line.

Reset: The reset input is used to provide a hardware reset for the 8086. It causes the process to immediately terminate its present activity. The signal must be active HIGH for at least 4 clock cycles.

CLK: The clock input provides the basic timing for processor operation and bus control activity. It is an asymmetric square wave with 33% duty cycle.

GND: Ground supply to the processor.

Vcc: +5 Volt power supply to microprocessor

$\overline{TEST}$: $\overline{TEST}$ input is tested by an WAIT instruction. 8086 will enter a wait state after execution of the WAIT instruction and will resume execution only when the $\overline{TEST}$ is made low by an active hardware.

QS_0 and QS_1 : The Queue state bits shows the state of internal instruction queue.

QS_1	QS_0	Function
0	0	Queue is idle
0	1	First byte of opcodes
1	0	Queue is empty
1	1	Subsequent byte of opcode

ALE: Address Latch Enable: This is for demultiplexing of address and data bus.

$\overline{INTA}$: it is an interrupt acknowledge and is a response to INTR. When the interrupt request will be accepted by the processor, $\overline{INTA}$ will be active low. The device will know that the processor has done into an interrupt acknowledge mode.

$\overline{S_0}$, $\overline{S_1}$ and $\overline{S_2}$: These status signals used by the 8086 bus controller to generate bus timing and control signals These are decoded as:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	machine Cycle
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/Inactive

$\overline{DEN}$: (Data Enable) It is signals external devices when they should put data on the bus, during read operation.

DT/$\overline{R}$ (Data Transmit or Receive): The direction of data transfer over the bus is signaled by the logic level output at DT/$\overline{R}$

DT/$\overline{R}$ = 1 (Bus is in transmit mode) ⇒ Data write to memory or I/O

DT/$\overline{R}$ = 0 (Bus is in Receive mode) ⇒ Data read from memory or I/O.

M/$\overline{IO}$: This signal tells whether a memory or I/O transfer is taking place over the bus:

M/$\overline{IO}$ = 1 ⇒ memory operation

M/$\overline{IO}$ = 0 ⇒ Input-output operation

$\overline{LOCK}$: (Bus Priority lock Control)

The 8086 output low on the $\overline{LOCK}$ pin while executing an instruction prefixed by LOCK to prevent other bus masters from gaining control of system bus.

$\overline{WR}$: write control signal. It is Low whenever processor writes data to memory or I/O.

$\overline{RB}$/$\overline{GT}$ (Bus Request/ Bus Grant). These requests are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. These pins are bidirectional. A Request on $\overline{GT_0}$ will have higher priority than $\overline{GT_1}$.

HOLD: For DMA transfer. The processor gets request on HOLD line for bus access for DMA transfer.

HLDA: The processor gives the acknowledgement of HOLD request.

$\overline{RD}$: Read control signal. It is low whenever processor reads data from memory or I/O port.

$MN/\overline{MX}$: Minimum or Maximum mode

This pin indicates in which mode the microprocess is operating

$MN/\overline{MX} = 1 \Rightarrow$ for minimum mode

$MN/\overline{MX} = 0 \Rightarrow$ for Maximum mode.

$\overline{BHE}$: (Bus High Enable) It is used to enable data onto the most significant half of data bus ($D_{15}-D_8$) during read or write operation.

S_7 : The state of S_7 is always a logic 1.

READY : This input is controlled to insert the wait states into the timing of the microprocessor. When Ready=0 the microprocessor enters into the wait state and remain idle.

This is for synchronizing slow devices.

common signals :-

1. $AD_{15} - AD_0$
2. $A_{19}/S_6 - A_{16}/S_3$
3. $\overline{BHE}/S_7$
4. $MN/\overline{MX}$
5. $\overline{RD}$
6. $\overline{TEST}$
7. READY
8. RESET
9. NMI
10. INTR
11. CLK
12. Vcc
13. GND

Minimum Mode signals :

1. HOLD
2. HLDA
3. $\overline{WR}$
4. $M/\overline{IO}$
5. $DT/\overline{R}$
6. $\overline{DEN}$
7. ALE
8. $\overline{INTA}$

Maximum Mode Signals:

1. $\overline{RQ}/\overline{GT_{0,1}}$
2. $\overline{LOCK}$
3. $\overline{S_2} - \overline{S_0}$
4. $\overline{QS_1} - \overline{QS_0}$

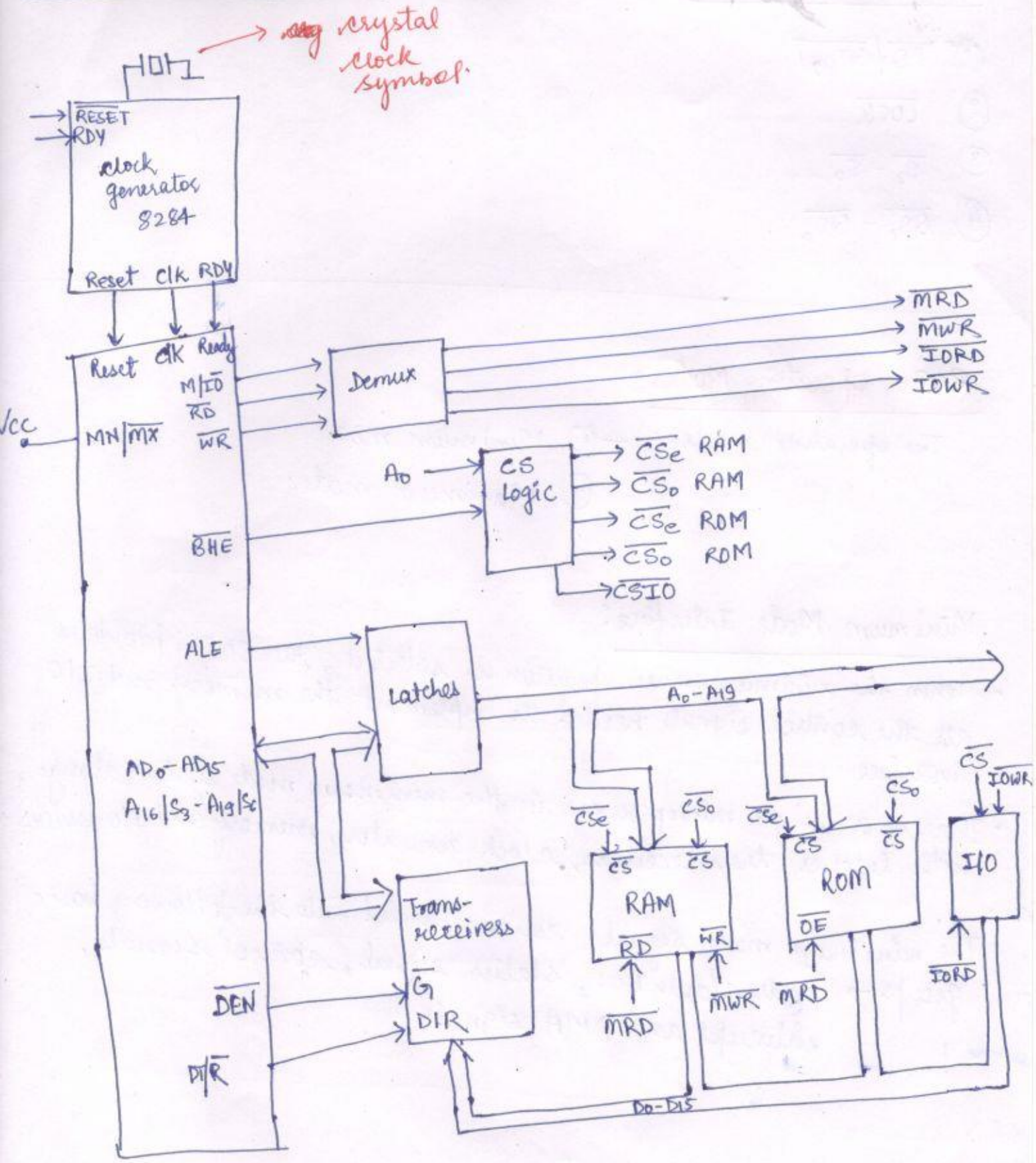
8086 operating Modes:

Two operating modes: — (1) Minimum mode
(2) Maximum mode.

Minimum Mode Interface:

- When the minimum mode operation is selected, the 8086 provides all the control signals needed to implement the memory and I/O interface.
- There is a single microprocessor in the minimum mode system along with latches, transreceivers, clock generator, memory and I/O devices.
- The minimum mode signals can be divided into the following basic groups → address/data bus, status signal, control signals, intercept and DMA signals.

minimum mode system organization:



DEN → Data Enable.
DT/R → Data Transmit or Receive
CS → chip select
CSe → chip select Even bank

CSo → chip select odd bank
DIR → Direction (transmit or Receive)
MRD → Memory Read
MWR → Memory write
IORD, IOWR → I/O Read, write

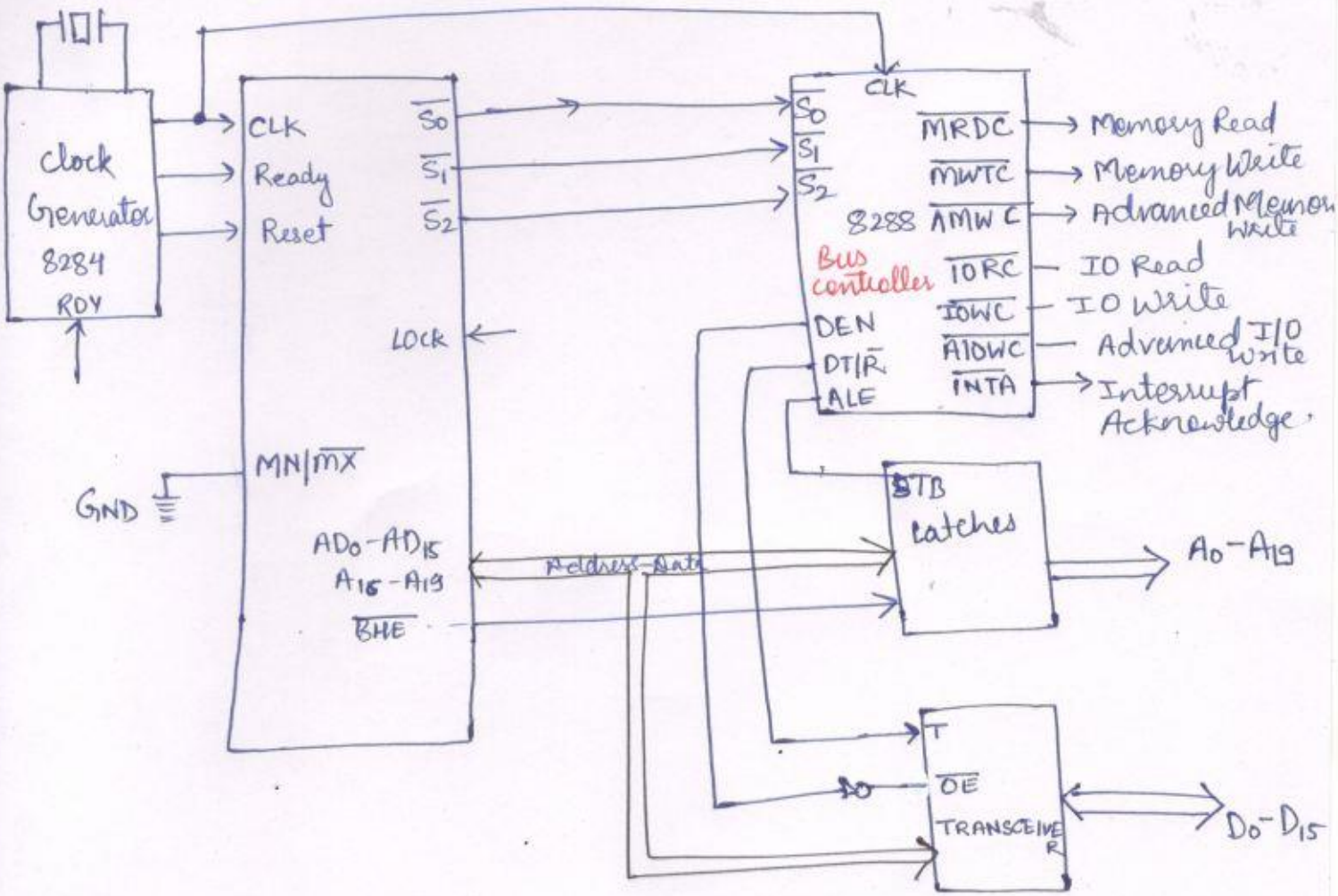
Maximum Mode Interface:

- When 8086 is set for maximum mode configuration, it provides signals for implementing a multiprocessor / co-processor system environment.
- In maximum mode, there may be more than one microprocessor.
- Usually in this type of system environment, there are some system resources that are common to all processors called global resources.
- There are some resources that are assigned to the specific processors called local or private resources.
- Two processors do not access the bus at the same time, one processor needs to pass the control of bus to the other, requires an another chip called bus controller.

8288 Bus Controller: The basic functions of bus controller chip is to derive control signals for memory or I/O. using the information (status) ~~available~~ provided by the processor.

Status Input

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CPU cycles	8288 command
0	0	0	Interrupt Acknowledge	$\overline{INTA}$
0	0	1	Read I/O Port	$\overline{IORD}$
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	$\overline{MWTC}$, $\overline{AMWTC}$
1	1	1	Passive	None ← no bus activity



Typical maximum mode configuration

8288 In maximum mode configuration, 8086 does not directly provide all the signals that are required to control memory, I/O and interrupt interfaces. Instead it outputs three status signals $\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$ to the external bus controller device, the bus controller generates the appropriately timed command and control signals.

Memory Segmentation:

→ Segmentation provides a powerful memory management mechanism.

→ Segmentation allows programmer to partition their programs into modules that operate independently of one another.

→ Segmentation provides a way to implement object oriented program

→ Segments allow two processes to share data easily.

→

→ Segment address has two components
- segment address
- offset

→ The size of the offset limits the maximum size of a segment.

→ In 8086 with 16 bit offset, a segment may be no longer than 64KB.

→ 1 MB memory of 8086 is partitioned into 16 segments - out of these segments only 4 segments can be active at any given instant of time (code segment, data segment, stack segment, and extra segment).

→ corresponding to these four segments, there are four registers
- → Code segment register
- → Data segment register
- → Extra segment register
- → Stack segment register

→ Maximum size of active memory for 8086 ⇒ 64 KB for program (code)
64 KB for stack
128 KB for data (DS+ES)

Memory Map of 8086

→ 1MB memory is divided into 16 blocks each consisting 64 KB.

→ Address lines ⇒ 20

- Number of locations addressed by address lines

$= 2^{20}$

Total memory in 8086.

$= 2^{20} \times 8$

$\Rightarrow 1024 \times 1024 \times 8$

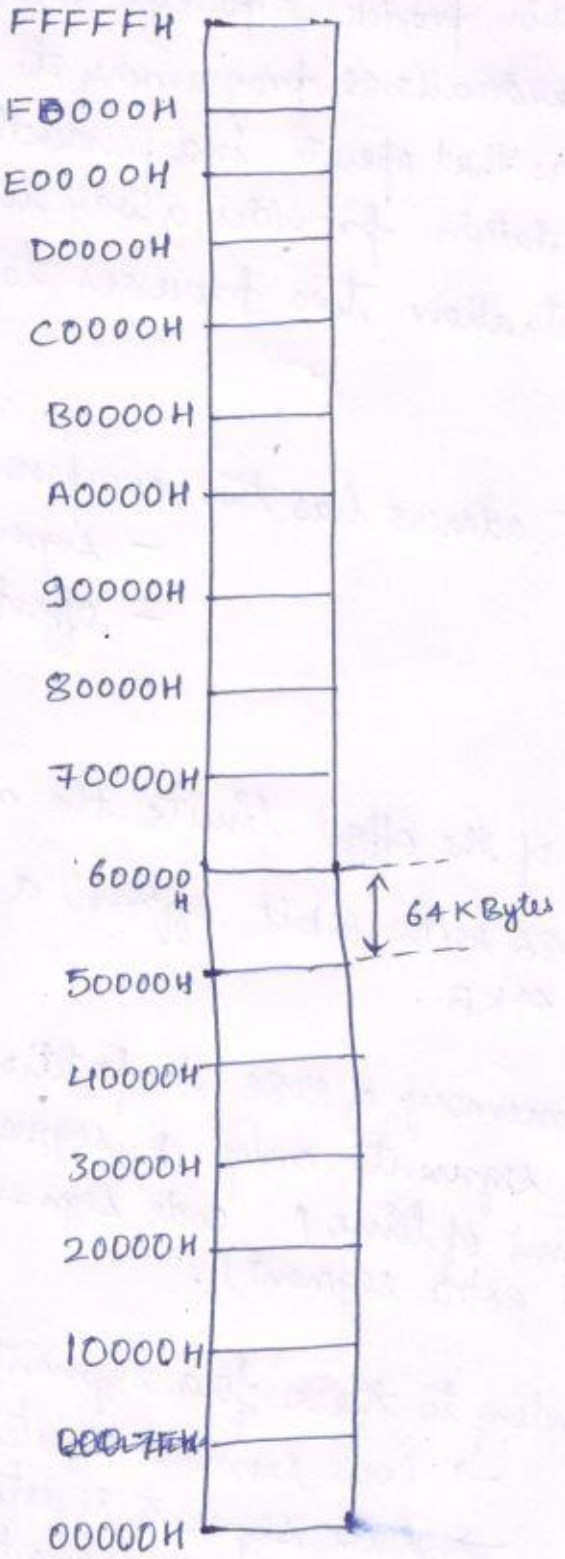
$\Rightarrow$ 1 MB

Minimum physical address

⇒ 00000 H

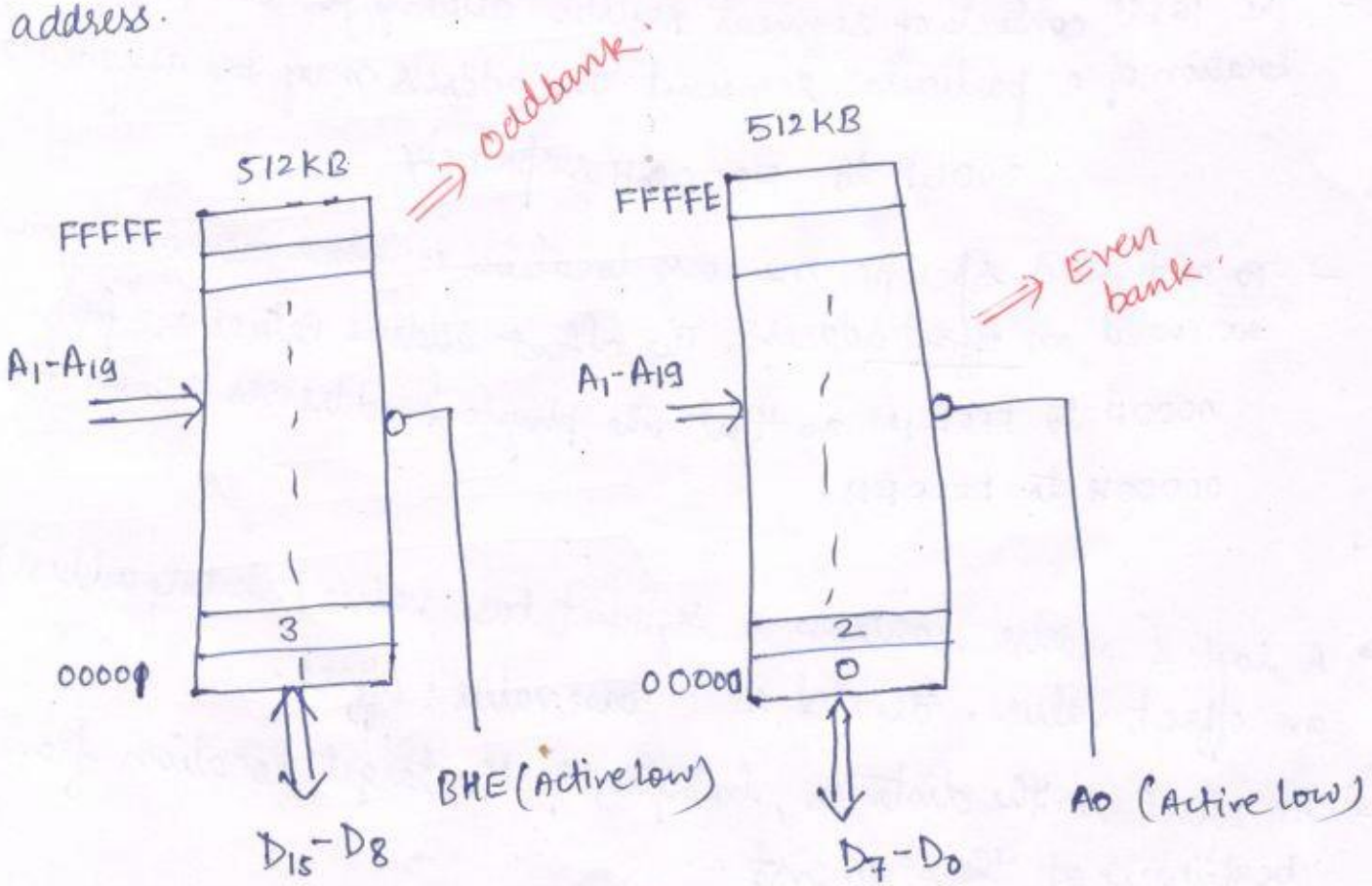
Maximum physical address

⇒ FFFFF H



Memory Bank:

- The memory is organized in the form of two banks.
- One memory bank contains all the even addressed locations like 00000, 00002, 00004 ..... and FFFFE. The data lines of this bank are connected to the lower eight data lines D_0 through D_7 of the 8086.
- The other memory bank has all odd addressed locations like 00001, 00003, ... FFFFF. The data lines of this bank are connected to the higher eight data lines D_8-D_{15} of 8086.
- A_0 is used to enable Even bank.
  BHE is used to enable odd bank.
- 1MB is devided into two 512KB (Even) and 512KB (odd). According to 512KB, 19 address lines are requeired to address each. So A_1 - A_{19} address lines are used to address.



BHE	A0	Bank Selected	Meaning
0	0	Both banks	16 bit whole word is transferred
0	1	Odd Bank	8 bit upper byte from/to odd address
1	0	Even Bank	8bit Lower byte from/to even address
1	1	None.	None.

Memory Segmentation:

- The memory in an 8086 based system is organized as segmented memory.
- The 8086 is able to access 1MB of physical memory (as it has 20 address lines)
- The complete 1 MB of memory can be divided into 16 segments, each of 64KB size and is addressed by one of the segment registers.
- The 16 bit contents of segment register actually points to the starting location of a particular segment. The address may be assigned as

  0000H to F0000H respectively.

- To address a specific memory location within a memory segment, we need an offset address. The offset address values are from

  0000H to FFFFH so that the physical addresses range from

  00000H to FFFFFH.

→ A logical address contains a segment base value (starting address) an offset value. denoted as Base value : offset.

→ An offset is the distance, in bytes, of the target location from the beginning of the segment.

→ 8086 can work with only four 64KB segments at a time.

→ These four memory segments are called:
- Code Segment
- Stack Segment
- Data Segment
- Extra Segment

Code Segment:

That part of memory from where BIU (Bus interface unit) is currently fetching instruction code bytes.

Stack Segment:

A section of memory set aside to store addresses and data while a subprogram executes.

Data and Extra Segments:-

used for storing data values to be used in the program.

Logical address Sources →

Type of memory Reference	Default segment Base	Alternative segment Base	offset
Instruction Fetch	CS	None	IP
Stack operation	SS	None	SP
variable	DS	CS, ES, SS	Effective address
String Source	DS	CS, ES, SS	SI
String Destination	ES	None	DI
BP used as Base Register	SS	CS, DS, ES	Effective address

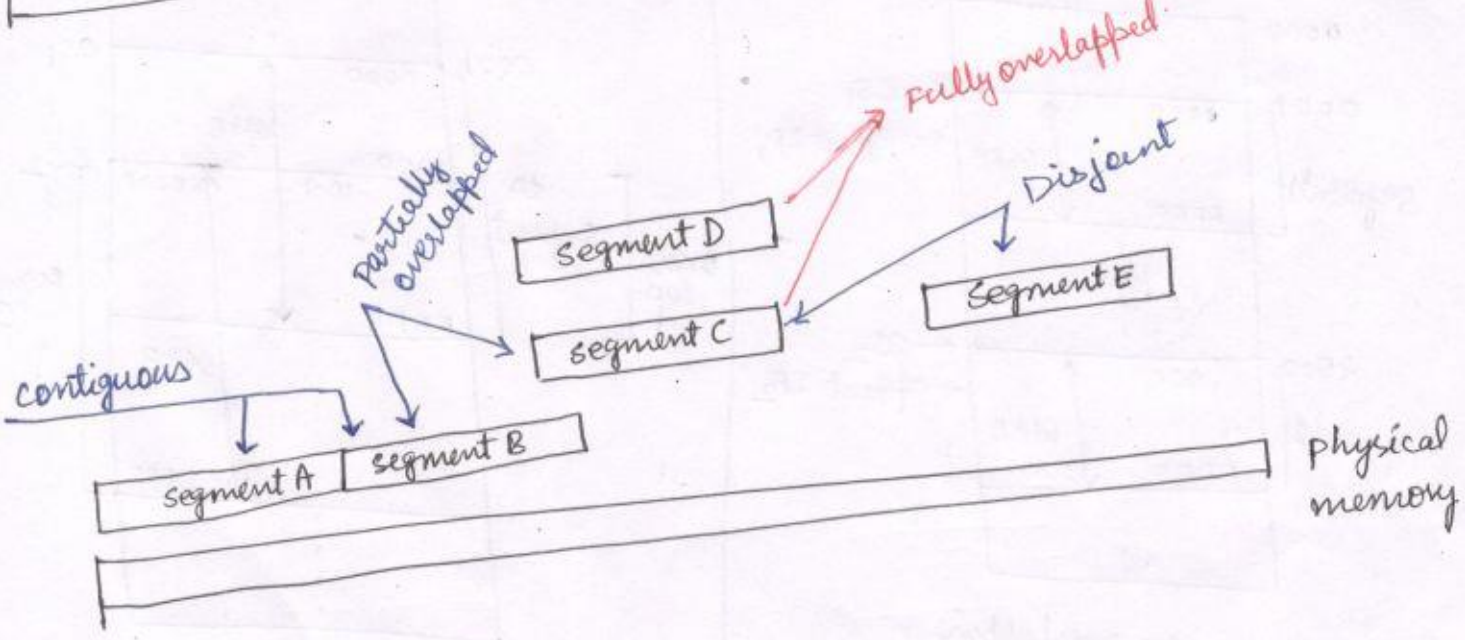
for solving question:⇒ where to look for the offset.

segment	offset Registers
CS	IP
DS	BX, DI, SI
SS	SP, BP
ES	BX, DI, SI

Memory Segmentation:

- 8086 views the megabyte of memory space as a group of segments.
- A segment is a logical unit of memory that may be upto 64KB byte long.
- Each segment is made up of contiguous memory locations and is independent, separately addressable memory unit
- Every segment is assigned (by software) a base address, which is its starting address in the memory space.
- All the segments begin on 16-bytes memory boundaries (restriction)
- Segments may be :
  - Adjacent
  - Disjoint
  - Partially overlapped
  - Fully overlapped.

* A physical memory may be mapped into (contained in) one or more logical segments.



Overlapping Segments: A segment starts at a particular address and its maximum size can go up to 64KB. But if another segment starts along with this 64KB location of the first segment, then two are said to be overlapping segments.

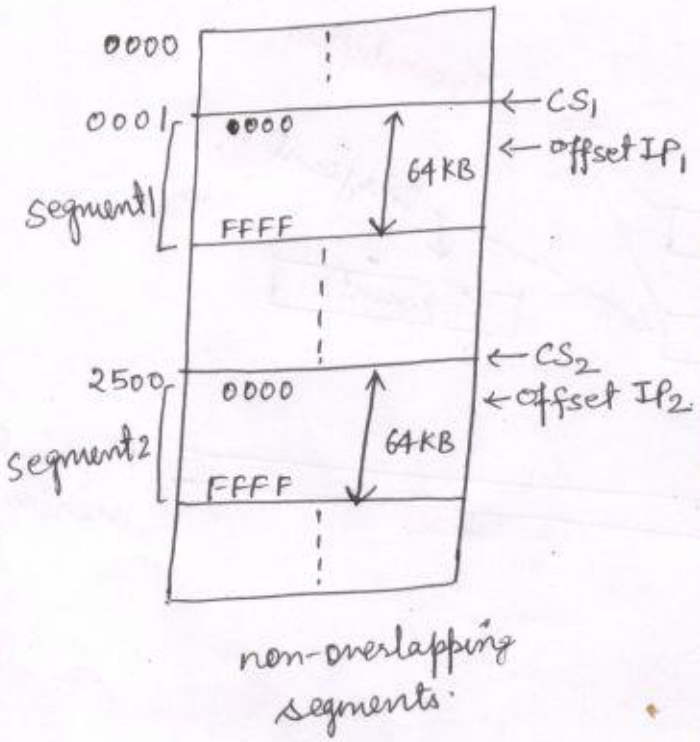
Non-overlapping

A segment starts after 64KB of a segment is called non-overlapping segment.

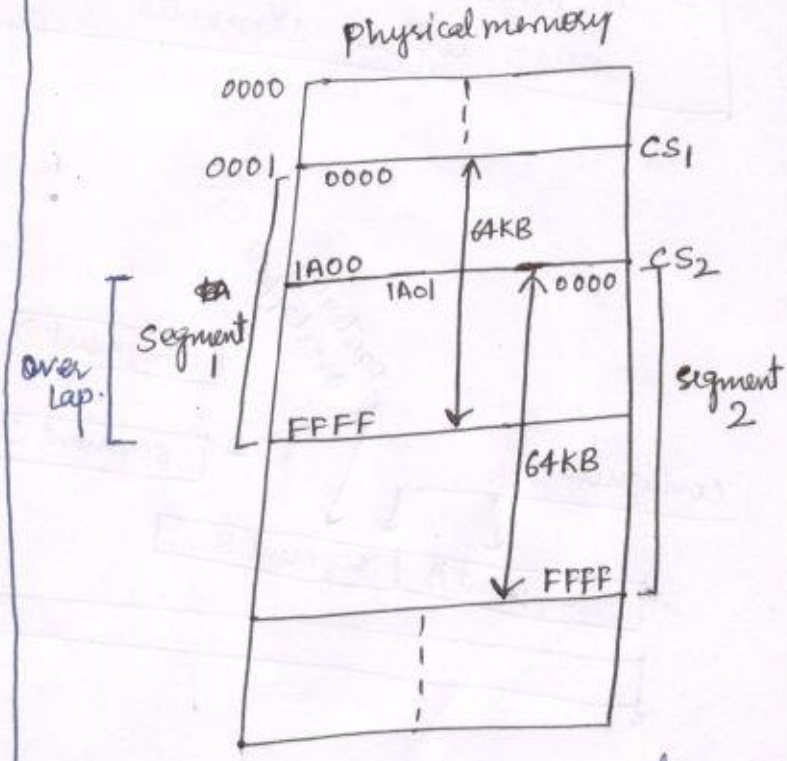
continuous segment:

A segments starts just after the location where first ends.

Disjoint Segment: A segment neither overlapping nor continuous will be a disjoint segment.



non-overlapping segments.



Overlapping segments.

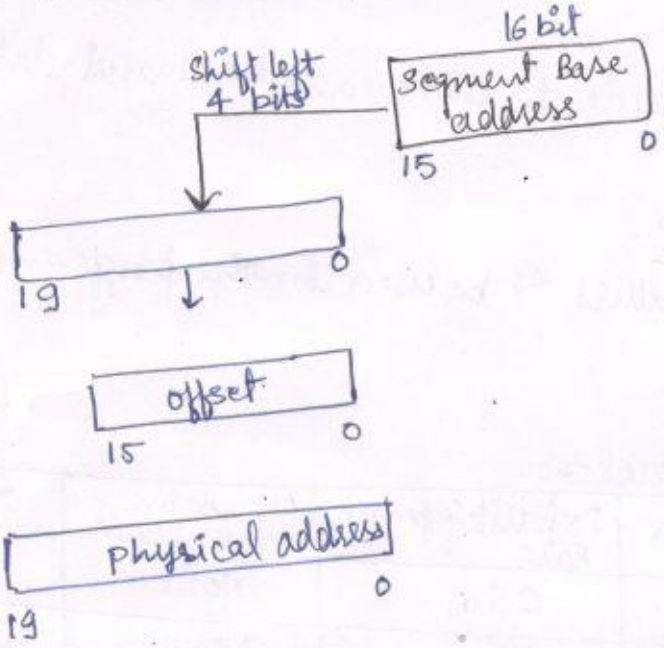
Memory Addressing

Physical Address Generation:

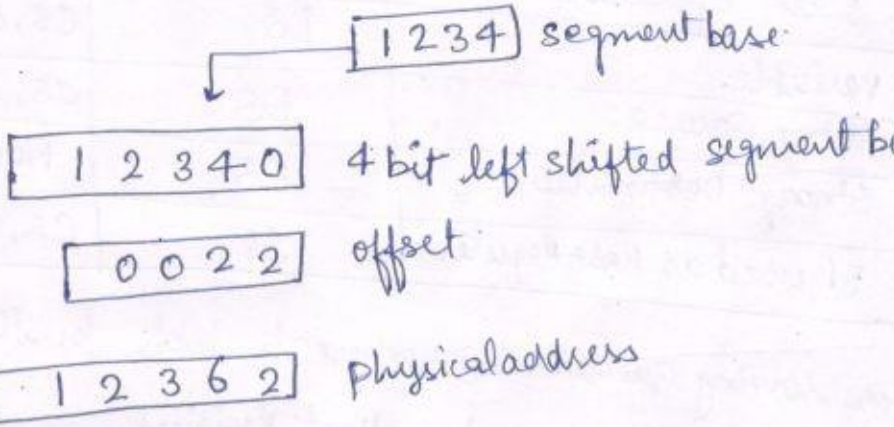
- A physical address is 20 bit value that uniquely identifies each location in 1 megabyte space.
- physical address may range from 00000H to FFFFFH.
- Since 8086 has 16 bit register. So the address can be stored in 16 bit on

So → Base address ⇒ 16 bit may range 0000H to F000H
offset ⇒ 16 bit may range 0000H to FFFFH

- **20 bit physical address generation:**



Example



Ques: Logical address 1005:5555 H calculate physical address.

Ans:- Give Logical address 1005:5555

1005 → starting segment address
5555 → offset.

① shift 4 bits left segment base address (1005) ⇒ 10050

```
  1   0   0   5   0
  1 0 0 0   0 0 0 0   0 1 0 1   0 0 0 0
+                     0 1 0 1   0 1 0 1
-------------------------------------------
  1 0 0 0   0 1 0 1   0 1 0 1   1 0 1 0   0 1 0 1
  ---------------------------------------------------
      1         5         5         A         5
```

Base address: 1005 H
offset : 5555 H
physical address: 15A5 H

Ques: The content of the following registers are

CS = 1111 H
DS = 3333H
SS = 2526 H
IP = 1232 H
SP = 1100 H
DI = 0020H

calculate the corresponding physical addresses for the address bytes in CS, DS and SS.

Ans:- ① CS = 1111 H , IP = 1232 H

```
CS ⇒   0 0 0 1   0 0 0 1   0 0 0 1   0 0 0 0   0 0 0 0
IP ⇒ +                     0 0 0 1   0 0 1 0   0 0 1 1   0 0 1 0
-------------------------------------------------------------------
       0 0 0 1   0 0 1 0   0 0 1 1   0 1 0 0   0 0 1 0
       ---------------------------------------------------
Physical address ⇒  1       2       3       4       2 H
```

(2) DS = 3333H DI=offset ⇒ 0020H

DS⇒ 0011 0011 0011 0011 0000
DI⇒ + 0000 0000 0010 0000
0011 0011 0011 0101 0000
3 3 3 5 0

physical address = 33350 H

(3) SS= 2526H SP= 1100H

SS⇒ 0010 0101 0010 0110 0000
SP⇒ + 0001 0001 0000 0000
0010 0110 0011 0110 0000
2 6 3 6 0

physical address ⇒ 26360H

Advantages of Segmentation:

→ It allows modularity (program can be divided into modules)
→ It allows to processes easily share data.
→ It allows address can be handled using 16 bit register, without segmentation, it requires 20 bit registers.
→ It permits a program and/or its data to be put into different areas of memory everytime the program is executed.

Rules for Segmentation

- The starting address of a segment should be such that it can be evenly divided by 16.
- The minimum size of a segment can be 16 bytes
- The maximum size of a segment can be 64 KB.

Physical Address:

- 20 bit address of physical memory in 8086.
- physical address in 8086, is obtained by shifting the segment address 4 bits to the left and adding the offset address.

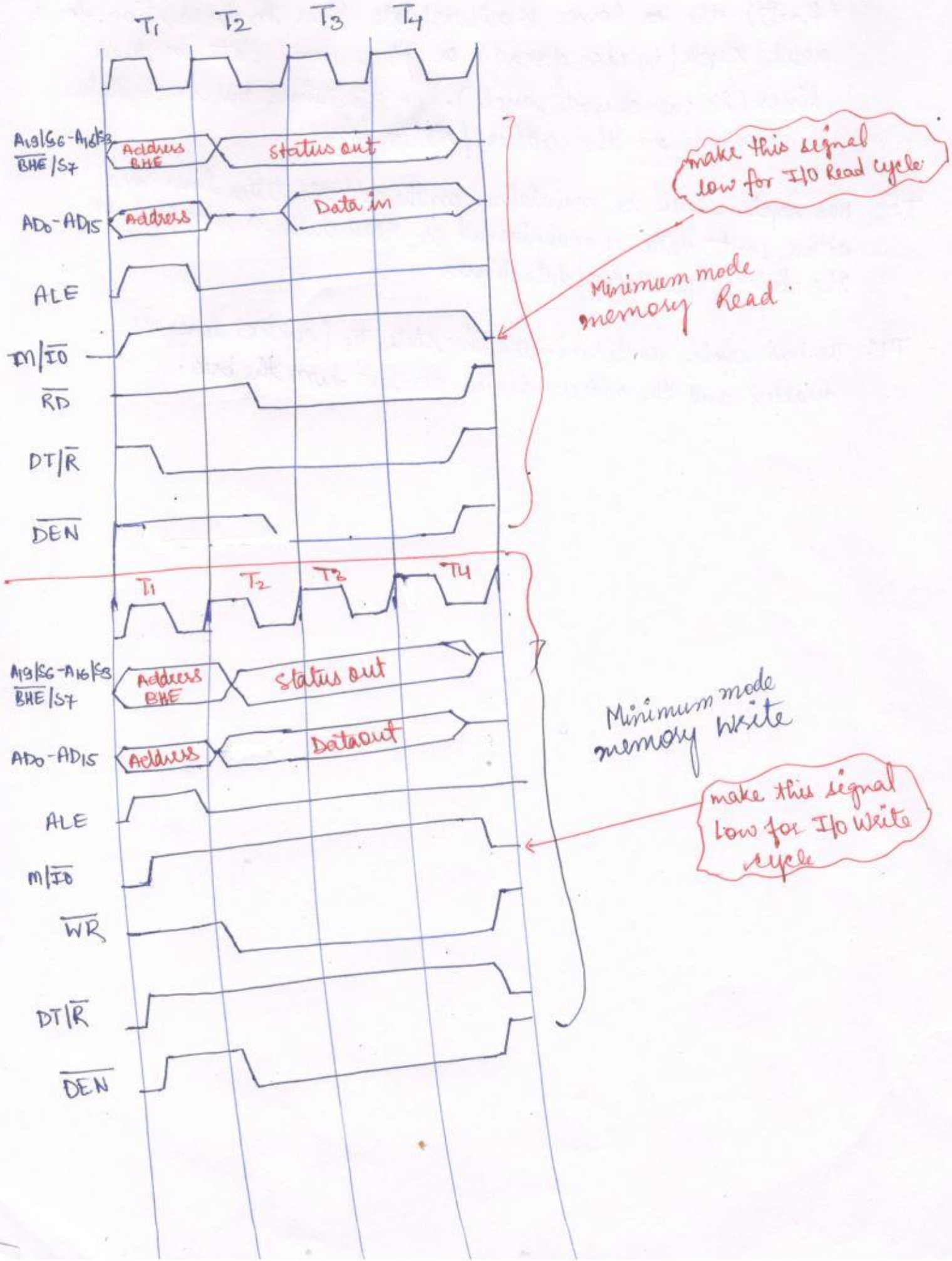
Logical Address:

- Logical address is specified as segment : offset
- Here segment is the base address (starting address) of a segment stored in any base segment register.
- Offset is the displacement or distance from the beginning of the segment (base of segment).

Effective address:

- In 8086, effective address is also referred to as offset address.
- it is given in number of bytes from the starting of segment base address.

8086 Read cycle & 8086 write cycle



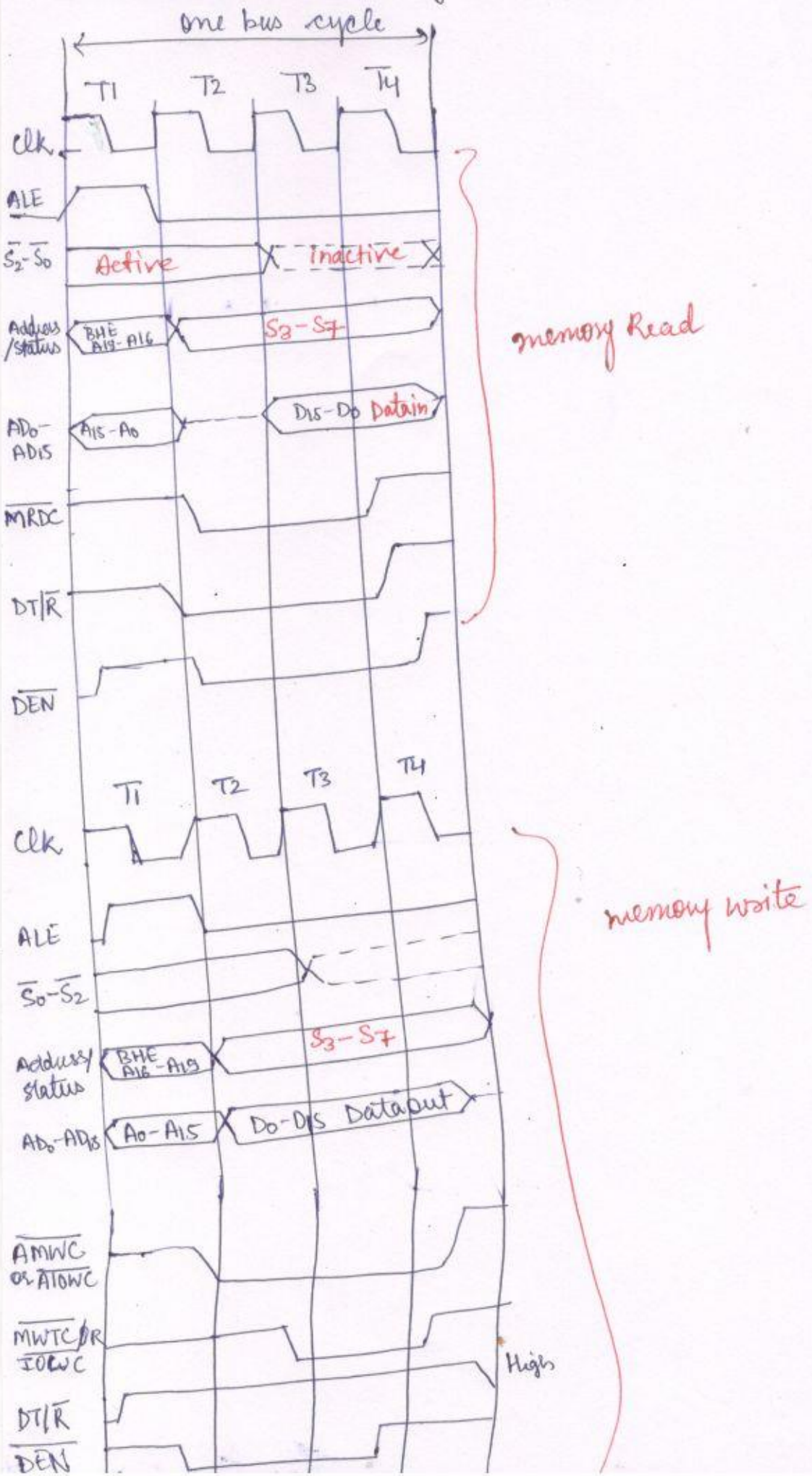
T1: 8086 places a 20 bit address on multiplexed address/data bus.

T2: the CPU removes the address from the bus and either three-states (floats) the lower 16 address/data lines in preparation for a read cycle (in case of read) or places write data on these lines (in case of write cycle). At this time, bus cycle status is available on the address/status lines.

T3: Bus cycle status is maintained on the address/status lines and either write data is maintained or read data is sampled on the lower 16 address/data lines.

T4: The bus cycle is terminated in state T_4 (control lines are disabled and the address device deselects from the bus.

Maximum Mode (Memory Read cycle)



Addressing modes of 8086 microprocessor:

- Every instruction of a program has to operate on a data
- The different ways in which a source operand is denoted in an instruction are known as addressing modes.

- Different types of addressing modes are:
  - Register Addressing Mode
  - Immediate addressing Mode
  - Direct addressing Mode
  - Register Indirect addressing Mode
  - Based addressing Mode
  - Indexed addressing Mode
  - Based Indexed addressing Mode
  - String addressing Mode
  - Direct I/O port addressing Mode
  - Indirect I/O port addressing Mode
  - Relative addressing Mode
  - Implied addressing Mode

---

① Register Addressing mode:

The instruction specifies the name of the register which holds the data to be operated by the instruction.

Example MOV CL, DH

The content of a 8-bit register DH is moved to another 8-bit register CL.

Immediate Addressing:

In immediate addressing mode, an 8-bit or 16-bit data is specified as part of the instruction.

Example: MOV DL, 08H

The 8-bit data given in the instruction is moved to DL

MOV AX, 0A9FH

The 16-bit data given in the instruction is moved to AX register.

Memory Access:

- 8086 has 20 address lines
- 8086 can address upto 1MB memory.
- However the largest register is only 16 bits. So the physical address will have to be calculated.
- Physical address is the actual address of a byte in the memory i.e the value which goes onto the address bus.
- Memory address is represented in the form:

  Segment: offset

- Each time, the microprocessor wants to access the memory it takes the contents of segment register, shifts it one hexadecimal place to the left (same as multiplying by 16) then add the required offset to form the 20-bit address.

Direct addressing Mode:

- Here the effective address of the memory location at which the data (operand) is stored is given in the instruction
- The effective address is just a 16-bit number written directly in the instruction.

```
MOV  BX, [1354 H]
```

[ ] → shows the contents of

- This instruction will copy the contents of a memory location into BX register.
- This addressing mode is called direct because the displacement of the operand from the segment base is specified directly in the instruction.

Register Indirect Addressing:

- In register indirect addressing, the name of register which holds the effective address will be specified in the instruction.
- Registers are used to hold EA are any of the following registers

  BX, BP, SI and DI.

- The contents of DS register is used for base address calculation.

Example

```
MOV  CX, [BX]
```

→ contents of register

EA (Effective Address) = (BX)
BA (Base address) = (DS) * 16_{10}
MA (Memory Address) = BA + EA

CX ← (MA) or
CL ← (MA)
CH ← (MA+1)

Based Addressing:

- In based addressing, BX or BP is used to hold the base value for the effective address and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction.
- In case of 8-bit displacement, it is sign extended before adding to the base value.
- when BX holds the base value of EA, 20 bit address physical address is calculated from BX and DS.
- when BP holds the base value of EA, BP and SS is used.

Example

MOV AX, [BX + 08H]

operations:

0008 ← (sign extended) 08H

EA = (BX) + 0008H

BA = (DS) * 16_{10}

MA = BA + EA

(AX) ← (MA) or AL ← (MA)
AH ← (MA+1)

Indexed Addressing:

- SI or DI register is used to hold an index value for memory data and 8-bit signed or 16-bit unsigned displacement will be specified in the instruction.
- This displacement is added to the index value in SI or DI to obtain the EA.

- In case of 8-bit displacement, it is sign extended to 16 bit before adding to the base value.

Example MOV CX, [SI + 0A2 H]

operations

$FFA2_H \xleftarrow{\text{sign extended}} 0A2_H$

$EA = (SI) + FFA2_H$

$BA = (DS) * 16_{10}$

$MA = BA + EA$

$(CX) \leftarrow (MA)$ or $CL \leftarrow [MA]$

$CH \leftarrow [MA+1]$

Based Indexed Addressing:

In based indexed addressing, the effective address is computed from the sum of a base register (BX or BP), an index register (SI or DI) and a displacement.

Example MOV DX, [BX + SI + 0A]

operations:

$000A_H \xleftarrow{\text{sign extended}} 0A_H$

$EA = (BX) + (SI) + 000A$

$BA = (DS) * 16_{10}$

$MA = BA + EA$

$(DX) \leftarrow [MA]$

String Addressing :

- employed in string operations to operate on string data.
- The effective address (EA) of source data is stored in SI register and the EA of destination is stored in DI register.
- The segment register for calculating base address of source data is DS and that of the destination data in ES.

Example: MOVS BYTE

operations: calculation of source memory address

$EA = (SI)$

$BA = (DS) * 16_{10}$

$MA = BA + EA$

calculation of destination address

$EA = (DI)$

$BA = (ES) * 16_{10}$

$MA = BA + EA$

$(MA_E) = (MA)$

IF DF = 1 then $(SI) \leftarrow (SI) - 1$ and $(DI) \leftarrow (DI) - 1$

IF DF = 0 then $(SI) \leftarrow (SI) + 1$ and $(DI) \leftarrow (DI) + 1$

Direction Flag

I/O Port Addressing

- These addressing modes are used to access data from standard I/O mapped devices or ports.
- In direct port addressing mode, an 8-bit address is directly specified in the instruction.

Example: IN AL, [09H]

operations: $port_{addr} = 09_H$

$(AL) \leftarrow (port_{addr})$ content of port with address 09_H is moved to AL register.

In indirect port addressing, the instruction will specify the name of the register which holds the port address.

- In 8086, 16-bit port address is stored in DX register.

EXAMPLE: OUT [DX], AX

operations: $PORT_{addr} = (DX)$

$(PORT_{addr}) = (AX)$

The contents of AX is moved to port whose address is specified by DX register.

Relative Addressing:

In this addressing mode, the effective address of a program instruction is specified relative to Instruction Pointer (IP) by an 8-bit signed displacement.

Example: JZ 0AH

operations: $000A_H \leftarrow$ sign extended $0A_H$

Zero flag.

IF ZF = 1 then

$EA = (IP) + 000A_H$

$BA = (CS) * 16_{10}$

$MA = BA + EA$

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

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

Implied Addressing :

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

Example CLC

This clears the carry flag CF to zero.