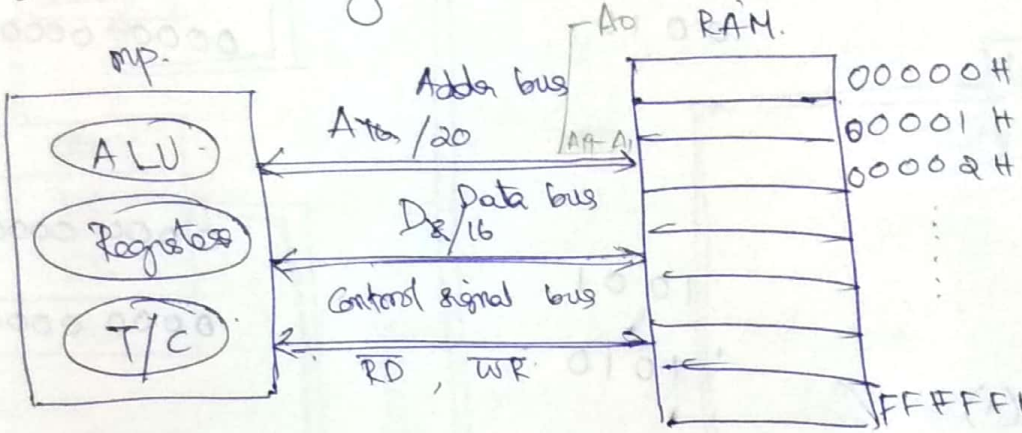
(MM) MICROPROCESSORS & MICROCONTROLLERS

MODULE - 1

* Basic Organisation of a Microprocessor :

A microprocessor consists of ALU (to manipulate data & instructions), Registers (to store temporary data) & Timing and Control Unit.



The ALU is provided with data & instns to manipulate on.
This data & instn is stored inside the memory called RAM.

→ In RAM, each location can store 8 bits and the address of each location is of 16 (8085) / 20 (8086) bits.

→ The address is expressed in hexadecimal and data in binary.

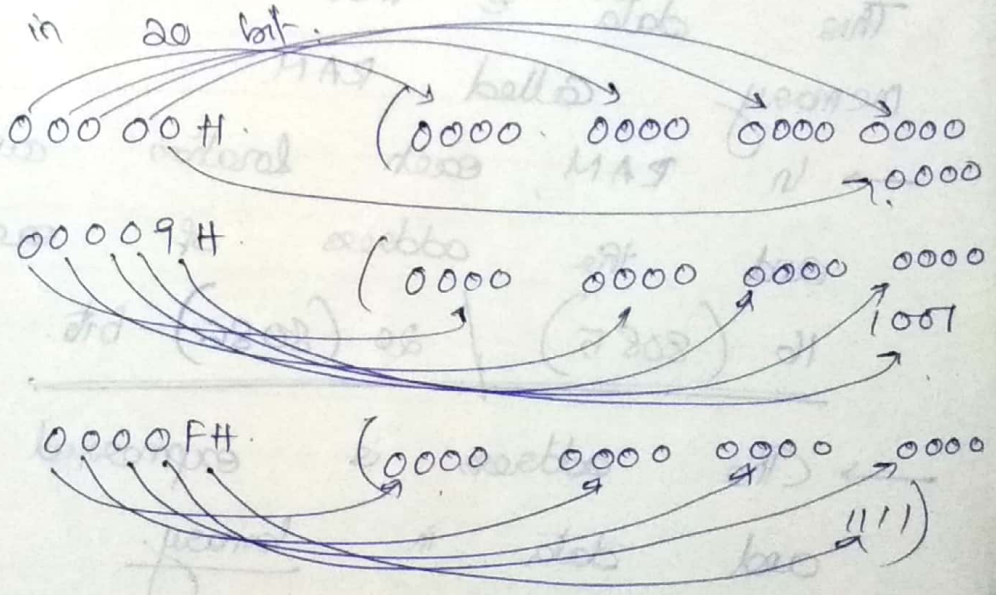
→ The address bus transfers address (usually A_{16} (8085) / A_{20} (8086). and the data bus is usually of D_8 (8086) / D_{16} (8086).

∴ In 8086, the data is manipulated as 16 bit (D_{15}) and address as 20 bits (A_{20}).

Decimal to Heradecima

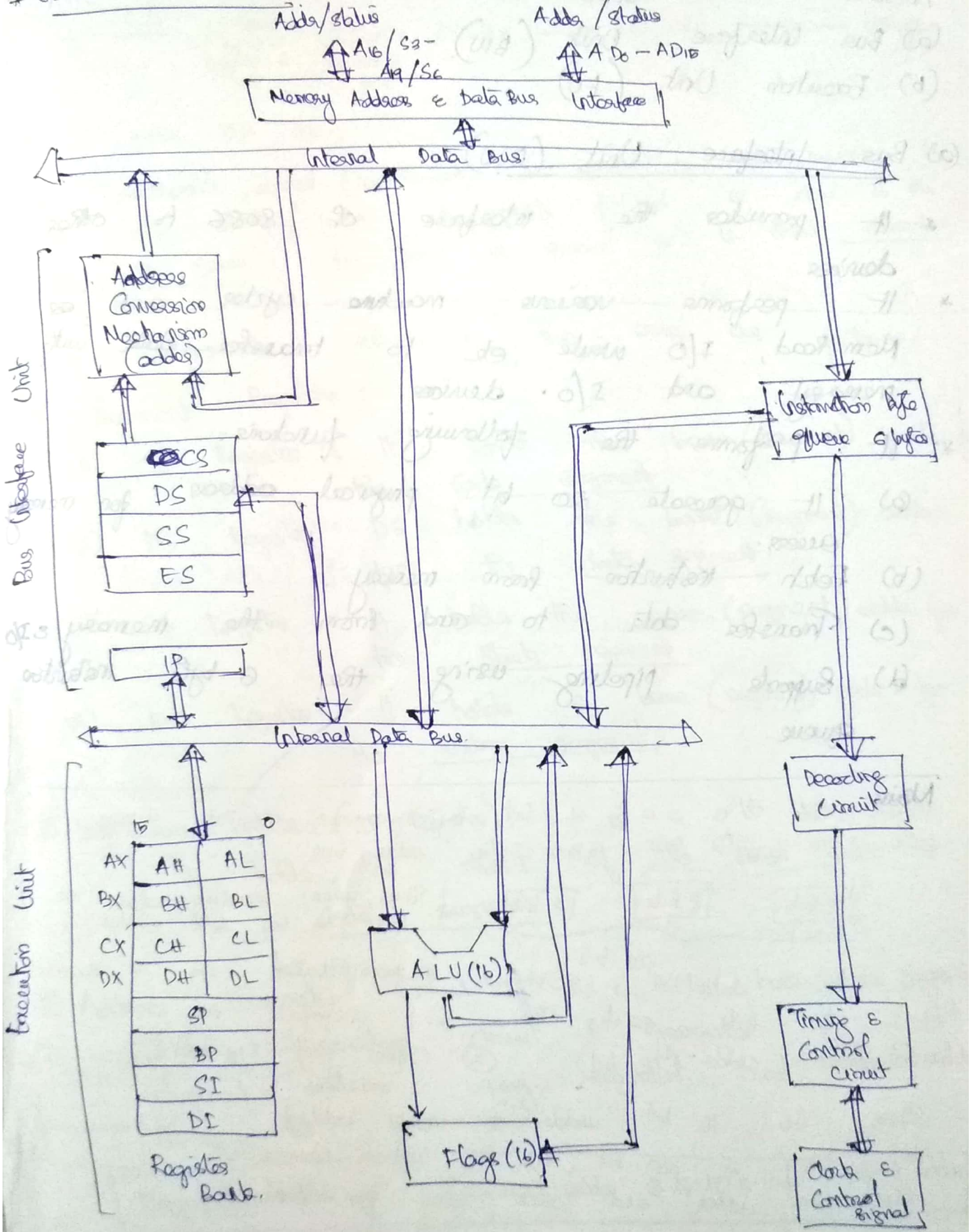
Dec	Binary (4 bit)	Heradecimal (16 bit)
0	0000	0000 0000 0000 0000
1	0001	0000 0000 0000 0001
2	0010	0000 0000 0000 0010
:		
9	1001	0000 0000 0000 1001
10 (A)	1010	0000 0000 0000 1010
:		
15 (F)	1111	0000 0000 0000 1111

∴ Hera decimal in 20 bit.



→ The address can vary from 00000H to FFFFFH.

* 8086 Architecture



Architecture of 8086 can be divided into 2 parts:

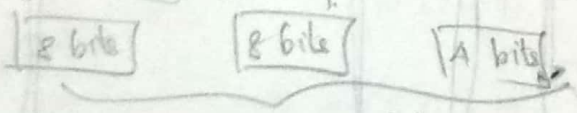
(a) Bus Interface Unit (BIU).

(b) Execution Unit (EU).

(a) Bus Interface Unit (BIU)

* It provides the interface of 8086 to other devices.
* It performs various machine cycles such as Mem Read, I/O write etc to transfer data with memory and I/O devices.
* It performs the following functions:
  (a) It generate 20 bit physical address for memory access.
  (b) Fetch instruction from memory.
  (c) Transfer data to and from the memory & I/O
  (d) Supports pipeling using the 6 byte instruction queue.

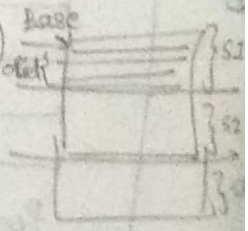
Note: 8086 is a 16 bit microprocessor, i.e, it process the data in 16 bits. But the addr bus is 20 bits



Thus, when sending data as 20 bits (4 bits is wasted)

Thus, the data is stored & manipulated as 2 segments of 16 bits each. The two segments are

① base addr (16 bit) ② Offset addr (16 bit)



These two 16 bit addresses are added to form a 20 bit (by the Address Generation Mechanism addr) and then send for fetching.

stores base addr
- CS — IP
- DS — SI
- SS — DI
- ES — BP, SP

} stores offset addr

Ultly, Data is stored temporily in AX, BX, CX, DX

AX ⟶ AH, AL ; BX ⟶ BH, BL ; CX ⟶ CH, CL

(AH: higher 8 bit, AL: lower 8 bit)

DX ⟶ DH, DL

The addresses added (16+16=20) is send to the memory to fetch the data / instr the insts fetched is send to the Instr Byte queue to form a queue & is then decoded

---

Main Components of the BIU are as follows:

(I) Segment Registers:

(1) CS Register: It holds the base (segment) addr for the Code segment.

(2) DS Register: DS holds the base (segment) addr for the data segment.

(3) SS Register: It holds the base (segment) addr for the stack segment.

(4) ES Register: It holds the base (segment) addr for the extra segment.

(II) Instruction Pointer (IP Register): It is a 16 bit register. It holds the offset of the next instr in the Code segment.

(III) Address Generation Circuit: The BIU has a physical address generation circuit. It generates the 20 bit physical address using segment and offset addr using the formula.

Physical addr = Segment addr * 10H + offset addr

The segment address is left shifted by 4 positions then multiply this number by 16 (i.e, 10H → 00010000 H) and then the offset addr is

added.

$\times 10H \Rightarrow$ multiplying 16 ~~shifting~~ 4 bits

as $0001 \xrightarrow{\text{shift 1 bit}} 0010 \Rightarrow 1 \to 2 \Rightarrow \times 2$

eg:- Segment addr : $1234H = (0001\ 0010\ 0011\ 0100)_b$

offset addr : $0005H = (0000\ 0000\ 0000\ 0101)_b$

$0001 \xrightarrow{\text{shift 2 bit}} 0100 \Rightarrow 1 \to 4 \Rightarrow \times 4$

$0001 \xrightarrow{\text{shift 3 bit}} 1000 \Rightarrow 1 \to 8 \Rightarrow \times 8$

Left shifting segment addr : $0001 \xrightarrow{\text{shift 4 bit}} 0001\ 0000 \Rightarrow 1 \to 16 \Rightarrow \times 16$

$0001\ 0010\ 0011\ 0100\ 0000$

Adding offset addr :

$0000\ 0000\ 0000\ 0101$

$0001\ 0010\ 0011\ 0100\ 0101$

$\Rightarrow [12345H]$.

(iv) 6 - Byte pre-fetch Queue :

It is a 6 byte - FIFO RAM used to implement pipeling. Fetching the new instruction while executing the current instruction is called pipelining.

BIU fetches the next "six-instruction bytes" from the code segment and stores it into the queue.

Execution unit (EU) removes instructions from the queue and executes them.

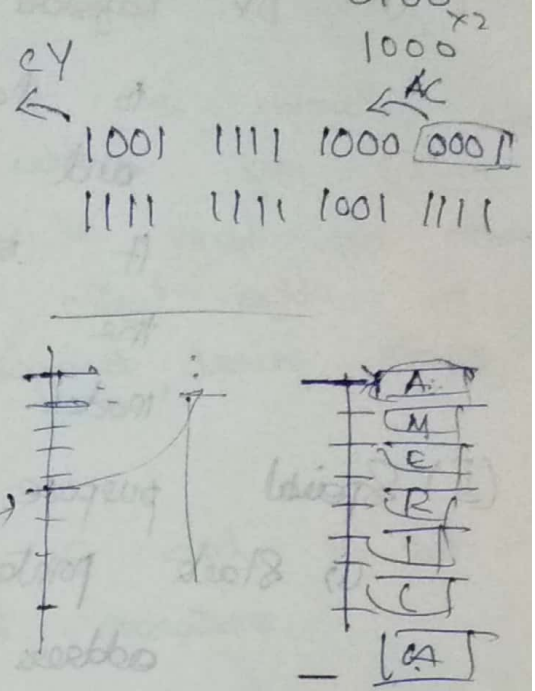
The queue is refilled when atleast two bytes are empty as 808 has a 16 bit data bus.

(b) Execution Unit

The main components of EU are as follows:

(I) General purpose Registers:

8086 has 4 16 bit-general purpose registers AX, BX, CX and DX. These are available to the programmer, for storing values during programming.



Each of these are divided into 8 bit registers such as AH, AL; BH, BL; CH, CL; DH, DL.

Each register apart from acting as GPR, has some specific functions as follows:

(i) AX Register (16 bits): It holds operands and results during multiplication & division operation. All I/O data transfers using IN and OUT instructions use AX register. It functions as accumulator during string operations.

(ii) BX Register (16 bits): Holds the memory address in indirect addressing modes.

(iii) CX Register (16 bits): Holds the count for instructions like: LOOP, Rotate, Shift and string operations.

(iv) DX Register (16 bits): It is used with AX to hold 32 bit values during multiplication and division.
It is used to hold the address of the I/O port in indirect I/O addressing mode.

(ii) Special purpose registers:

(i) Stack pointer (SP - 16 bits): It holds the offset address of the top of the stack.
Stack is a set of memory locations operating in LIFO manner. Stack is present in memory in stack segment.
SP is used with the SS register to calculate the physical addr for the stack segment.
It is used during instructions like PUSH, POP, CALL, RET etc.

During PUSH instr: SP is decremented by 2.
During POP instr: SP is incremented by 2.

(ii) Base Pointer (BP - 16 bits): BP can hold offset address of any location in the stack segment. It is used to access random locations of the stack

(iii) Source Index (SI - 16 bits): It is normally used to hold the offset address for data segment but can also be used for other segments. It holds

offset address of source data in Data segment during storing operations.

(iv) Destination Index (DI - 16 bits): It is normally used to hold the offset address for Extra segment, but can also be used for other segments. It holds the offset addr. of destination in Extra segment, during storing operations.

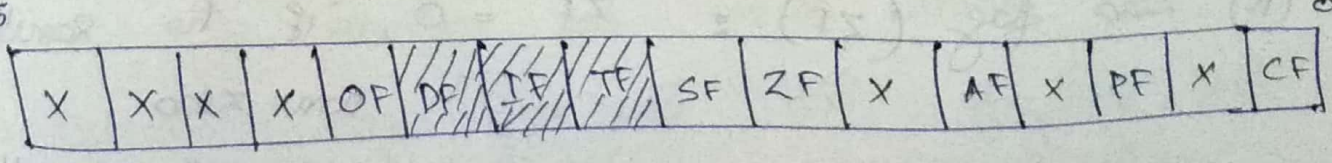
(iii) ALU (16 - bits): It performs 8 and 16 bit arithmetic and logical operations.

(ii) Instruction Decoder: The EU fetches an opcode from the queue into the instruction registers. The instruction decoder decodes it and sends the information to the control circuit for execution.

(v) Flag register (16 bits): It has 9 flags (though it has 16 bits). These flags are of two types: 6 status (condition) flags and 3 control flags.

Status flags are affected by many arithmetic and logical operations.

Control flags are used to control certain operations and are changed by the programmer.



Status flags :

(i) Carry flag (CF or CY) : It is set whenever there is a carry (or borrow) out of the MSB of a result.

i.e, D_7 bit for a 8 bit operation,
D_{15} bit for a 16 bit operation

eg:-

```
  1111  1111  1111  1010
  1000  0011  1111  1010
  1111  1111  1111  1111
 (1)1000  0011  1111  1001
```

stored in CF

(ii) Parity flag (PF) : It is set '1' if the result has even parity, and '0' for odd parity.

[Even parity : the result has even no. of 1's
odd parity : the result has odd no. of 1's]

(iii) Auxiliary Carry flag (AC) : It is set if a carry is generated out of the lower nibble. It is used in 8-bit operations like DAA and DAS.

eg:-

```
      (1)
  1111  1010
  1111  1111
 (1)1001
```

stored in AC

(iv) Zero flag (ZF) : ZF = 0, if the result is non-zero

ZF = 1, if the result is zero

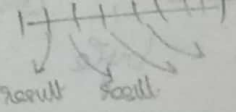
(v) Sign flag (SF): It is set '1', when the MSB of the result is 1, i.e, if the result is -ve.

∴ SF = 1, if result is -ve.
SF = 0, if result is +ve.

(vi) Overflow flag (OF): It will be set '1' if the result of a signed operation is too large to fit in the number of bits available to represent it.

Control flags:

(i) Trap flag (TP): It is used to set the trace mode, i.e, start single stepping mode. Here, μP is interrupted after every instn, so that the program can be debugged. i.e, after every instruction is executed, its result is displayed.



It is set by the programmer, if he wants the result of every instruction and not after the execution of the entire set of instns.

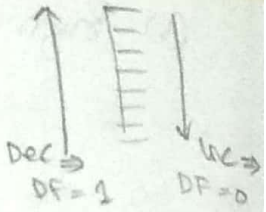
(ii) Interrupt Enable flag (IF): It is used to mask (disable) or unmask (enable) the INTR interrupt. The programmer sets the IF if wants the μP to be interrupted during its execution and execute another set of instns.

(iii) Direction flag (DF): If this flag is set SI and DI are in auto-decrementing mode in string operations.
i.e, this flag sets the direction of flow of execution

of the instructions.

when DF = 0, it is in incrementing mode, i.e, instrs are executed from top to bottom incrementing the addr.

when DF = 1, it is in decrementing mode, i.e, instrs are executed from bottom to top decrementing the addr.



* Physical Memory Organization

Memory Banking:

The physical memory of 8086 is divided into two banks: even / lower bank; odd / upper bank.

→ As 8086 has a 16 bit data bus, it should be able to access 16-bit data in one cycle

→ To do so, it needs to read from 2 memory locations, as one memory location carries only one byte, 16 bit data is stored in two consecutive locations.

→ However, if both of the memory locations are in same chip they cannot be accessed at the same time.

→ Hence, the memory of 8086 is divided into two banks, each provides 8 bits.

→ One bank contains all even addresses called the "Even Bank", while the other is called "odd Bank" containing all odd addresses.

→ Generally for any 16-bit operation, the even bank provides the lower byte and the odd bank provides the higher byte. Hence the

Even bank is also called the lower bank and the odd bank is also called the higher bank.

This is divided into 2 → 512 KB of each.

Address bits = 20 bits (∵ total memory → 2^{20} = 1 MB)

1 MB

512 KB

512 KB

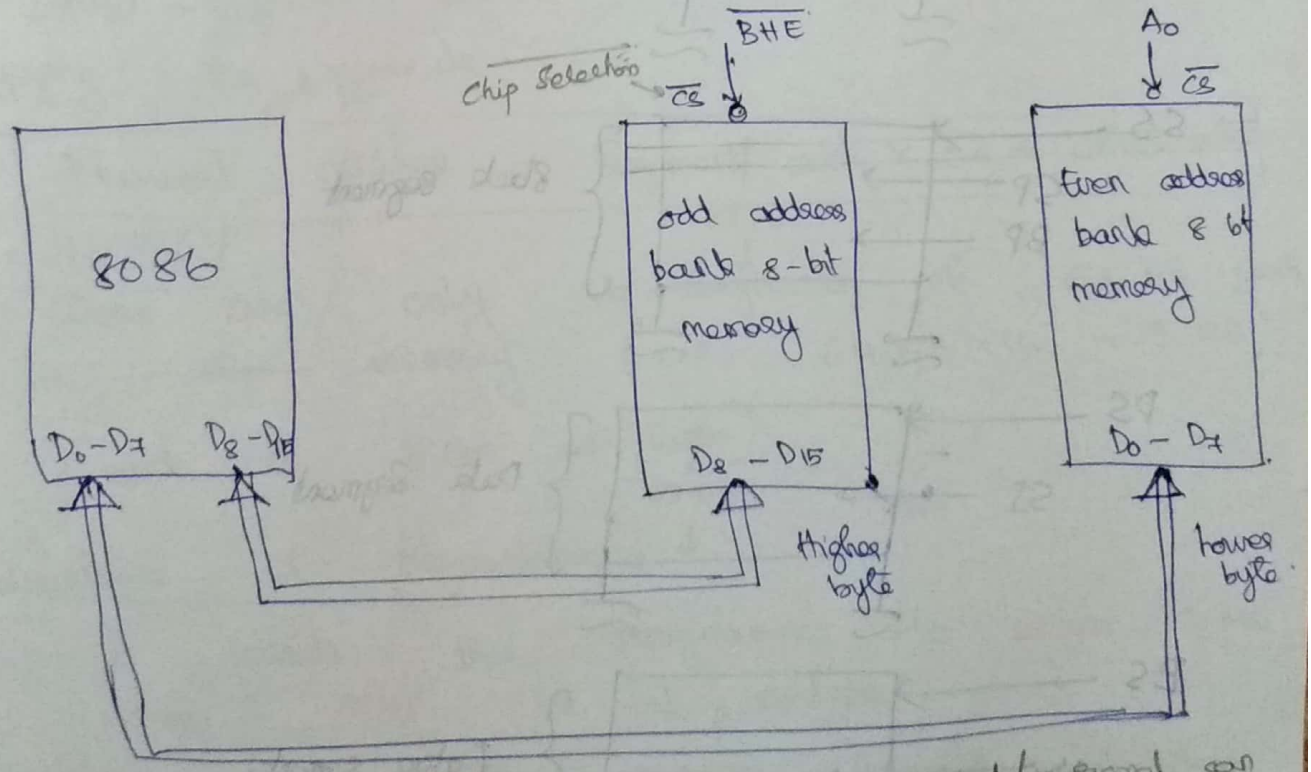
→ Odd Bank
→ (Higher Bank)
→ Address Range:
00001H (8 bits)
00003H "
00005H "
⋮
FFFFFH.

→ Selected when $\overline{BHE}$ = 0.

→ Even bank
→ (lower bank)
→ Address Range:
00000H (8 bits)
00002H "
00004H "
⋮
0000EH

→ selected when A_0 = 0.

// The data bus = 16 bits, i.e, thus, odd & even banks are selected alternatively. ($D_0 - D_{15}$)



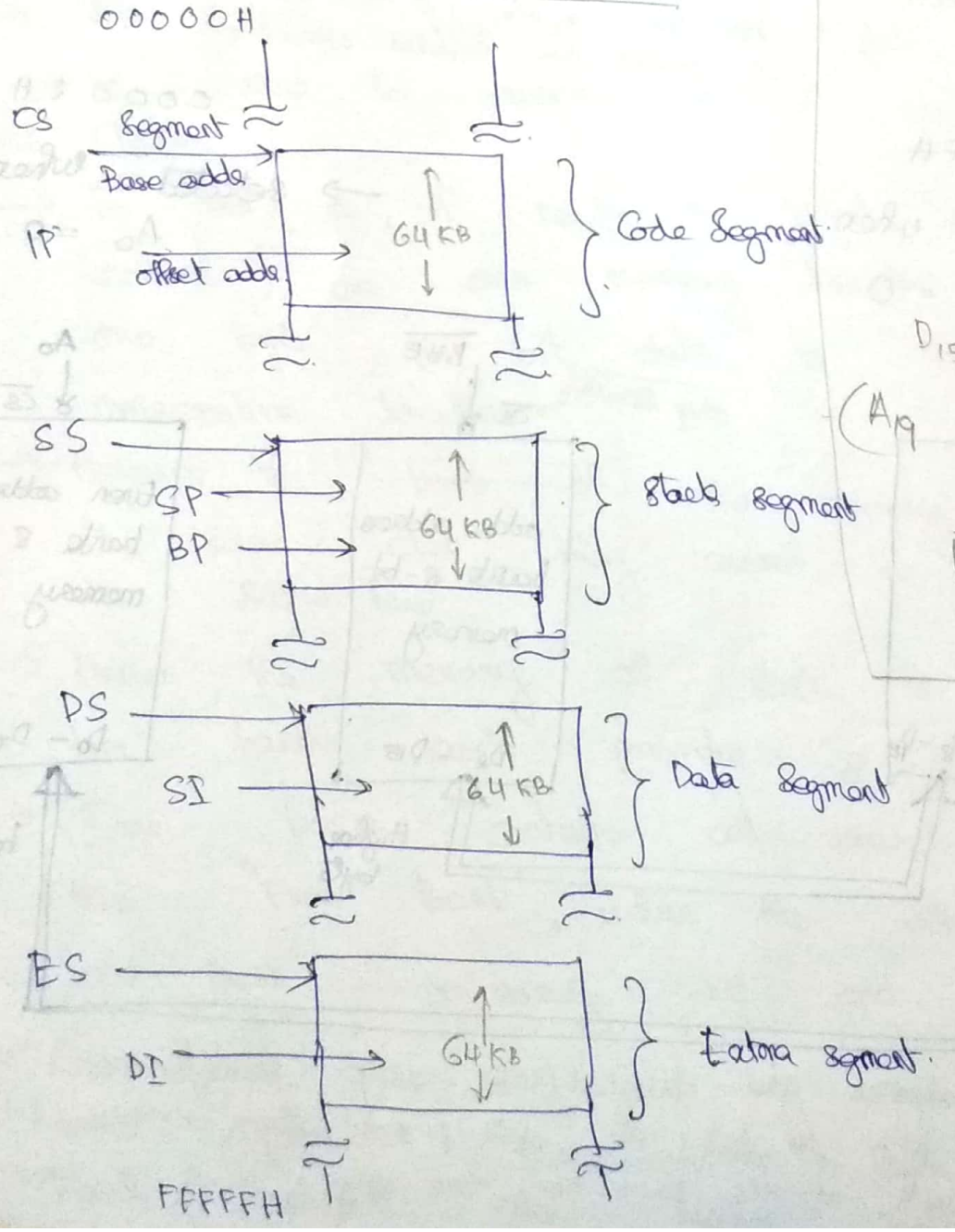
Since the banks are aligned a single read/write signal can be used to fetch 16 bits of data (8 bit from even & 8 bit from odd banks at the same time). Thus, only 1 signal is required.

$\overline{BHE}$	A_0		
0	0	→	R/W 16 bit from both bank
0	1	→	R/W 8 bit from higher bank
1	0	→	R/W 8 bit from lower bank
1	1	→	R/W No operation.

In address bus, $A_0 - A_{19}$, $(A_1 - A_{19})$ is given to the memory & A_0 is used for chip selection ($\overline{CS}$) and Bus high Enable ($\overline{BHE}$) is generated by 8086.

* Logical Memory Organization

Memory Segmentation in 8086:



Addr = 20

2^{20} = 1MB

512KB 512KB

$D_{15} - D_8$ $D_7 - D_0$

$(A_{19}$ $A_1 A_0)$

64KB × 16 = 1MB

→ Segmentation means dividing the memory into logically different parts called segments.

→ 8086 has a 20 bit address bus, hence it can access 2^{20} bytes i.e, 1 MB memory.

→ But this also means that physical address will now be 20 bit. ~~The~~ is not possible to work with a 20 bit address as it is not a byte compatible number.

→ To avoid working with this incompatible number, we will create a virtual model of the memory which is called memory segmentation.

→ Here, the memory is divided into 4 segments Code, Stack, Data & Extra.

→ Each segment has a 16 bit base address & 16 bit offset address.

→ The physical address is calculated by the microprocessor using the formula:

Physical address = Segment addr × 10H + offset addr

→ There are only 16 segments of 64 KB each in the memory (i.e, 64 KB × 16 = 1 MB).
(each of these 4 kinds).

Advantages of Segmentation:

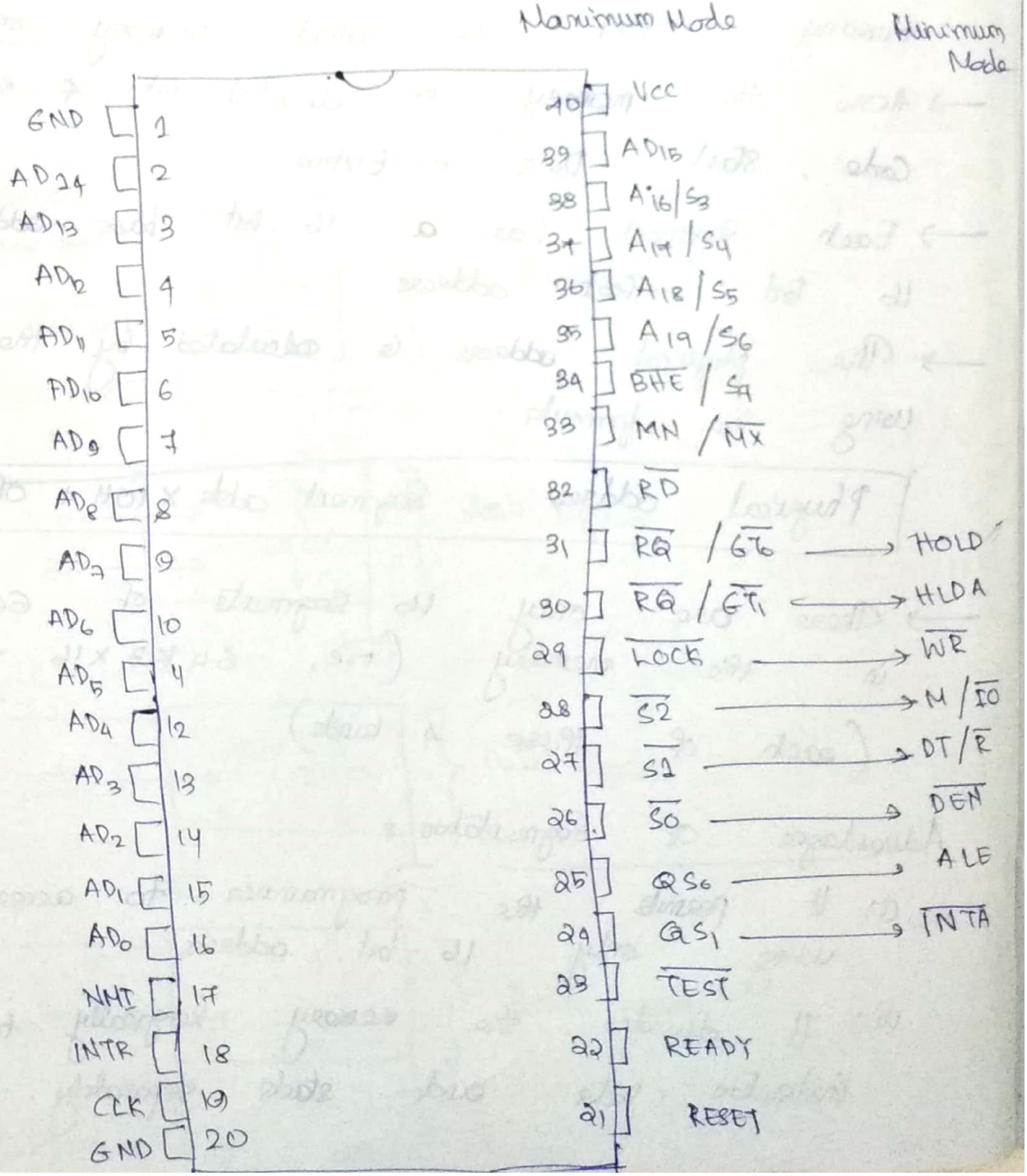
(i) It permits the programmer to access 1 MB using only 16-bit address.

(ii) It divides the memory logically to store instruction, data and stack separately.

Disadvantages of Segmentation:

(i) Although the total memory is 16 × 64 KB, at a time only 4 × 64 KB memory can be accessed.
(i.e, only 1 out of a kind of segment (CS, DS, SS, ES can be accessed at a time).

* 8086 Pin Diagram



Pin Description:

(i) $AD_{15} - AD_0$: These are time multiplexed memory I/O address and data lines. Address remains on the lines during T_1 state, while the data is available on the data bus during T_2, T_3, T_w and T_4. These, T_1, T_2, T_3, T_4 and T_w are the clock states of a machine cycle. T_w is a wait state.

multiplexed to carry both address & data through the same bus.

(ii) A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, A_{16}/S_3 : These are time multiplexed address and status lines. During T_1 these are the most significant address lines for memory operation. During I/O operation, these lines are low.

as address lines are not multiplexed, they are used with status lines

During memory or I/O operations, the status signals are available on this lines for T_2, T_3, T_w and T_4.

S_5 - status of interrupt enable flag bit.
(IF flag ⇒ set 1 → accepts interrupt
set 0 → does not accept int)

S_4	S_3	Indication
0	0	Alternate Data (Extra Segment)
0	1	Stack (Stack segment)
1	0	Code or none (Code or None Segment)
1	1	Data (Data Segment)

combination of S_4 & S_3, status is which segment being used.

S_6 - always low.

(iii) $\overline{BHE}$ / S_7 (Bus High Enable / status) : $\overline{BHE}$ is used to select the odd address memory bank or peripherals. $\overline{BHE}$ is low during T_1 for read, write and interrupt acknowledge cycles. Whenever a byte is written on the higher byte

S_7 has no specific use.

of the data bus.

S_4 status is not currently used.

$\overline{BHE}$	A_0	Indication
0	0	whole word
0	1	upper byte from to odd address
1	0	lower byte from or to even address.
1	1	None

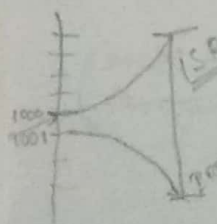
(iii) $\overline{RD}$ (Read) : Read signal when low, indicates the peripheral that the processor is performing a memory or I/O read operation. $\overline{RD}$ is active low and shows the state for T_2, T_3, T_W of ~~read~~ ~~write~~ cycle.

RD = 0 → reading
RD = 1 → no reading

(iv) Ready: This is an acknowledgement from the I/O devices or memory that they have completed the data transfer.

shows if the I/O devices are ready or not.

(v) INTR (Interrupt Request) : This is a level triggered input. This is sampled during the last clock cycle of each instruction to determine the availability of the request.



The IF flag can control INTR; IF IF = 0 → no interrupt is accepted
IF = 1 → interrupt is accepted.

(vi) $\overline{TEST}$: This input is examined by a WAIT instruction. If the $\overline{TEST}$ input goes low, execution will continue, else the processor remains in an idle state.

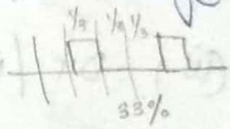
the MP checks $\overline{TEST}$ before working
If TEST = 0 → continues
TEST = 1 → waits
it depends on the keyword in the WAIT program

(viii) NMI (Non-Maskable Interrupt) : This is an edge-triggered input which causes a Type 2 interrupt. The NMI is not maskable internally by software. (This interrupt cannot be controlled by IF flag)

(ix) RESET : This input causes the processor to terminate the current activity and start execution from <u>FFFF0H</u>.

The signal should be high for atleast four clock cycles.

(x) CLK Input : The clock input provides the basic timing for processor operation and bus control activity. It is an asymmetric square wave with 33% duty cycle. The range of freq. is 5 MHz to 10 MHz.



(xi) Vcc : +5V power supply for the operation of the internal circuit.

(xii) GND : Ground for internal circuit.

(xiii) MN / $\overline{MX}$: This pin decides whether the processor is to operate either in minimum or maximum mode.

operates the μP in min or max mode
if input = 1 → MIN
= 0 → MAX

• <u>Pins in Minimum Mode</u>

(i) M / $\overline{IO}$ (Memory / $\overline{IO}$) : This is a status line. $\overline{S2}$ in the maximum mode. When it is low, it indicates that CPU is having an I/O operation and when it is high, it indicates that the CPU is

shows if the μP is doing a memory or IO operation
if the pin = 1 → memory
= 0 → IO operation

having a memory operation.

(ii) $\overline{INTA}$ (Interrupt Acknowledge – Input) : This is an active low pin. Whenever it goes low, it means that the processor has accepted the interrupt. It is active low during T_2, T_3 & T_W of each interrupt acknowledge cycle.

when a device request INTE the MP if accepts gives the signal INTA if; INTA = 1 → not acknowledged INTA = 0 → acknowledged

(iii) ALE (Address Latch Enable – Output) : This signal indicates the availability of the valid address on the address / data lines and is connected to latch enable input of latches.

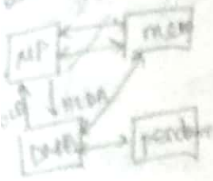
shows if data address or data is currently given through AD_0–AD_7 pins if ALE = 1 ⇒ address transferred = 0 ⇒ data is transferred

(iv) DT/$\overline{R}$ (Output) : This output indicates the direction of the data flow through the transreceivers. When the processor sends out data, the signal is high and when the processor is receiving data, this signal is low. This is equivalent to $\overline{S1}$, in maximum mode.

Data transmit/Receive shows if data is received or transmitted by MP. MP pin = 1 → µP transmits data to mem. pin = 0 → µP receives data from mem.

(v) $\overline{DEN}$ (Data Enable) : This signal indicates the availability of valid data over the addr / data lines. It is used to enable the transreceivers to separate the data from the multiplexed address / data signal.

(vi) HOLD, HLDA (Hold / Hold Acknowledge) : When 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.

Pins in Maximum Mode

(i) $\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$ (Status lines) : These are status lines which indicate the type of operations carried out by the processor.

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Indication
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	HALT
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive.

(ii) $\overline{Lock}$: This output pin indicates that other system bus masters will be prevented

from gaining the system bus, while the $\overline{LOCK}$ signal is low.

The $\overline{LOCK}$ signal is activated by the LOCK prefix instruction and remains active until the completion of the next instruction.

(iii) QS_1, QS_0 (Queue status):

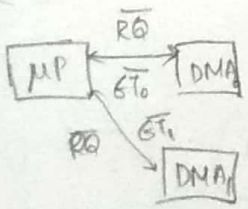
shows the status of the instruction queue (contains 6 bytes)

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

(iv) Operation for Queue

$\overline{RQ}/\overline{GT_0}$, $\overline{RQ}/\overline{GT_1}$: They are used by other local bus masters, in max mode to force the processor to release the local bus at the end of the processor current bus cycle.

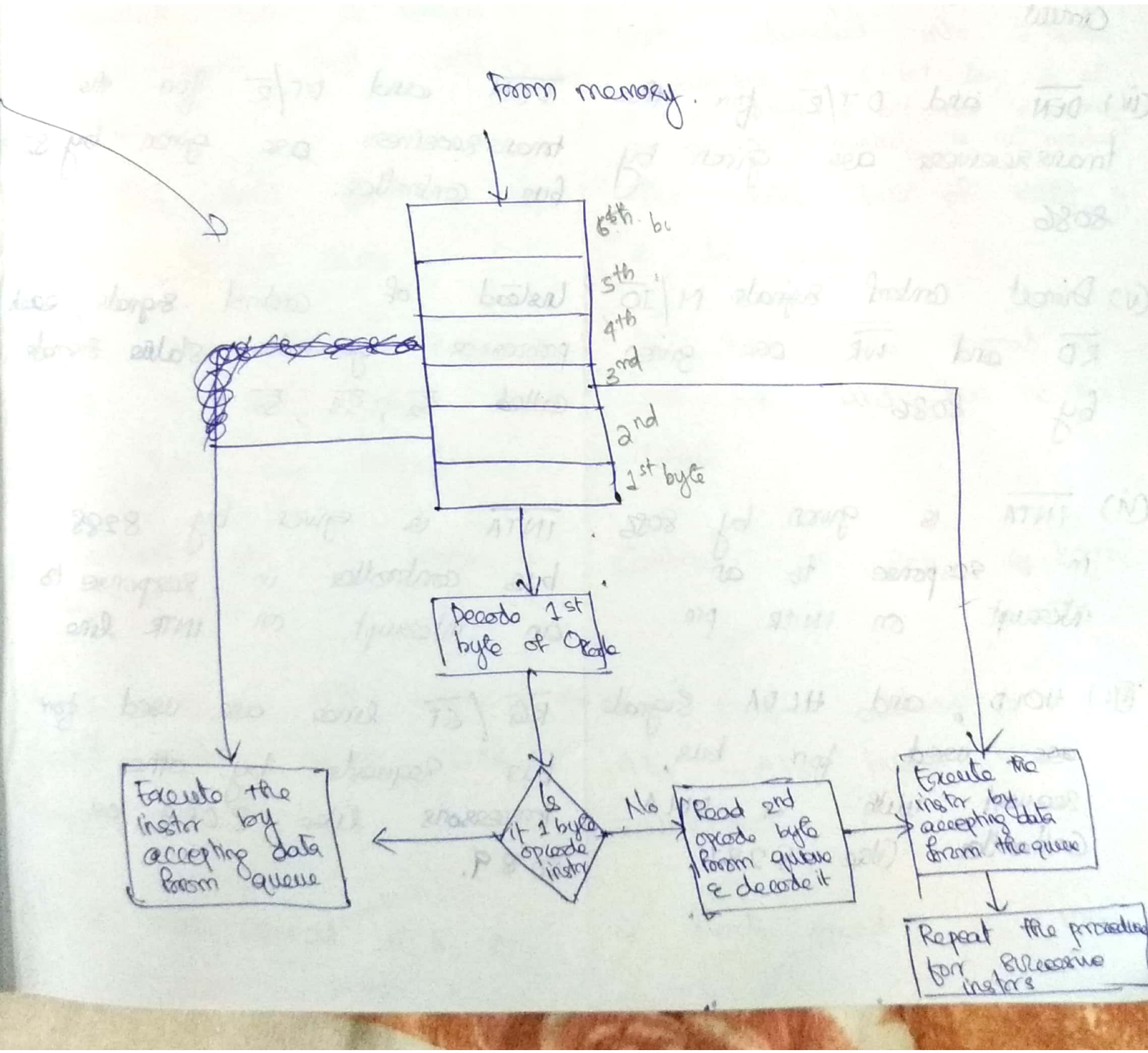
(like to HOLD & HLDA)



$\overline{RQ}/\overline{GT_0}$ has higher priority than $\overline{RQ}/\overline{GT_1}$

When DMA needs the bus, it gives a low signal thus $\overline{RQ}$ = 1 ⇒ a request is send to MP

When MP acknowledges the signal, it gives a low signal, i.e, $\overline{GT_0}$ = 1.



* Minimum and Maximum Mode Operation (refer fig)

Minimum Mode	Maximum Mode
(i) In min mode, there can be only 1 processor e.; i.e, 8086	In max. mode, there can be multiple processors with 8086 like 8087, 8089, etc.
(ii) MN/$\overline{MX}$ is high to set 8086 into min mode	MN/$\overline{MX}$ is low.
(iii) ALE for the latch is given by 8086 as it is the only processor in the circuit.	ALE for the latch is given by 8288 bus controller as there can be multiple processors
(iv) $\overline{DEN}$ and DT/$\overline{R}$ for the transreceivers are given by 8086	$\overline{DEN}$ and DT/$\overline{R}$ for the transreceivers are given by 8288 bus controller.
(v) Direct control signals M/$\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ are given by 8086	Instead of control signals, each processor generates status signals called $\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$
(vi) $\overline{INTA}$ is given by 8086 in response to an interrupt on INTR pin	$\overline{INTA}$ is given by 8288 bus controller in response to an interrupt on INTR line.
(vii) HOLD, and HLDA signals are used for bus request with a DMA Controller like 8237.	$\overline{RQ}$/$\overline{GT}$ lines are used for bus requests by other processors like 8087 or 8089.

(viii) The circuit is simples. The circuit is more complex.

(ix) Multiprocessing cannot be performed hence performance is lower. As multiprocessing can be performed it can give very high performance.

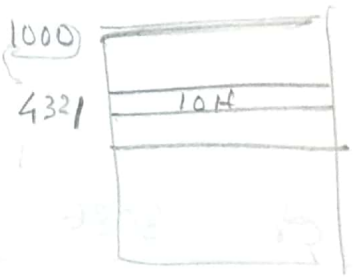
* Comparison of 8086 and 8088

	8086	8088 (to make it compatible to 8085)
(i)	The instruction queue is 6 byte long	The instr queue is 4 byte long.
(ii)	The physical memory is divided into 2 banks even/lower and odd/higher banks.	The memory in 8088 is not divided into 2 banks. (data bus is 8-bit long; so at a time only 8-bit can be accessed so memory banking is not needed)
(iii)	The data bus of 8086 is 16 bit wide.	The data bus of 8088 is 8-bit wide.
(iv)	It has $\overline{BHE}$ signal on pin no: 34 and there is no $\overline{SS_0}$ signal.	It has $\overline{SS_0}$ signal on pin 34 and has no $\overline{BHE}$ signal.
(v)	Control pin in 8086 is M/$\overline{IO}$	Control pin 8088 is IO/$\overline{M}$.
(vi)	In 8086, all address and data buses are multiplexed.	In 8088, address bus $AD_7 - AD_0$ buses are multiplexed.
(vii)	3 clock speed: 5, 8, 10 MHz	2 clock speed: 5, 8 MHz

(viii) Can read/write 8 bit or 16 bit data at a time

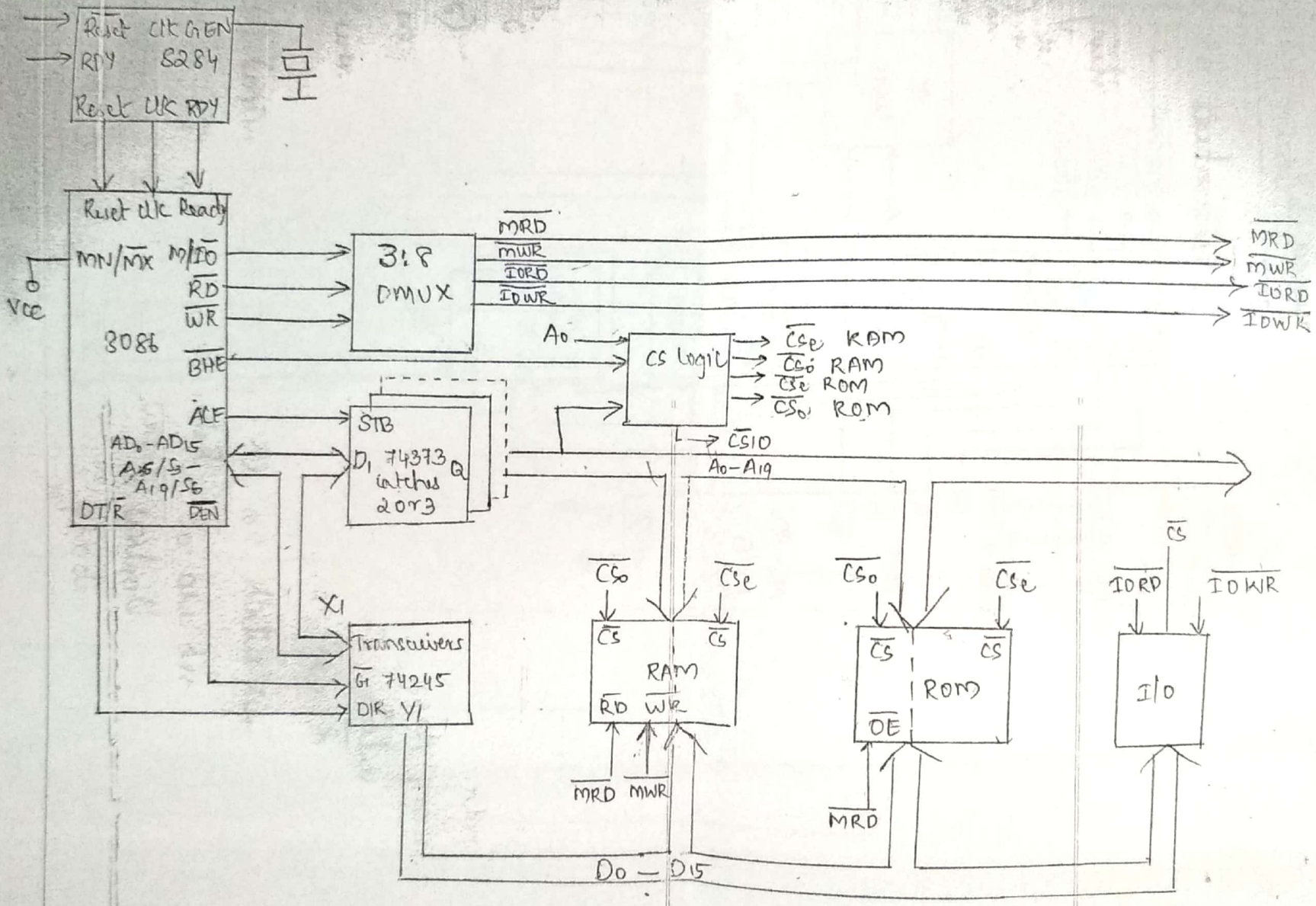
Can read/write 8 bit data at a time.

000 – 0
001 – 1
010 – 2
011 – 3



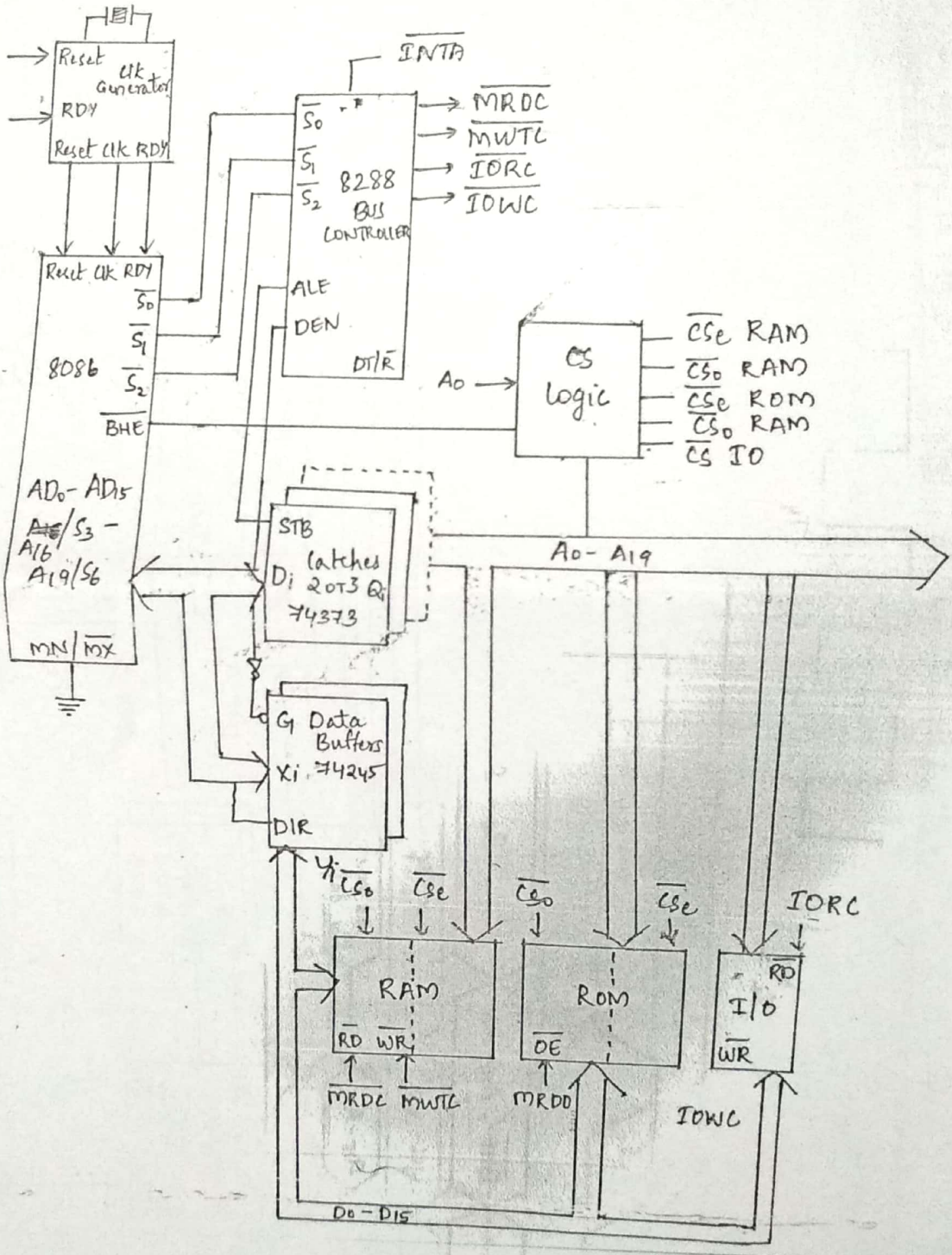
BX = 4321

[BX]

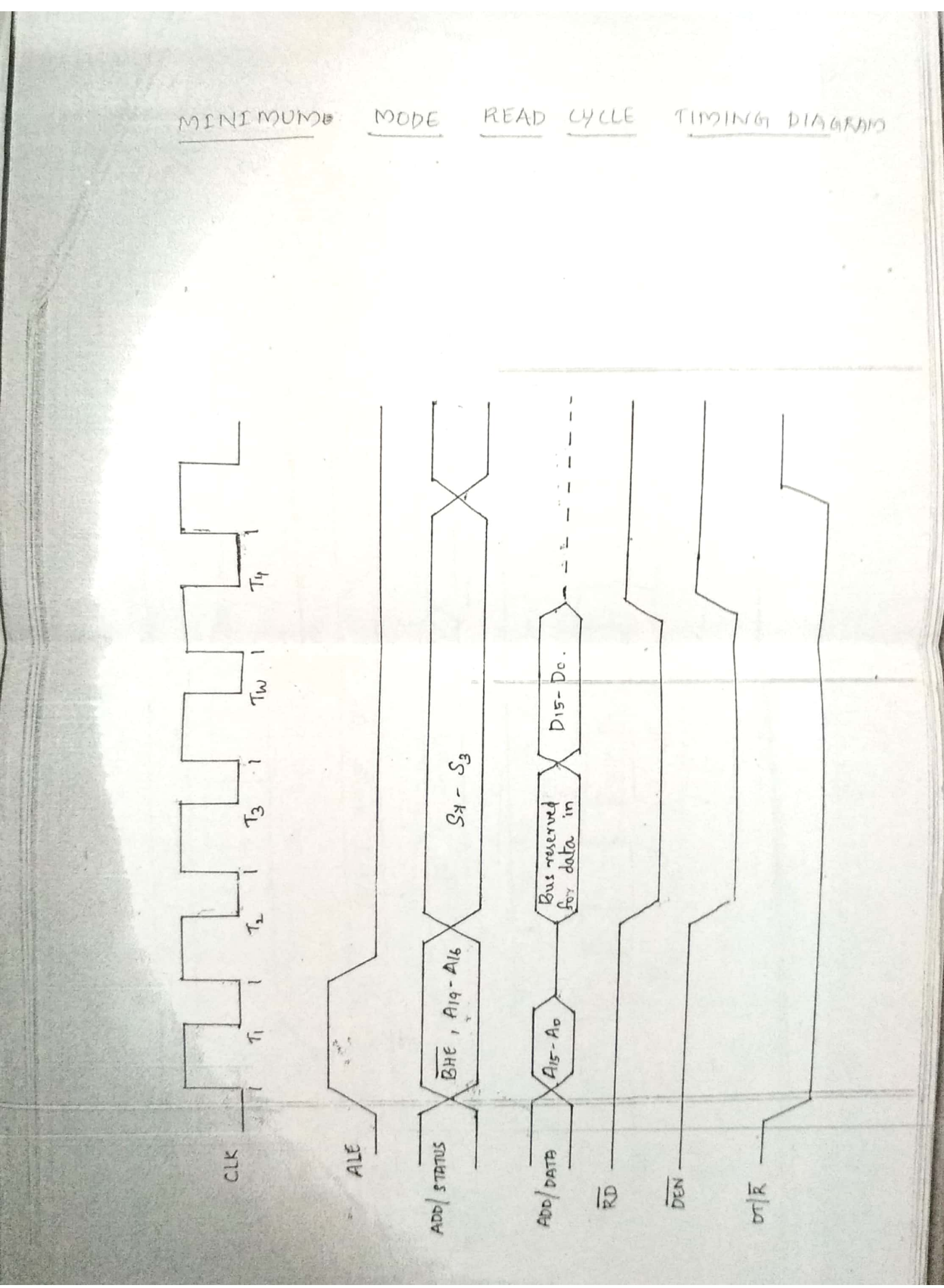


MINIMUM MODE 8086

MAXIMUM MODE 8086

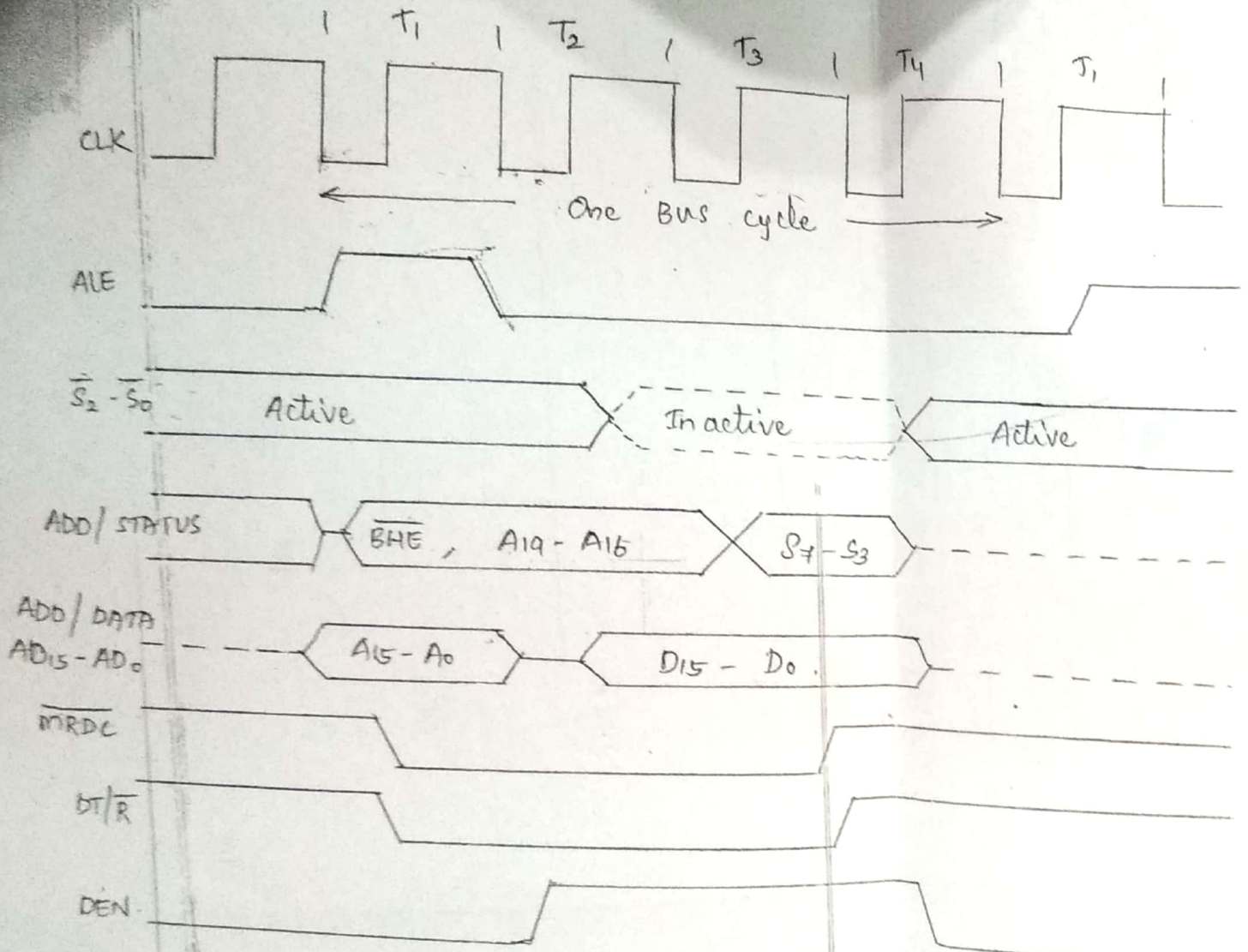


MINIMUM MODE READ CYCLE TIMING DIAGRAM



MINIMUM MODE WRITE CYCLE TIMING DIAGRAM

MEMORY READ TIMING IN MAXIMUM MODE



MEMORY WRITE TIMING IN MAXIMUM MODE