UNIT - II

Microprogram sequencer:

central processing unit.

The 8086 processor.

Register organisation of 8086

8086 General purpose register or special purpose Registers

→ 8086 has a powerful set of registers known as general purpose registers.

→ All are 16 bit registers, they can also be used as 8-bit registers

→ they are used for holding data, variables and temporary storage of data.

Four type of registers

1). General data register.
2). segment Registers.
3). pointer and index registers
4) flag registers

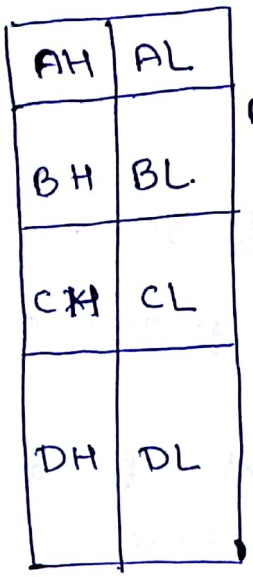
1. General data Registers.

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

'X' shows that it is a 16 bit

AX is used as 16 bit accumulator.

divided into two parts
AL is low 8 bit
AH is high 8 bit.



BX is used as an offset storage for forming physical address

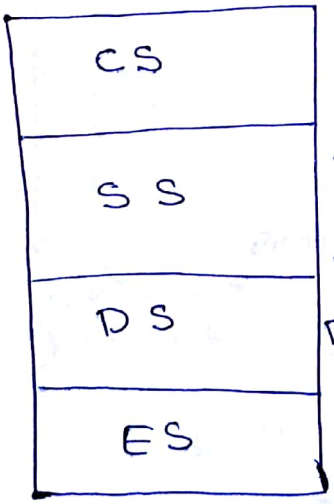
CX is used as a default counter in case of ~~loo~~ string and loop instruction

DX is used as implicit operand or destination instruction

Segment Registers

It has 1mega byte memory.

Each segment contains 64 Kbyte memory



code segment registers used for addressing memory location

stack segment register used for addressing stack segment of memory.

Data segment Register used for dage segment of memory

Extra segment Register is also used for data segment of memory

If we want to take address from memory location we have to calculate the physical address. The physical address is calculated from two parts

① segment address

②. offset address (BX - general purpose Reg)

Pointer and index Register

The pointer ~~and~~ contains offset within the particular segment

SP
BP
SI
DI
IP

The IP, BP, SP contain offsets

SI, DI are index ~~Based~~ Registers

SI is generally used to store the offset of source data.

DI is used to store the offset of destination in data or extra segment.

Flag registers

8086 has 16 bit flag registers

They are two types

1). condition code ~~coorde~~ or status flag.

2). machine control flag.

The condition code flag register is the lower byte of the 16 bit flag register along with the overflow flag. The machine control flag is the higher byte of the flag register. It contains three flag. direction flag (D), interrupt flag (I) and trap flag (T)

flag Register of 8086

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	O	D	I	T	S	Z	X	Ac	X	P	X	CY

S → sign flag

flag is set when the result of any computation is negative. It is the MSB bit

Z → zero flag.

If previous instruction is zero this flag is set.

P → parity flag.

This flag is set to 1 if the lower byte of the result contains even number of 1's

CY → carry flag →

If carry is produced in MSB this flag is set.

T → Trap flag :-

If this flag is set, the processor enters the single step execution mode.

I → Interrupt flag –

If maskable interrupts are recognised by the CPU this flag is set

D → Direction flag –

used by string manipulation instruction If this flag bit is '0', the string is processed beginning from the lowest address to the highest address i.e auto incrementing mode otherwise, the string is processed from the highest address towards the lowest address i.e auto decrementing mode.

AC → Auxiliary carry flag →

This is set if there is a carry from the lowest nibble i.e bit three.

O - overflow flag →

This flag is set if an overflow occurs.

ARCHITECTURE

It supports 16 bit ALU and also provide segment memory capability, rich instruction set, powerful interrupts, execution etc.
It divide ~~in two~~ into two parts

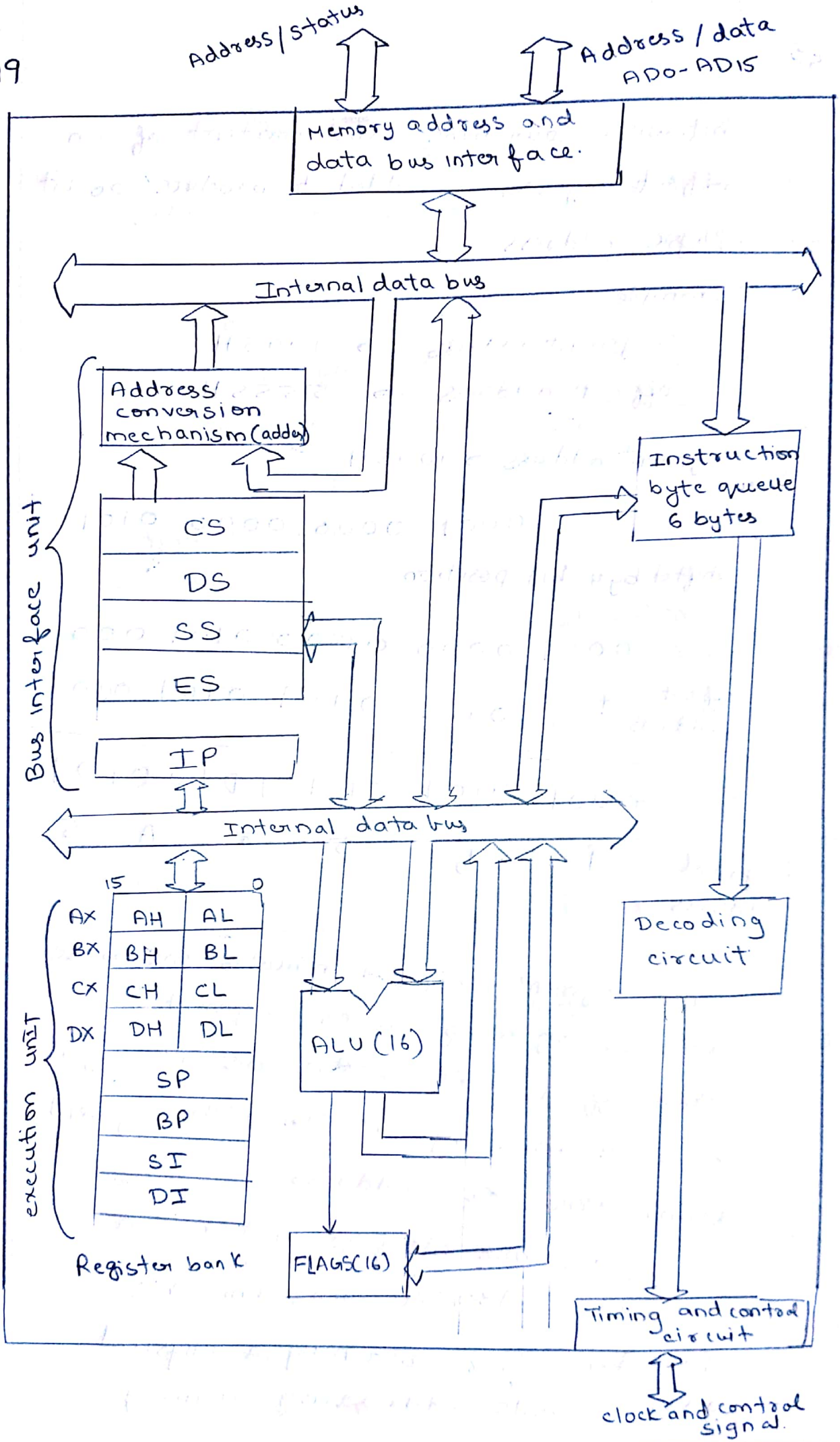
Bus interfacing unit
execution unit

The bus interfacing unit contains the circuit for physical address calculation and a predecoding instruction byte queue (6 bytes long)
The bus interface unit makes the system bus signal available for external interfacing of the device.

~~The BIU along with execution unit [EU]~~ the bus interface unit manages the complete interface of execution unit with memory and I/O device under the control of the timing and control unit.

The complete physical address which is 20 bits long is generated using segment and offset registers, each 16 bit long.

For generating a physical address from contents of these two registers the content of a segment register also called as segment address is shifted left



bit wise four time and content of an offset register is added to produce 20 bit Physic address

example

segment address → 1005H

Offset address → 5555H.

Segment address → 1005H

0001 0000 0000 0101

Shifted by 4 bit position

0001 0000 0000 0101 0000

Offset address + 0101 0101 0101 0101

0001 0101 0101 1010 0101

Physical address 1 5 5 A 5

The segment register indicates the base address of a particular segment The offset indicates the distance of the required memory location in the segment from the base address

The bus interface unit has a separate adder to perform the procedure for obtaining a physical address while addressing memory

The segment address value is taken from an appropriate segment register depending whether code, data or stack are to be accessed.

OFFset is from IP, BX, SI, DI, SP, BP depending on the addressing mode.

In 8086 for utilization of bus (time slot) we have overlapped fetch and execution cycles.

while the fetched instruction is executed internally, the external bus is used to fetch the machine code of the next instruction and arrange it in a queue know as Predecoded instruction byte queue. It is 6 bytes long, first-in first-out structure. The instruction from the queue are taken for decoding sequentially

while the opcode is fetched by the Bus interface unit (BIU), the Execution unit (EU) executes the previously decoded instruction concurrently.

The execution unit contain register set except segment register and IP. 16 bit ALU to perform arithmetic and logic operation 16 bit flag registers to reflect the result of execution by the ALU.

The decoding unit decodes the opcode 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.

The execution unit may pass the results to the bus interface unit for storing them in memory

PHYSICAL MEMORY ORGANISATION

In an 8086 based system, the 1Mbytes memory is physically organized as an odd bank and an even bank. each of 512 K bytes

Byte ~~address~~ data with an even address is transferred on $D_7 - D_0$

Byte data with an odd address is transferred on $D_{15} - D_8$ bus lines.

The processor provides two enables signals

$\overline{BHE}$ and A_0 for selection of either even or odd or both the banks

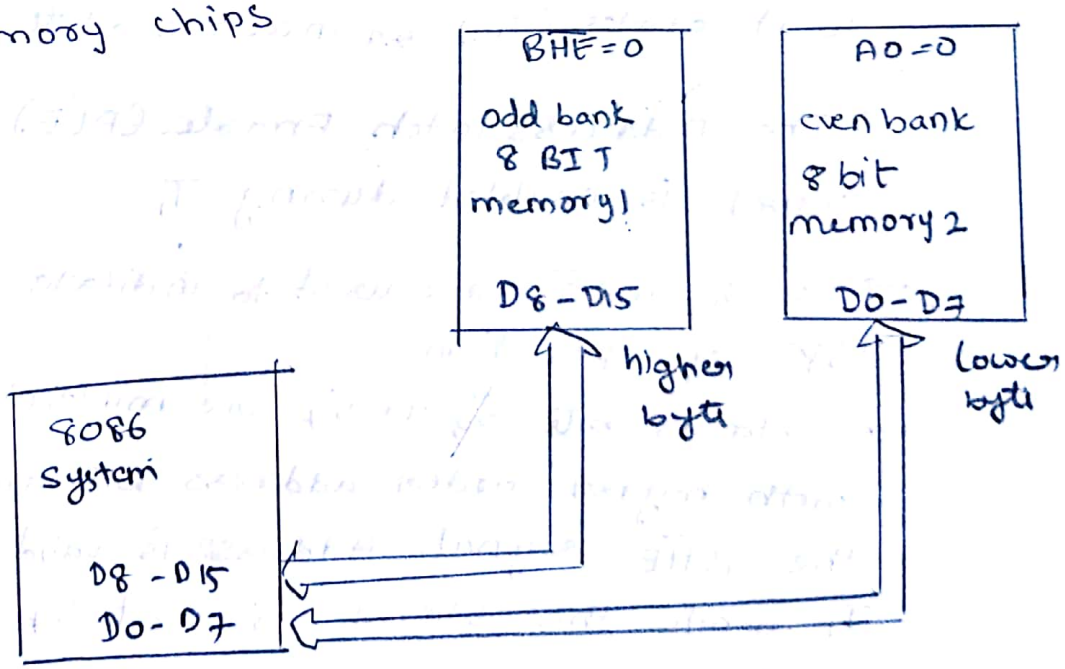
If the processor fetches a word (2 byte) from memory, there are different posibilities

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.

A0 and $\overline{BHE}$ → selection of memory bank.

BHE	A0	Indication
0	0	2 byte word.
0	1	upper byte odd address
1	0	lower byte even address
1	1	none.

A map of an 8086 memory system starts at 00000H and ends at FFFFFH. 8086 being 16-bit processor is expected to access to and from 8-bit commercially available memory chips



General bus operation

→ Basically, all the processor bus cycle consist of four clock cycle.

→ These are referred to as T_1, T_2, T_3 and T_4

→ Tw → wait state or Ti (Idle state) or inactive state

→ The address is transmitted by the processor during T_1. It is present on the bus only for one cycle.

→ During T_2 the bus is tristated for changing the direction of the bus for the data read cycle.

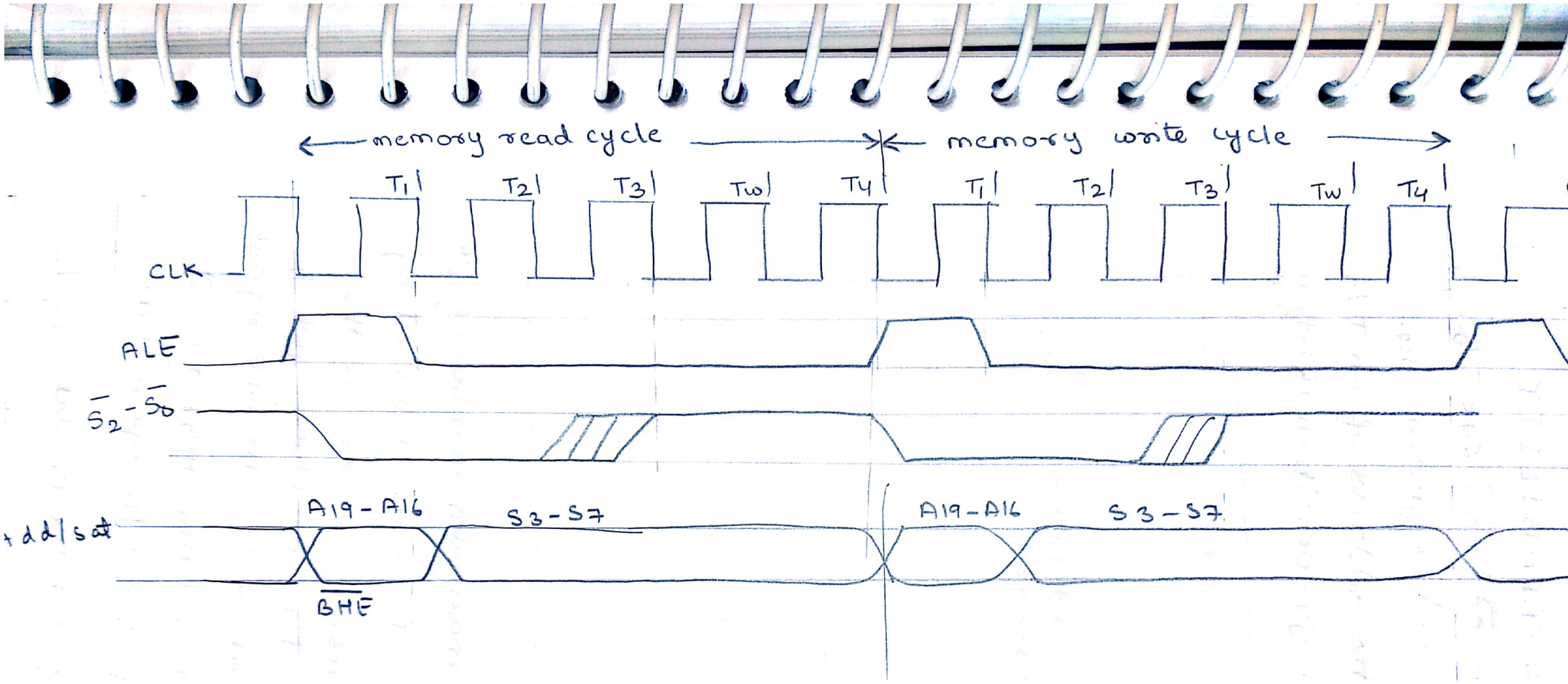
→ Data Transfer takes place during T_3 and T_4

→ In case, an addressed device is slow A WAIT state Tw are inserted between T_3 and T_4. These clock state during wait period are called idle state (Ti) wait states (Tw) or inactive state

→ The Address latch Enable (ALE) signal is enabled during T_1

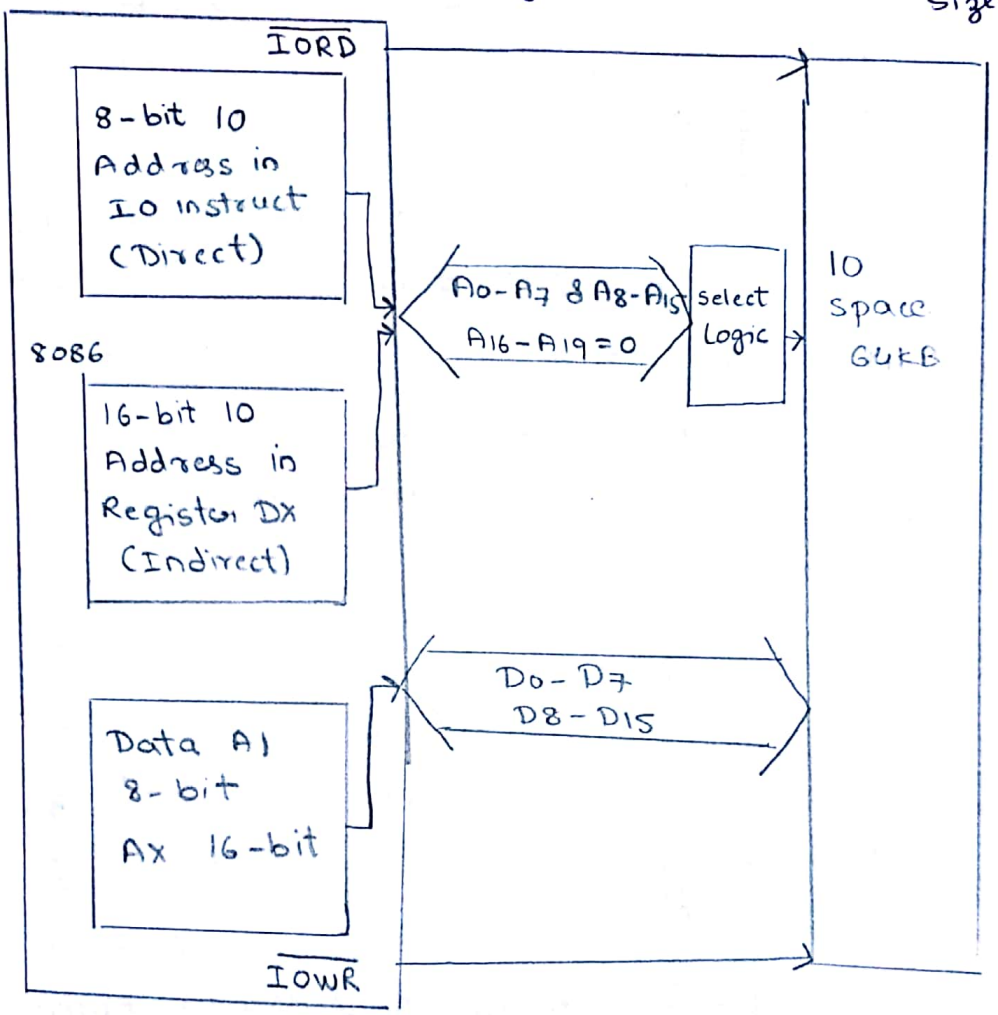
→ $\overline{S0}$, $\overline{S_1}$, and $\overline{S_2}$ are used to indicate the type of operation

→ Status bits $\overline{S_3}$ to $\overline{S_7}$ are multiplexed with higher order address bits and the $\overline{BHE}$ signal. Address is valid during T_1 while the status bits S_3 to S_7 are valid during T_2 through T_4.



I/O ADDRESSING CAPABILITY

8086 processor can address 64K I/O byte registers or 32K word register. the limitation is that the address of an I/O device must not be greater than 16 bits in size



this means that a maximum number of 2^{16} i.e 64K byte I/O device may be accessed by the CPU. The I/O address appears on the address line A0 to A15 for one clock cycle T_1 it may then be latched using the ALE signal The upper address lines (A16-A19) are at logic 0 level during the I/O operation.

The 16-bit register Dx is used as 16-bit I/O address pointer. with full

capability to address upto 64K devices.

Even addressed bytes are transferred on $D_7 - D_0$ and odd addressed bytes are transferred on $D_8 - D_{15}$ lines.

SPECIAL PROCESSOR ACTIVITIES

MINIMUM MODE 8086 SYSTEM AND TIMINGS

→ 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 group

address / databus
status
control
Interrupt and DMA

→ In a minimum mode 8086 system the microprocessor 8086 is operated in minimum mode by strapping its MN/$\overline{MX}$ (pin33) 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
Demultiplexer
memory
I/O devices

→ Latches are generally buffered output D-type flipflop like 74LS373 or 8282. There are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signals generated by 8086.

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

→ They are controlled by two signals namely, $\overline{DEN}$ and $DT/\overline{R}$

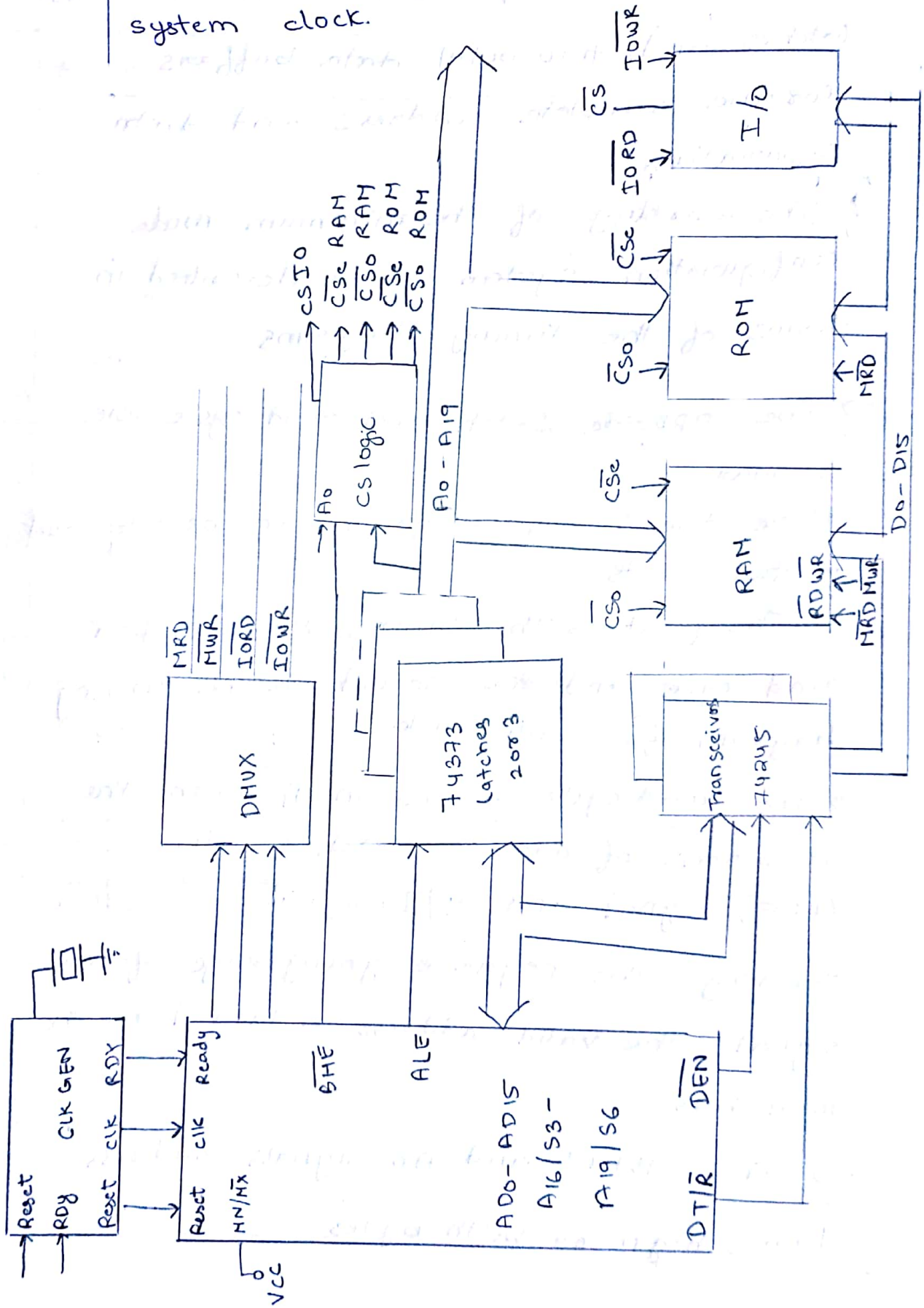
→ The $\overline{DEN}$ signal indicates the valid data is available on the data bus. while $DT/\overline{R}$ indicates the direction of data .ie from or to the processor.

→ The system contains memory for the monitor and user program storage. usually EPROM are used for monitor storage, while RAM for users program storage.

→ A system contains I/O devices.

→ The clock generator generates the clock from the crystal oscillator.

which is used as timing reference. It also synchronizes some external signal with the system clock.



→ It has 20 address lines and 16 data lines the 8086 CPU requires three octal address latches and two octal data buffers for the complete address and data separation

→ The working of the minimum mode configuration system can be described in terms of the timing diagrams

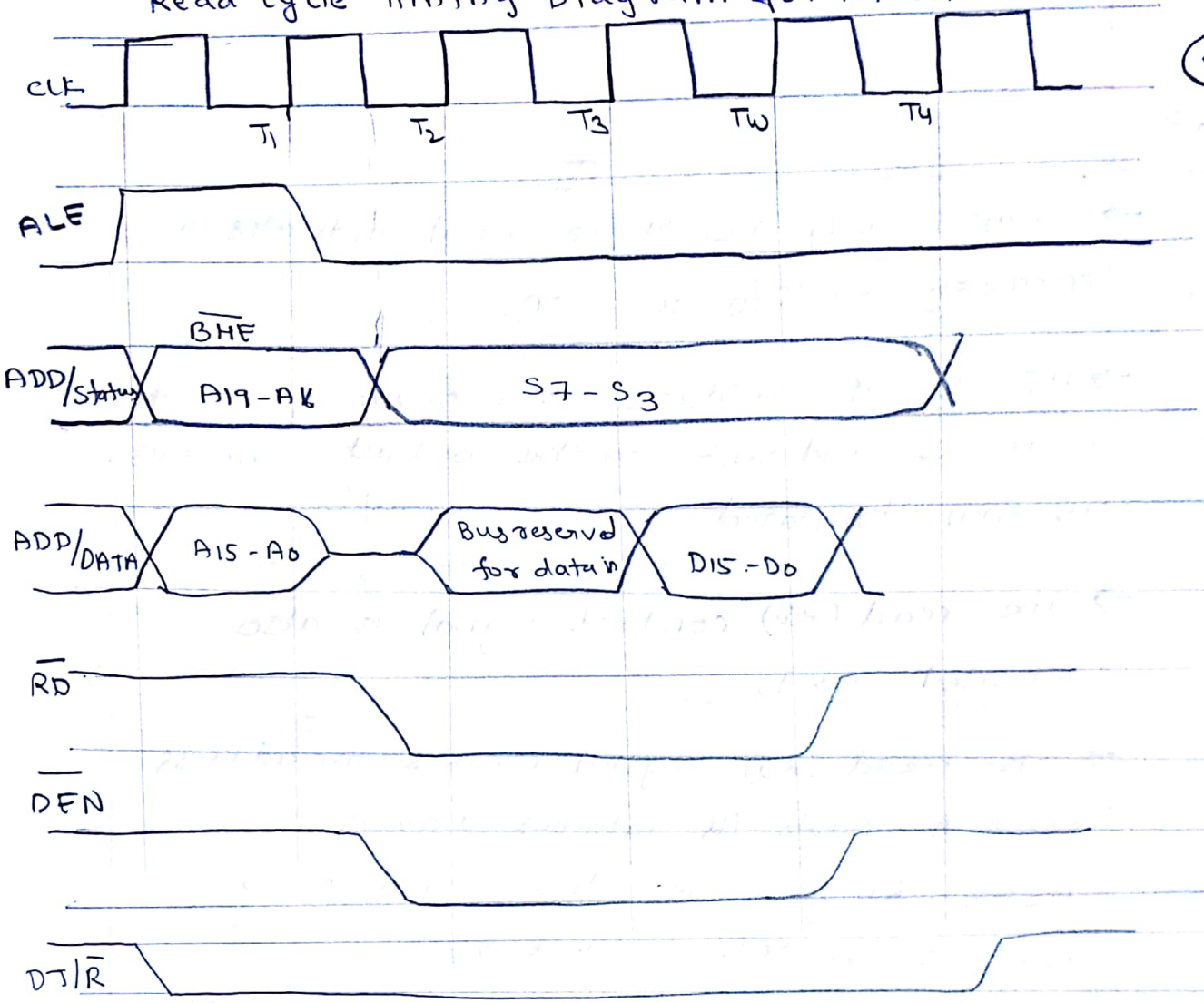
→ The opcode fetch and read cycle are similar.
Hence the timing diagram can be categorized in two parts.

The first is the timing diagram for read cycle and the second is the timing diagram for write cycle.

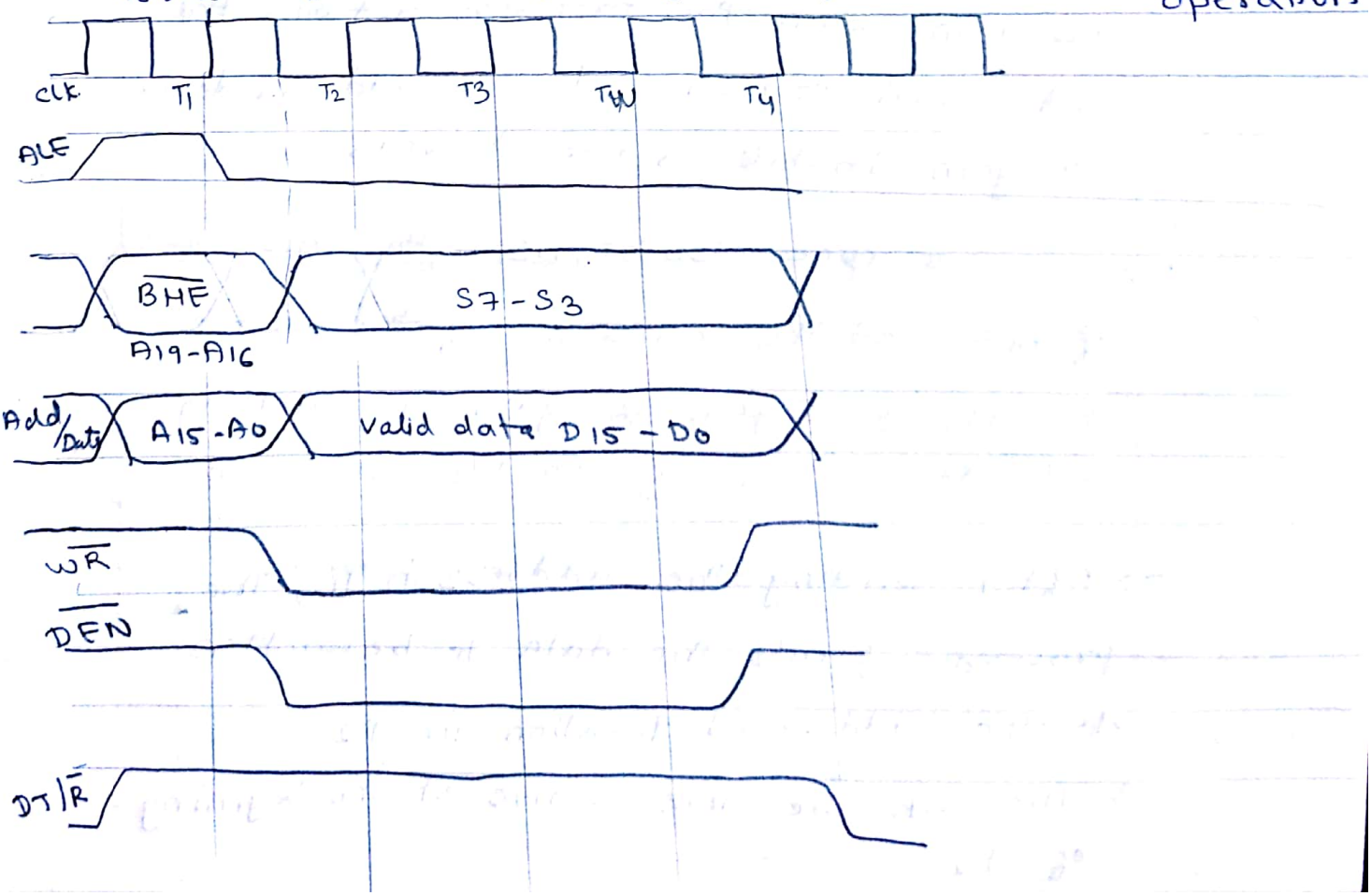
→ The read cycle begins in T_1 with the assertion of address latch enable (ALE) signal and M/IO signal.
During the negative going edge of this signal, the valid address is latched on the local bus

→ The BHE and A0 signals address low, high or both bytes

Read cycle Timing Diagram for minimum mode.



write cycle timing Diagram for minimum mode operation



→ From T_1 to T_4 the M/IO signal indicates a memory or I/O operation

→ AT T_2, the address is removed from the local bus and sent to the output. The bus is then tristated

→ The read (RD) control signal is also activated as T_2.

→ 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 address 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

→ After sending the address in T_1, The processor sends the data to be written to the addressed location in T_2.

→ The WR becomes active at the begining of T_2

→ 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.

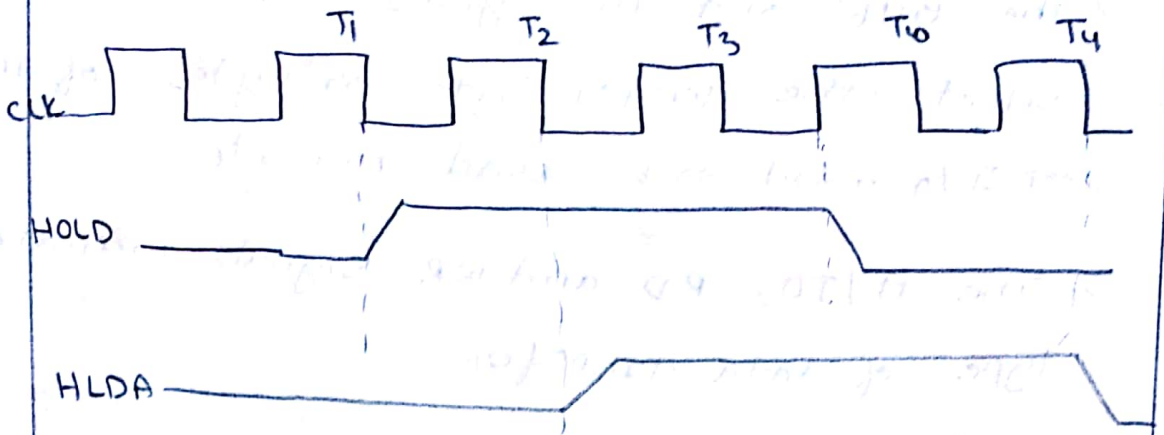
$M/\overline{IO}$	$\overline{RD}$	$\overline{WR}$	Transfer type
0	0	1	I/O read
0	1	0	I/O write
1	0	1	memory read
1	1	0	memory write

CS logic indicates chip select logic and 'e' and O sufixes indicates even and odd address memory banks

The $\overline{BHE}$ and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or written

$\overline{BHE}$	A0	Indication
0	0	whole word
0	1	upper byte from or to odd address
1	0	lower byte from or to even address
1	1	None.

Hold Response sequence.



Bus Request and Bus Grant Timings in minimum Mode system

The HOLD pin is checked at the end of each bus cycle.

CPU activates HLDA for the succeeding ~~by~~ bus cycles.

~~The HOLD pin~~

If it is received active by the processor before T_4 of the previous cycle or during T_1 state of the current cycle, the CPU activates HLDA in the next clock cycle.

and for the 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 and timings

When the 8086 is set for the maximum-mode configuration. it provides signals for implementing a multiprocessor / co-processor system environment

By multiprocessor environment we mean that more than one microprocessor exists in the system and that each processor is executing its own program

In this type of system environment, there are some system resources that are common to all processor. They are called as global resources. There are also other resources that are assigned to specific processor. These are known as local or private resources

co-processor also means that there is a second processor in the system.

→ 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 signal.

→ $\overline{S0}$, $\overline{S1}$, $\overline{S2}$ 3 bit bus code identifies which type of bus cycle is to be followed.

S2, S1, S0 are input to the external bus controller device. The bus controller generates the appropriately timed command and control

signal.

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CPU cycles	8288 command
0	0	0	Interrupt Acknowledge	$\overline{INTA}$
0	0	1	Read I/O port	$\overline{IORC}$
0	1	0	Write I/O port	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	HALT	None.
1	0	0	Instruction Fetch. code access	~~$\overline{MRDC}$~~ code access
1	0	1	Read memory	MRDC
1	1	0	Write memory	$\overline{AMWTC}$
1	1	1	Passive.	None

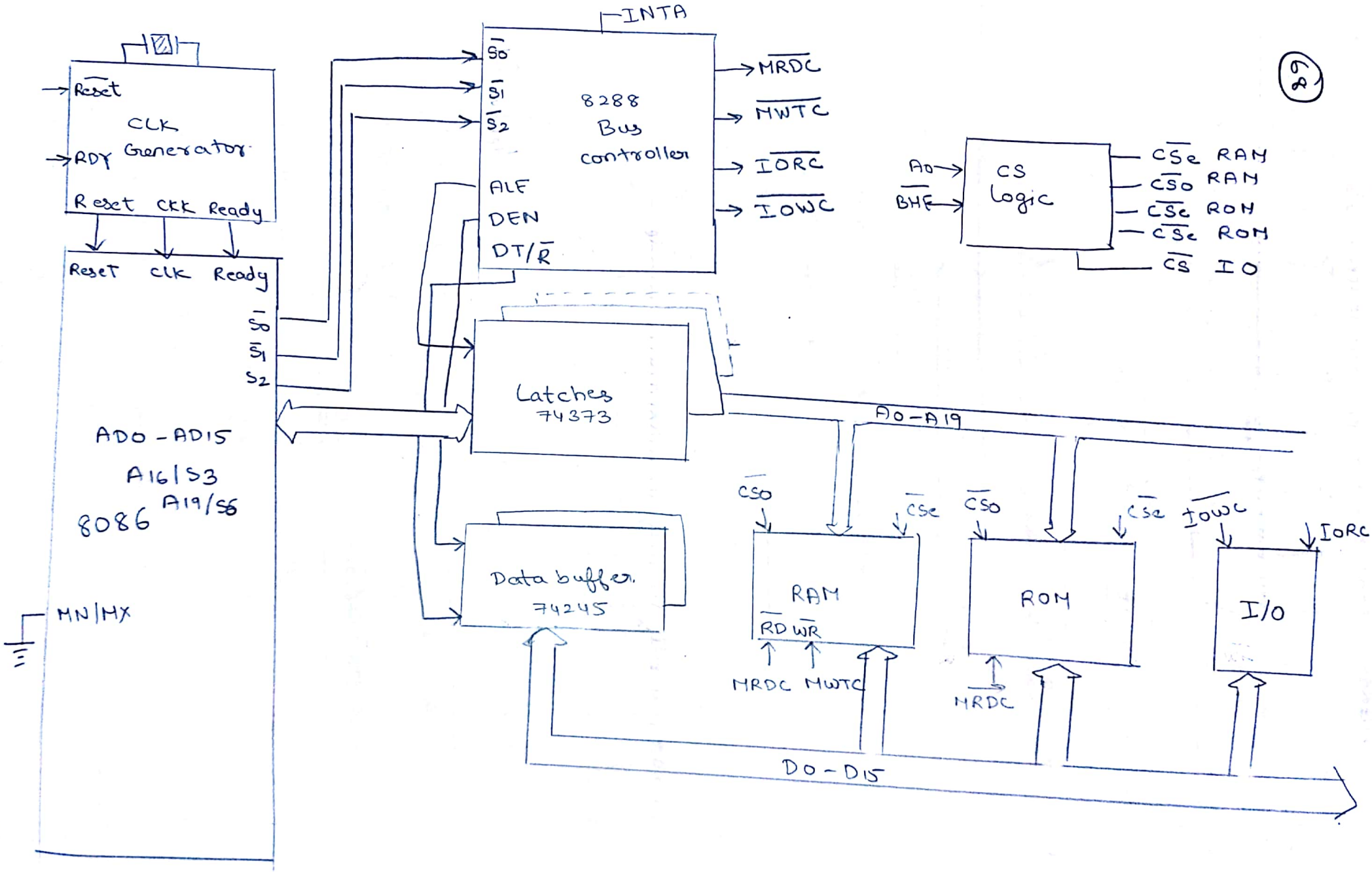
→ The 8288 produces one of two command signals for each bus cycle.

→ The control output produced by the 8288 are DEN, DT/R and ALE

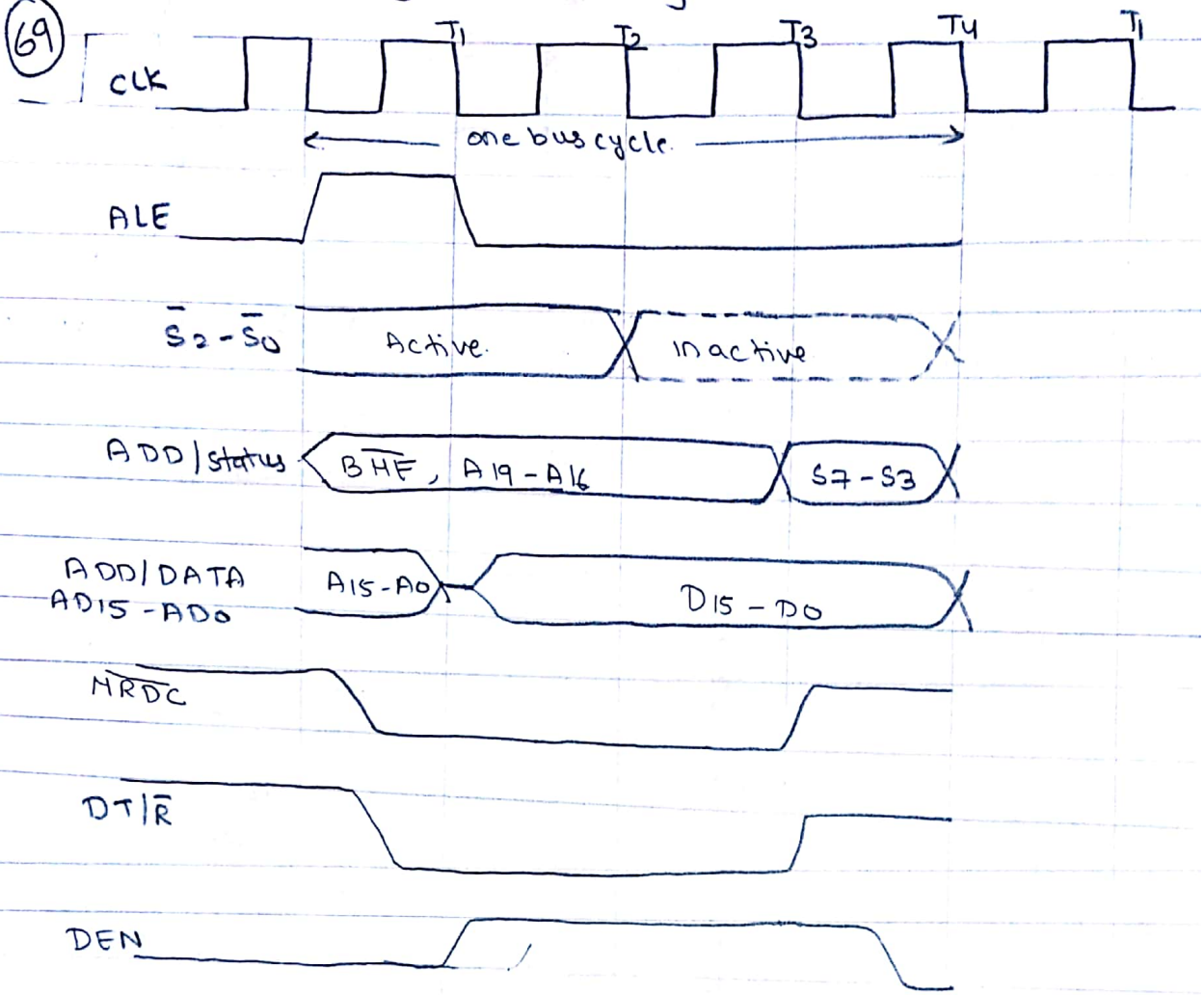
These three signals provide the same function as those described for the minimum system mode.

QS_1, QS_0 - Queue status

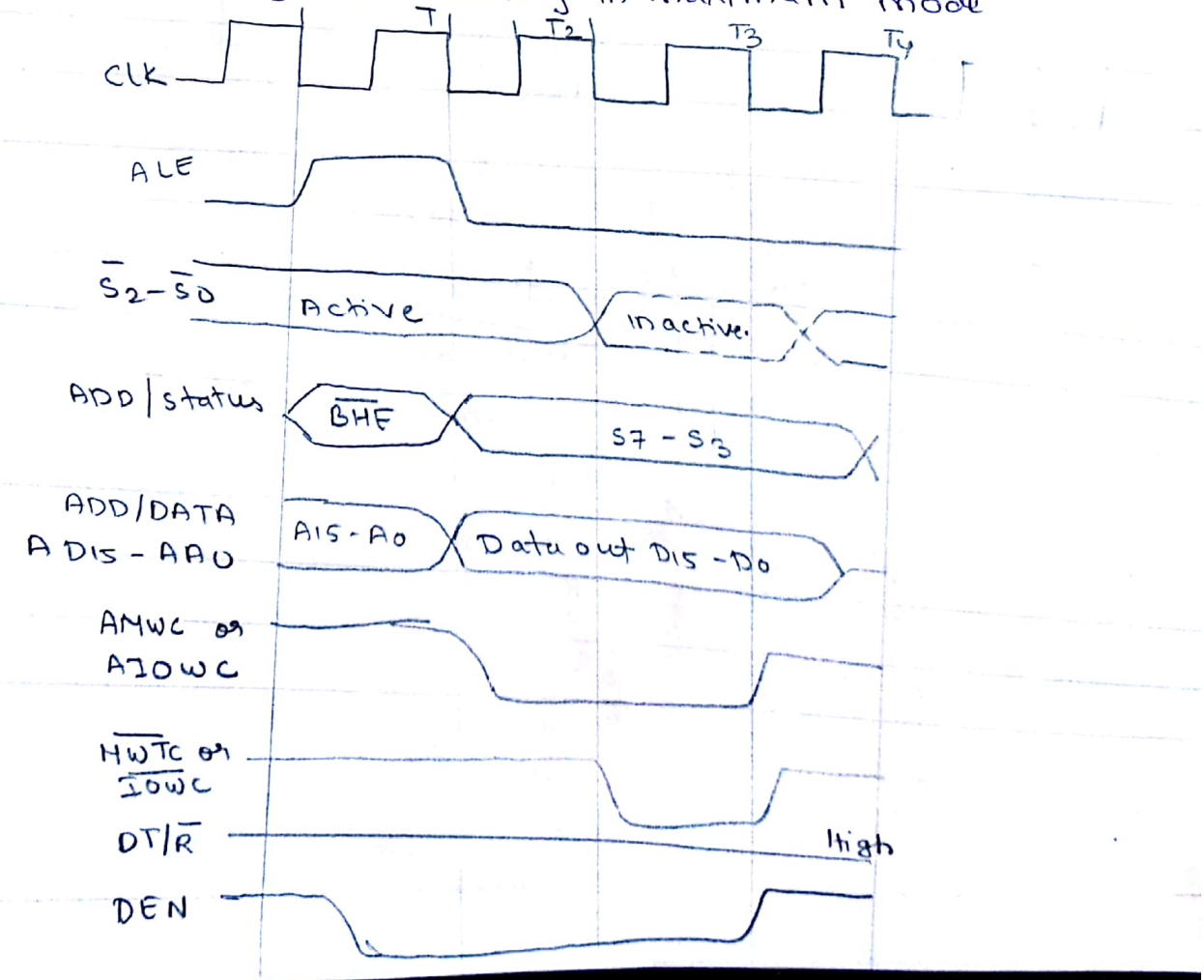
QS_1	QS_0	Indication.
0	0	No operation
0	1	first byte of opcode from the queue.
1	0	Empty queue.
1	1	Subsequent byte from the queue.



memory Read Timing in maximum mode



memory write Timing in maximum mode



→ The 8086 is operated by strapping MN/$\overline{MX}$ pin to ground.

→ In this mode, the processor derives the status signals $\overline{S}_2$, $\overline{S}_1$ and $\overline{S}_0$

→ Bus controller derives the control signal using this status information

→ bus controller chip IC 8288 basic function is to derive control signals like ~~$\overline{RD}$ and $\overline{WR}$~~ $\overline{RD}$ and $\overline{WR}$ (for memory and I/O devices)

$\overline{DEN}$

DT/$\overline{R}$

ALE.

→ The bus controller chip has input lines $\overline{S}_2$, $\overline{S}_1$ and $\overline{S}_0$ and CLK.

→ The o/p of bus controller or it derives ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC IOWC and AIOWC.

→ INTA pin is used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.

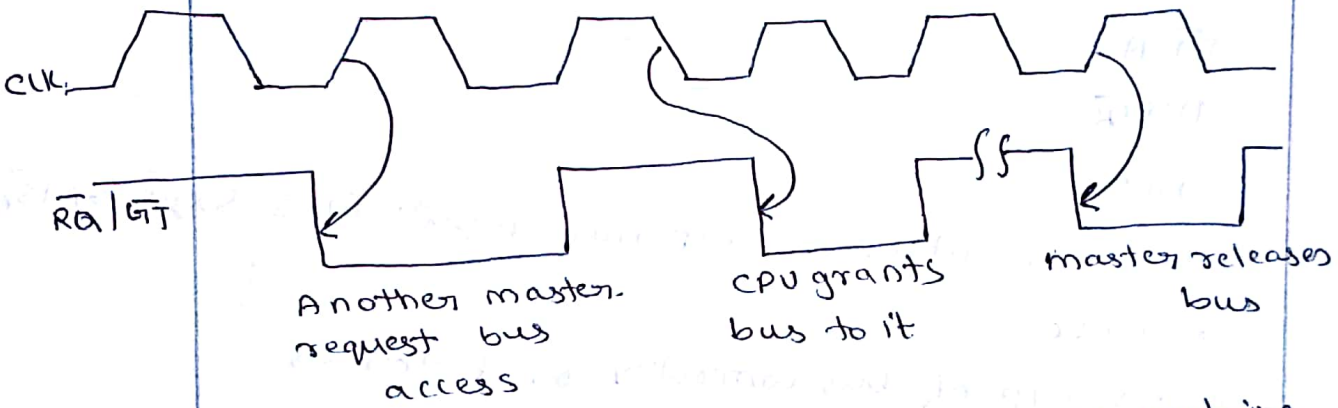
IORC → I/O read command signal

IOWC → I/O write command signals

This signals enable an IO interface to read or write the data from or to the addressed port.

$\overline{MRDC}$ → memory read command
MWTC → memory write command
are used as memory read and write signal.
The maximum mode system timing diagrams are also divided in two portions as read (Input) and write (output) timing diagram.
The only difference lies in the status signals used and the available control.

Timing for $\overline{RQ}$ / $\overline{GT}$ signals



The request / grant response sequence contains a series of three pulses
The request / grant pins are checked at each rising pulse of clock input.
when the request is detected and if the condition are satisfied, the processor issues a grant pulse over the $\overline{RQ}$ / $\overline{GT}$ pin immediately during the T_4 (current) or T_1 (next) state. when the requesting master receives this pulse, it accepts the control of the bus. The requesting master uses the bus till required. when ready to leave it sends a release pulse to the processor using RQ/GT pin

8086 Instruction set and assembler Directives

Machine language instruction formats

A machine language instruction format has one or more number of fields associated with it.

The first field is called as operation code field or opcode field. which indicates the type of operation to be performed by the CPU.

The instruction format also contains other fields known as operand fields. The CPU executes the instruction using the information which resides in these fields

There are six general formats of instruction in 8086 instruction set.

The length of an instruction may vary from one byte to six byte.

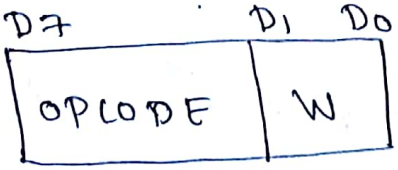
1. one byte instruction
2. Register to Register
3. Register to/from memory with no Displacement
4. Register to/from memory with Displacement
5. Immediate operand to Register
6. Immediate operand to memory with 16-bit Displacement

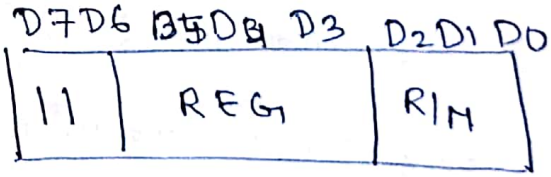
① One byte instruction

This format is only one byte long and may have the implied data or Register operands. The least significant 3-bits of the opcode are used for specifying the registe operand, if any

② Register to Register

This format is 2 bytes long. The first byte of the code specifies the operation code and width of the operand is specified by W bit.
The 2nd byte of the code shows the Register operand and R/M field.
R/M field register or memory location

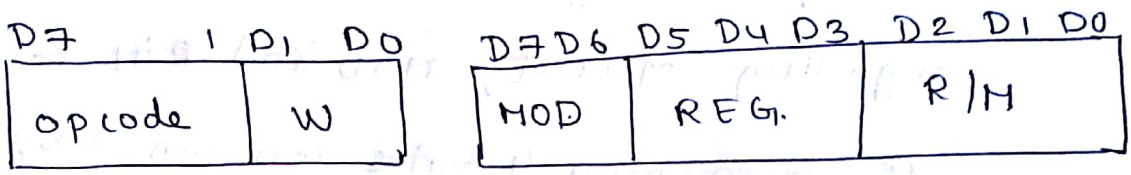




W-bit → This indicates whether the instruction is to operate over 8-bit or 16-bit data/operands.
If w bit is 0, the operand is of 8-bit and if w is 1, the operand is of 16 bit.

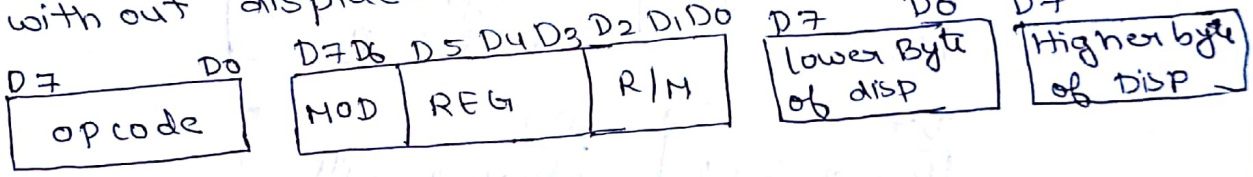
(3) Register to / from memory with no Displacement

This format is also 2 bytes long and similar to the register to register format except for the MOD field.



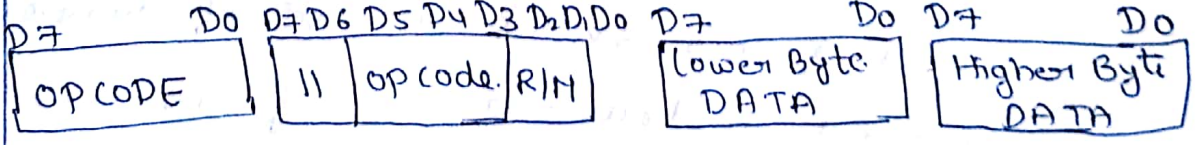
(4) Register to / from memory with Displacement

It contains one or two additional bytes for displacement along with 2-byte the format of the register to / from memory with out displacement



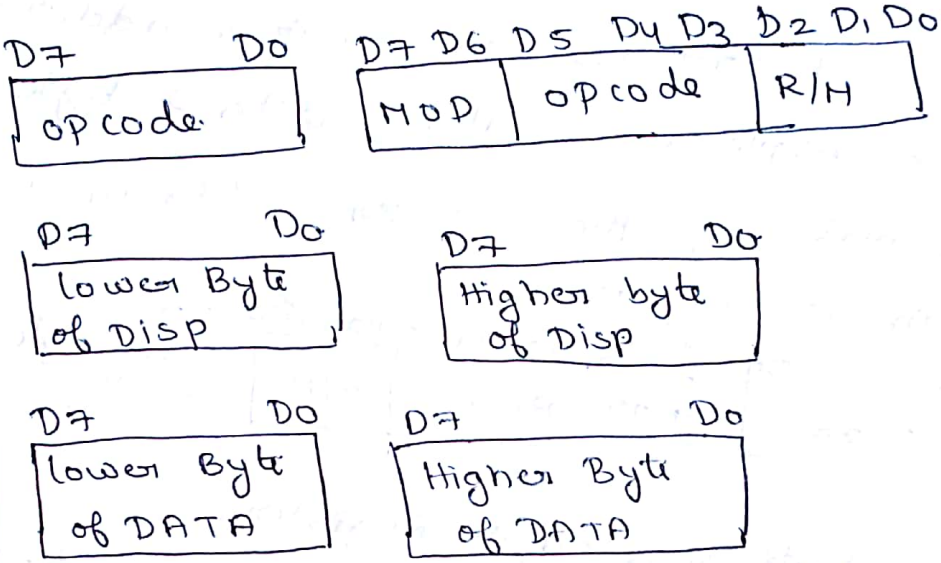
(5) Immediate operand to Registers

The first byte as well as the 3-bits from the the second byte are used for REG field. in case of register to register format are used for opcode. It also contains one or two bytes of immediate data.



6. Immediate operand to memory with 16-Bit Displacement.

It requires 5 or 6 bytes for coding. the first 2 bytes contain the information regading OPCODE, MOD and R/M fields. The remaining 4 bytes contain 2 bytes of displacement and 2 bytes of data.



W-bit → This indicates whether the instruction is to operate over an 8-bit or 16-bit if w bit is 0, the operand is of 8 bit and if w is 1, the operand is of 16 bit

D-bit → This is valid in case of double operand instruction. one of the operand must be a register specified by the REG field. The register specified by REG is source operand if D = 0 else it is

a destination operand

S-bit

This bit is called as sign extension bit

The Sbit is used along with w-bit to show the type of the operation.

V-bit

This is used in case of shift and rotate instruction.

This bit is set to 0, if shift count is 1

If set to 1, if CL contains the shift count.

Z-bit

This bit is used by REP instruction to control the loop. If Z bit is equal to 1, the instruction with REP prefix is executed until the zero flag matches the Z bit

The REG code of the different registers in the opcode byte are assigned with binary code. The segment registers are only 4 in number hence 2 binary bits will be sufficient to code them.

ASSIGNMENT OF CODES WITH DIFFERENT REGISTERS

W bit	Register Address (Code)	Registers
0	0 0 0	AL
0	0 0 1	CL
0	0 1 0	DL
0	0 1 1	BL
0	1 0 0	AH
0	1 0 1	CH
0	1 1 0	DH
0	1 1 1	BH
1	0 0 0	AX
1	0 0 1	CX
1	0 1 0	DX
1	0 1 1	BX
1	1 0 0	SP
1	1 0 1	BP
1	1 1 0	SI
1	1 1 1	DI

Segment 2 bit Register code.

0 0	ES
0 1	CS
1 0	SS
1 1	DS

Addressing modes and the corresponding MOD, REG and R/M fields

operands MOD / R/M	memory operands: No displacement 00	Displacement 8-bit 01	Displacement 16-bit 10	Register operand 11 W=0	Register operand 11 W=1
000	(BX)+(SI)	(BX)+(SI)+D8	(BX)+(SI)+D16	AL	AX
001	(BX)+(DI)	(BX)+(DI)+D8	(BX)+(DI)+D16	CL	CX
010	(BP)+(SI)	(BP)+(SI)+D8	(BP)+(SI)+D16	DL	DX
011	(BP)+(DI)	(BP)+(DI)+D8	(BP)+(DI)+D16	BL	BX
100	(SI)	(SI)+D8	(SI)+D16	AH	SP
101	(DI)	(DI)+D8	(DI)+D16	CH	BP
110	D16	(BP)+D8	(BP)+D16	DH	SI
111	(BX)	(BX)+D8	(BX)+D16	BH	DI

ADDRESSING MODES OF 8086

Addressing mode indicates a way of locating data or operands depending upon the data types used in the instruction. According to the flow of instruction execution, the instruction may be categorised as

(1). sequential control flow instruction.

2). control transfer instruction

sequential control flow instructions are the instructions which after execution transfer control to the next instruction appearing immediately after it in program

eg: Arithmetic, logical, data transfer - - -

The control transfer instructions are the instructions which ~~after~~ after execution, transfer control to some pre defined address in the program.

eg: INT, CALL, RET, JUMP - - -

Addressing modes for sequential control flow instruction

1). Immediate →

Here, immediate data is part of the Instruction.

It may be 8 bit and 16 bit in size

eg MOV AX, 0015H

Here 0015H is immediate data, which is stored into AX register

Effective address → NO effective address

(2) Direct →

In the direct addressing mode, a 16-bit memory address (offset) of the data is specified in the instruction.

If offset we have to find the effective address

eg ⇒ MOV AX, [1000H]

If any location is given in bracket then it will be OFFSET address

Here the data resides in the memory location [1000] is copied into AX register

[1000H] offset address

DS segment address

effective address 10H * DS + 1000H

3) Register → there, the data is stored in a register and it is referred using the particular register. All the registers except IP may be used in this mode.

eg ⇒ MOV AX, BX : the content of BX is copied into AX register there is no effective address

4- Register Indirect →

The address of memory location which contains data or operand is determined in an indirect way, using the offset register BX

eg → MOV AL, [BX]

The content in the address specified by BX register is copied in the AL register

BX OFFSET ADDRESS
DS ES Data present in memory location

effective address 10H * DS + [BX]

5. INDEXED

The address of memory location which contains data or operand is determined in an Indirect way, using the indexed registers. It is known as Indexed addressing mode.

eg MOV AL, [SI] (or) MOV AL, [DI]

Here the content in the address specified by SI (or) DI is copied into AL

effective address =

10H * DS + SI/DI

6). Register Relative. ⇒

Here, data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI, DI

eg: MOV AL, 50H [BX]

Here the content in the address specified by [Bx+50] is copied in the AL

Effective address : 10H * DS + [BX] + 50h

(7). Base indexed.

Here the effective address is formed by adding content of a base register (Bx (or) BP) to the content of an index register (SI / DI)

eg: MOV AL, [BX] [SI]:

Here the content in the address specified by [BX + SI] is copied into AL register.

Effective address : 10H * DS + [BX] + [SI]

(8). Relative Based Indexed ⇒

The effective address is formed by adding an 8 (or) 16-bit displacement with the sum of content of any one of the base registers (BX (or) BP) and any one of the index registers

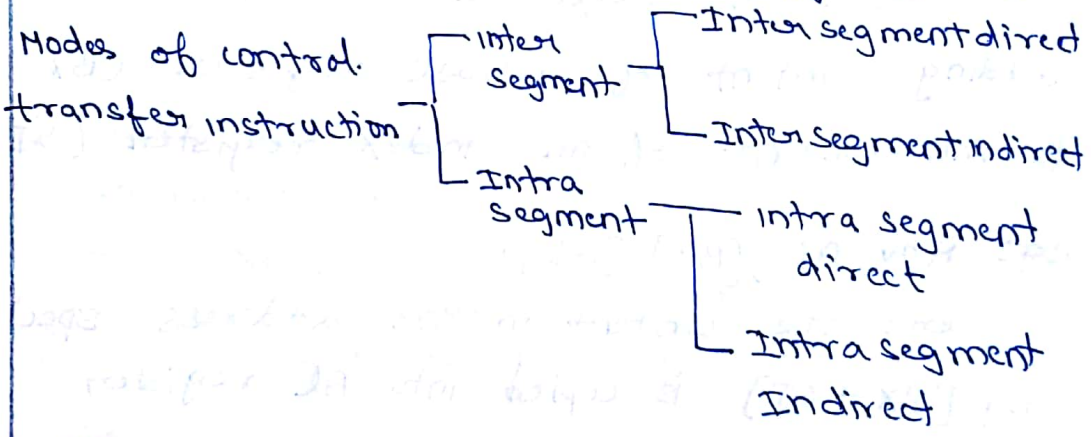
eg→ MOV AL, 50H [BX][SI]:

Here, the content in the address specified by [50+BX+SI] is copied into AL registers

Effective address: 10H*DS + 50 + BX + SI

ADDRESSING MODES FOR CONTROL TRANSFER INSTRUCTIONS

For the control transfer instruction, the addressing modes depends upon whether the destination location is within the same segment or in a different one



If location to which the control is to be transferred lies in a different segment other than current one, the mode is called intersegment mode. If the destination location lies in the same segment, the mode is called intra segment mode

9. Intra segment Direct mode → Here, the address to which the control is to be transferred lies in the same segment in which the control transfer instruction lies and appears directly in the instruction as an immediate displacement value.

eg: Jmp up :

10). Intra segment Indirect mode → Here the address to which the control is to be transferred is in the same segment in which the control transfer instruction lies, but it is passed to the instruction indirectly.

eg: Jmp [BX];

11) Intersegment Direct → Here the address to which the control is to be transferred is in a different segment. The address to which the control transfers is specified directly in the instruction.

eg: Jmp 5000H : 2000H;

12). Inter segment Indirect : Here, the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly

eg: Jmp [2000H]

Instruction Set of 8086:

Data copy/Transfer Instructions:

Instruction	operation	Example.

MOV	Copies content ~~from~~ to destination from ~~to~~ source.
	source: Immediate, direct, Register, indirect
	destination: Register, direct, indirect
		MOV Ax, Bx ;
		MOV Ax, 1111h
		MOV [SI], Al

PUSH	copies content in to stack. SP decrement by 1 for byte ope. SP dec. by 2 for word operation.	Push Ax push Ds push [5000]
POP	copies content from stack to specified Register/Memory/. SP inc. by 1 (or) 2.	Pop Ax POP [5000].
XCHG	this instruction exchanges the contents of specified source & destination.	XCHG Ax, Bx XCHG [5000], Ax
* IN	this instruction used to read the i/p port. ~~source~~ destin : Al (or) Ax only ~~destination~~ source : address (or) Dx only.	IN Al, Dx IN Ax, FFFFh.
* OUT	this instruction is used for writing to an output port. ~~source~~ destin. : address (or) Dx only ~~destination~~ source : Al (or) Ax only	out Dx, Ax out FFF9h, Al.
XLAT	this inst. copies content in to Al from the address specified by [Al+Bx]. It uses look up tables.	XLAT

Instruction	operation	Example.
LEA Load effective Address	this ins. loads the effective address formed by ~~destination~~ source operand in to the specified ~~source~~ destination register	LEA BX, ADR
LDS/LES Load Pointer to DS/ES	This instruction loads DS/ES and the specified destination register with the content of memory location specified in the source.	LDS BX, 5000h. 5000:01 5001:02 5002:03 5003:04 DS:0403 BX:0201
LAHF	This instruction loads the AH register with the lower byte of flag register.	LAHF
SAHF	This instruction loads the lower byte of flag register with contents of AH.	SAHF
PUSHF	This ins. Pushes the contents of flag register in to stack. SP decremented by 2	PUSHF
POPF	This ins. load the flag register with contents of stack. SP incremented by 2	POPF

Arithmetic Instructions:

Instruction	operation	Example.
ADD	The content of source and destination operands are added. and result is stored in to destination. source: immedi, direct, Register, indirect destinat: direct, Register, indirect.	Add Al, Bl. Add Ax, [5000] Add Ax, 1111h.
ADC	the content of source & dest. operands are added with carry flag and result is stored in to destination.	Adc Al, Bl Al ← Al+Bl+C.F
INC	the content of specified operand is incremented by 1. (direct/ Regis/indirec)	inc Al
DEC	the content of direct/ Register/indirect is decremented by 1.	dec Al
SUB	the content of destination operand is subtracted with source operand. & result is stored in to destination.	sub Al, Bl Al ← Al - Bl
SBB	the content of destination ope. is subtracted with source ope. & with carry flag & result stored in destinat.	Sbb Al, Bl Al ← Al-Bl-C.F
CMP	this instruction compares the source and destination operands, result reflects in Flag register: source: imm / direc / reg / indir. desti : dir / reg / indir.	cmp Al, Bl cmp Bx, [SI]

Instruction	operation	Example.
AAA Ascii Adjust after Addition.	After addition of two Ascii numbers, to make result valid. AAA instruction must be used.	AAA
AAS Ascii Adjust AL after subtr.	After subtraction of two Ascii no. to make result valid. AAS instru. must be used.	AAS
AAM Ascii adjust after multiplicat	After Multiplicati of two Ascii no. to make result valid. AAM instr. must be used	AAM
AAD Ascii Adjust before division	Before performing Division op. of Ascii no. AAD shold write. It converts two unpacked BCD to its equ. binary.	AAD.
DAA Decimal Adjust Accumulator.	It converts the result of the additio of two packed BCD no. to a valid BCD. It affects AF, CF, PF & ZF flags.	DAA
DAS Decimal adjust after subtract.	It converts result of sub. of two packed BCD no. to a valid BCD no. It affects: AF, CF, SF, PF & ZF flags.	DAS
NEG Negate	It performs 2's compliment of specified operand, and result is stored in the same operand.	Neg Al Neg [1111h]
CBW	converts a signed byte to a signed word. It copies the sign bit of a byte to be convert in to higher byte of word source : Al (only), dest : Ax (only).	CBW
CWD	converts a signed word to double word source : Ax (only), dest : Dx Ax (only)	CWD

Scanned by CamScanner

instruction	operation	Example.
MUL	Multiplies unsigned source & destination operands, Results stored in to Destination operant. source : Reg / direct / indirect / destination : Al (or) Ax	MUL BL Ax ← Al * BL MUL CX DxAx ← Ax * CX
IMUL	multiplies signed source with dest. operands & result stored in dest. op.	IMUL BX
DIV	It divides an unsigned word (or) double word must be Ax (or) Dx Ax with unsigned byte (or) word. source : Reg/dir/indi ; dest : Ax/Dx Ax	DIV BL Ax / BL : Q : Al, R : Ah DIV BX Dx Ax / Bx : Q : Ax, R : Dx
IDIV	It divides a signed word (or) double word with signed byte (or) word operand. source : Reg / direct / indirect destina : Ax / Dx Ax	IDIV BL

Scanned by CamScanner

logical Instructions :

Instruction	Operation	Example
AND	It performs logical AND oper. between source & desti operand and result stored in desti opera. source : Imm / reg / direct / indirec. desti : reg / direct / indirect.	AND Al, Bl AND [1000], cl AND Ax, [1111]
OR	It performs logical OR operation. source : Imm / reg / direct / indir. desti : reg / dir / indirect.	OR Al, Bl.
NOT	It performs 1's compliment of operand & result stored in same operand ; reg / dir / indir	NOT Al
XOR	It performs logical XOR operation.	XOR Ax, Bx
TEST	It performs logical compare, by ANDing each bit. Result not stores but OF, CF, SF, ZF & PF flags affected.	TEST Ax, Bx Test [0511], 06h.

Scanned by CamScanner

instruction	operation	Example
SHL/SAL Logical/Arithmatic left	These ins. shift the operand bit by bit to the left and insert zero's in the newly introduced LSB's. the count must be in CL.	shl AL, 01 sAL AL, 01 shl AL, CL
SHR shift logical Right	shifts operand bit by bit to right & insert zero's in the newly introduced MSB's.	shr AL, CL
SAR shift Arithmatic right	shifts operand bit by bit to right & insert MSB in the newly introduced MSB's	SAR AL, CL
ROR Rotate right without carry	shift operand bit by bit right & insert LSB in the newly introduced MSB's.	ROR AL, CL
ROL Rotate left without carry	shift operand bit by bit left & insert MSB in the newly introduced LSB's.	ROL AL, CL.
RCR Rotate Right through carry	shifts operand bit by bit right & insert LSB in carry flag, then newly introduced MSB copied with CF.	RCR AL, CL
RCL Rotate left with carry	shift operand bit bybit left, newly introduced LSB is inserted with CF then CF is copied with MSB.	RCL AL, CL.

Scanned by CamScanner

String Manipulation Instructions

Instruction	operation	Example
MOVSB	string of byte stored at DS:SI copied into ES:DI.	MOVSB
MOVSW	word string of word stored at DS:SI copied into ES:DI.	MOVSW
CMPSB/W	It compares a byte string of byte (or) word stored at DS:SI with a string of byte (or) word stored at ES:DI.	CMPSB ; CMPSW
SCASB/W	It compares a string of byte (or) word in AL (or) AX with a string of byte (or) word in ES:DI.	SCASB ; SCASW ;
LODSB/W	string of byte or word stored at DS:SI copied into AL (or) AX register.	LODSB ; LODSW
STOSB/W	string of byte (or) word stored at AL (or) AX register is copied into ES:DI.	STOSB ; STOSW.
REP	It is used as prefix for instruc. It repeats the particular instruction till counter (CX) becomes zero, & decrements CX by 1 each time & increments SI & DI register by 1 each time	REP: MOVSB

Scanned by CamScanner

Control Transfer (or) Branching Instructions:

Control transfer Instructions transfers the flow of execution of the program to a new address specified in the instruction. this type of instructions are classified in to two types:

Unconditional control Transfer Instructions:

The execution control is transferred to the specified location independent of any status (or) condition.

Instruc.	operation	Example.
Call	This instruction is used to call a subroutine. the execution control transfers to subroutine. the address of subroutine specified in the instruction directly/indirectly	Call delay
RET	This instruction returns the execution control from subroutine to Main program. the last instruction of subroutine must be RET. this instruction retsieve addresses of old CS & old IP from Stack.	RET

Instruction	Operation.	Example.
INT N	when an INT N instruction is executed, the type byte N is multiplied by 4 and the content of IP & CS of the interrupt service routine will be taken from vector table. ISR executes.	INT 03h.
INTO	Interrupt on overflow. this command is executes, when the overflow flag OF is set.	INTO
IRET	this instructions returns the execution control from ISR to main program.	IRET
JMP	this instruction unconditionally transfers the control of execution to the specified address in the program.	Jmp up.
LOOP	this instruction unconditionally transfers the control of execution to the specified address untill the CX (counter) becomes zero.	

Scanned by CamScanner

Conditional Branch instructions

Instruction	operation.	exa. f/c.
JZ / JE	Transfer execution control if ZF=1	JZ up
JNZ / JNE	Trans. exe. ctrl if Z.F = 0	JNZ up ;
JS	Trans. exe. ctrl. if S.F :1 (-ve)	JS back ;
JNS	Trans. exe. ctrl. if S.F : 0 (+ve)	JNS back ;
JO	Trans. exe. ctrl. if O.F : 1	JO repeat
JNO	Trans. exe. ctrl. if OF : 0	JNO repeat.
JP / JPE	Trans. exe. ctrl if P.F : 1	JP back.
JNP	Trans. exe. ctrl if P.F : 0	JNP back.
JB / JNAE / JC	Trans. exe. ctrl. if C.F : 1	JC back.
JNB / JAE / JNC	Tran. exe. ctrl. if C.F : 0	JNC up.
JBE / JNA	Tran. exe. ctrl. if C.F:1 & Z.F : 1	JBE back.
JNBE / JA	Tran. exe. ctrl. if C.F : 0 & Z.F : 0	JA up
JL / JNGE	Tr. exe. ctrl. if neither S.F : 1 nor OF : 1	JL up
JNL / JGE	Tr. exe. ctrl if neither SF : 0. nor OF : 0	JNL up.
JLE / JNG	Tr. exe. ctrl. if Z.F : 1 (or) neither SF nor OF is 1	JLE up
JNLE / JE	Tr. exe. ctrl if Z.F : 0 (or) atleast any one of S.F & OF is 1 (both ndo)	JE up.

Conditional Loop instructions.

Instruction	Operation	Example.
LOOPZ / LOOPE	Repeats the loop till Z.F is set.	LOOPZ UP
LOOPNZ / LOOPENE	Repeats the loop till Z.F	LOOPNZ back

Flag Manipulation instructions :

Instruct+	Operation	Example
CLC	clear carry flag	CLC
STC	set carry flag	STC
CMC	compliment carry flag	CMC
CLD	clear Directional flag	CLD
STD	set Directional flag	STD
CLI	clear Interrupt flag	CLI
~~STI~~ STI	set Interrupt flag	STI

Machine control Instructions.:

Wait	Wait for test i/p pin to go low
HLT	halt the processor
NOP	No operation till 4 clock cycles.
ESC	Escape to external devices like NDP
LOCK	Bus lock instruction Prefix

Scanned by CamScanner

Assembler Directives : directions to the Assembler, not instructions for the 8086, for TASM, MASM.

1, DB : Define byte : The DB directive is used to reserve byte or bytes of memory locations in the available memory.

eg: ax db 49h, 98h
eg: ax1 db 'THOMAS'
eg: ax2 db 10 dup(?)

2, DW : Define word : the DW directive is used to reserve word or words of memory locations in the available memory.

eg: ECF db 10 dup(0)

3, DQ : Define Quad word : this directive is used to direct the assembler to reserve 4 words of memory for the specified variable and may initialise it with the specified values.

4, DT : Define Ten Bytes : the DT directive directs the assembler to define the specified variable reserving 10 bytes for its storage and initialise 10 bytes with the specified values.

5, ASSUME : Assume logical segment name : the Assume directive is used to inform the assembler, the name of the logical segments to be assumed for different segments used in the program.

eg: Assume cs:code, ds:data.

6, END : End of Program : the END directive marks the end of an ALP.

7. ENDP : End of procedure : the ENDP directive is used to indicate the end of procedure.

8. ENDS : End of segment : this directive marks the end of a logical segment.

9. EVEN : Align on even memory Address : the EVEN directive updates the location counter to the next even address.

10. EQU : equate : the directive EQU is used to assign a level with a value or a symbol. the use of this directive is just to reduce the recurrence of the numerical values or constants in a programming code.

eg: x EQU 03h

11. EXTRN : External and public : the EXTRN directive is used to tell the assembler that the names or labels following the directive are in some other assembly module.

eg: Extrn divisor: word.

12. GROUP : Group the Related segments : the GROUP directive is used to tell the assembler to group the logical segments named after the directive into one logical group segment.

eg sys Group code, data, stack, Extra.

Assume cs: sys, DS: sys, SS: sys, ES: sys.

Scanned by CamScanner

13. LABEL : label : The LABEL directive is used to assign a name to the current content of the location counter.

14. LENGTH : Byte length of a label : Actually this directive is not available in MASM. This is used to refer to the length of a data or a string

eg: MOV CX, length arl.

15. OFFSET : offset of a label : when the assembler comes across the 'OFFSET' operator along with a label, it first computes the 16-bit displacement of a particular label, and replaces the string 'OFFSET LABEL' by the computed displacement.

eg: MOV SI, offset arl.

16. PROC : Procedure : The PROC directive marks the start of the named procedure in the statement.

17. SEG : segment of a label : The SEG operator is used to decide the segment address of a label, variable (or) procedure.

18. SEGMENT : logical segment : The segment directive marks the starting of a a logical segment.

eg: code segment

eg: data segment

Scanned by CamScanner

19. GLOBAL : The labels, variables, constants or procedures declared 'GLOBAL' may be used by other modules of the program.

eg: Global Divisor : word
→ divisor is variable of type word.

20. PUBLIC : 'PUBLIC' directive is used along with the EXTRN directive this informs that the labels variables, or procedures declared PUBLIC may be accessed by other assembly modules

eg: Public Divisor, Dividend.

21. ORG : origin : The 'ORG' Directive directs the assembler to start the memory allotment for the particular segment, block or code from the declared address in the ORG statement.

eg: ORG 2000h.

22. PTR : Pointer : The pointer operator is used to declare the type of a label, variable or memory operator. The operator PTR is prefixed by either Byte or word.

eg: INC [Bx] — ? byte (or) word
so
eg: INC byte PTR [Bx]
INC word PTR [Bx]

23. TYPE : The type operator directs the assembler to decide the data type of a specified label and replaces the type label by the decided data type. for a byte-type, the assembler give a value 1, for a word-type, the assembler give a value 2,

eg
Add Bx, type Arr1.

Scanned by CamScanner