# 8086 Addressing Modes



→ Each instruction specifies an operation to be performed on certain data called operand.
→ The various methods in which the processor can access the data are referred as addressing modes.
→ The 8086 processor uses the following addressing modes :-
(i) Register (ii) Immediate (iii) Direct (iv) Indirect (v) Register relative (vi) Base indexed (vii) Relative based indexed.

## (1) Register Addressing

Operand to be accessed is specified in one of the internal register of 8086 except IP. In this, both source and destination are registers of 8086.

eg MOV AX, CX
→ This will move the content of CX into AX.

## (2) Immediate Addressing

This ~~moves~~ immediately perform the instruction using data.

eg MOV BL, 20H
→ This will move the 20H data into BL register.

## (3) Direct Addressing

In this, the 16-bit data (offset) is directly specified with in the ~~specif~~ instruction itself.

eg MOV AX, [1234]

→ Move the content of memory location with offset address 1234H to internal register AX. When this instruction is executed, the 8086 combines 1234H with the current contents of data segment to get the physical address of the source operand and whose content is transfer to the destination register.

(4) Indirect Addressing

→ In this, the offset address of operand is resides in either a base register (BX or BP) or an index register (SI or DI)

eg MOV AX, [SI]

→ Move the contents of memory location whose offset is equal to the contents of source index register, into the internal register AX).

→ When the instruction is executed, the 8086 combines the contents of source index register with the contents of data segment to get the physical address of the source operand whose content is transfer to the destination address.

(5) Register Relative Addressing

In this, the effective address of the operand is the sum of an 8-bit or 16-bit number (called displacement) and the contents of either a base register (BX or BP) or an index register (SI or DI)

eg:- MOV AX, [BX + 0003H]

→ This will move the contents of memory location whose offset address is equal to the sum of displacement 0003H and the contents of the base register, into the internal register AX.

⑥ Base Index Addressing

In this, the offset address of the operand is the sum of the contents of any base register (BX or BP) and the sum of any index (DI or SI)

eg MOV AX, [BX][SI] ⇒ MOV AX, [BX + SI] means

→ This will move the contents of memory location whose offset is equal to the sum of contents of base register (BX) and the contents of index register (SI) into the internal register AX.

⑦ Relative Based Index Addressing

When this instruction is executed, the 8086 combines the effective address with the contents of data segment to get the physical address of the source operand, whose contents are transfer to the destination ~~set~~ register.

eg MOV AX, [BX] [SI] + [1234H]

↑ Base register ↑ Index Register ↑ Displacement

# 8086 Instruction Set

## (1) Data Transfer Instructions

### (a) Move Byte or Word
8 bits 16 bits

→ This moves the contents of source operand to destination operand.

eg MOV AX, CX
MOV AX, 1234H
MOV AX, [1234H]

* MOV AX, [1234]
Let the contents of Data segment register is equal to 0500H. Then the physical address of the source operand is –

PA = 0500H × 10H + 1234H
= 06234H

8 bit data / 1 byte data is there at memory location 06234H and we have to store the content in 16 bit register i.e AX, So, when the execution takes place, the 8086 moves the content of memory location whose address is 06234H to AL register and the contents of memory location ~~move to~~ 06235H to AH register.

(b) XCHG D, S

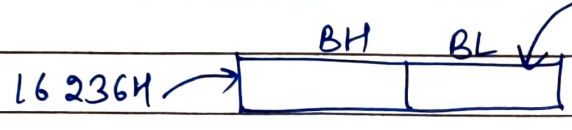
[The contents of source operand are exchanged with the destination operand]

eg (i) XCHG BX, [1235H]

Let contents of DS = 1500H

PA = (1500 × 10)H + 1235

= 16235H




(ii) XCHG BL, [1235H]

PA = 16235H

BL

Contents of 16235H address.

(C) XLAT

The contents of AL are replaced by the contents of address look up table.

eg:-

Let DS = 1500H

BX = 0010H

AL = 01H

then after execution of XLAT instruction, the contents of AL are replaced by the contents of the physical address of source operand i.e 15011H.

PA = DS×10 + BX + AL

= 15000 + 0010 + 01

= 15011H

## (d) Load Instruction.

### (i) LDS D, M

→ Load the data segment register and other specific register from memory.

eg:- LDS DI, [1234H]

Let DS = 5000H

PA = 51234H

After executing LDS command.

The content of 51234H → DI (lower byte)
51235H → DI (Higher byte)
51236H → DS (Lower)
51237H → DS (Higher)

### (ii) LES D, M

→ Load the extra segment register and other specified register from memory.

eg:- LES DI, [1234H]

After executing this if DS = 5000H.

The content of 51234H → DI (lower)
51235H → DI (higher)
51236H → ES (lower)
51237H → ES (Higher)

### (iii) LEA D, EA

load effective offset address of operand into the specified register.

eg LEA CX, [BX]
Content of BX is stored to the CX register.

(2) ARITHMETIC INSTRUCTIONS

(i) ADD Instruction

→ This is used to add contents of destination with that of source and store the result in destination
→ All the flags are modified.
→ General format
ADD Destination, Source.

eg:- ADD AL, 0FH
ADD AX, BX
ADD AX, [SI]
ADD 0100H

(ii) ADC

→ Perform addition but also adds the carry flag bit to the result.
→ All flags are affected/ modified.

eg:- ADC AX, BX
ADC AX, [SI]
ADC AX, [5000H]
ADC 0100H

(iii) SUB instruction

→ This is used to subtract the contents of destination with that of source and store the result in destination

→ All flags are modified.

→ General format SUB Destination, Source

eg

```
SUB AL, 0FH        (AL-0FH)
SUB AX, BX         (AX-BX)
SUB AX, 0100H
SUB AX, [5000H]
```

(iv) SBB

→ Subtracts the source operand and the borrow flag from the destination.

→ Means subtracting 1 from the subtraction obtained by SUB.

eg:

```
SBB AX, 0100H      (AX - 0100H - Borrow)
SBB AX, BX
SBB AL, 0FH
SBB AX, [5000H]
```

(v) CMP

→ compares the source operand which may be a register or an immediate data or a memory location with a destination operand.

→ Subtracts the source operand from the destination

→ Cases:

- Source = Destination ; zero flag = 1

- source > destination  Carry flag = 1
- source < destination  Carry flag = 0

eg:
```
CMP BX, 0100H
CMP AX, 0100H
CMP BX, [SI]
CMP BX, CX
```

(vi) INC instruction

→ Used to increment the contents of specified destination.

→ All the flags are modified.

eg:
```
INC AL      (AL+1 = AL)
INC AX      (AX+1 = AX)
```

(vii) DEC instruction

→ Used to decrement the contents of specified destination

→ All flags are modified.

eg
```
DEC AL      (AL-1 = AL)
DEC AX      (AX-1 = AX)
```

## (3) LOGICAL INSTRUCTIONS

(i) AND instruction

→ Logically AND each bit of the source byte/word with the corresponding bit of destination.

→ Result is stored in destination

→ CF, OF = 0

→ Z, P, S are modified

→ AC flag is undefined

→ General format : AND Destination, Source

eg: AND BL, AL

If BL = 1000 0110
AL = 1100 1010
Result BL = 1000 0010

(ii) OR Instruction
→ logically OR each bit of source with destination
→ CF, OF = 0
→ Z, P, S are modified
→ AC flag is undefined.
→ General format: OR Destination, Source.

eg OR BL, AL
if BL = 10000110
AL = 11001010
Result BL = 11001110

(iii) NOT instruction
→ complement the content of an operand.
→ General format NOT operand.

eg NOT AX
if AX = 1011 1011 1011 1011
NOT AX = 0100 0100 0100 0100

(iv) XOR instruction

→ XOR operation is carried between the source and destination.

eg XOR AX, 0098H

if:

AX = 0011 1100 0101 1101

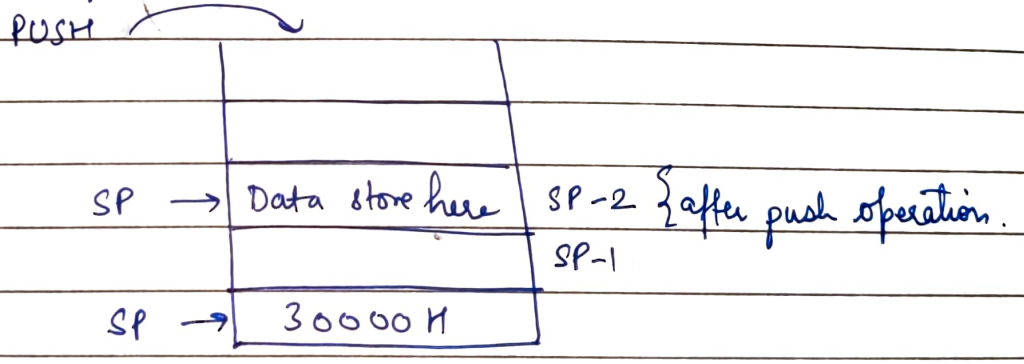
0098H = 0000 0000 1001 1000

XOR = 0011 1100 1100 0101 → stored in AX register

* continuation with data transfer instructions

(e) PUSH

→ It decrements the stack pointer by two and copies the word from source to the location where stack pointer now points.




example :- PUSH CX

PUSH DS

SP = 0034

SS = 3000

PA = 3000 x 10 + 0034

= 30034 H ← initial stack address

After push operation

SP = 30032

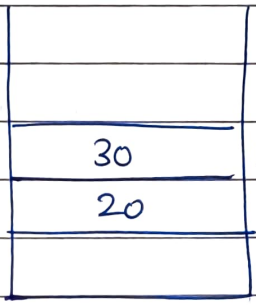
| CH | CL |
|---|---|
| 20 | 30 |

At 30032 = 30

30031 = 20

(f) POP

In this the address is incremented by two and it is used to fetch data.




After pop instruction

POP CX

| CH | CL |
|---|---|
| 20 | 30 |

→ Copies a word from the stack location pointed by SP to the destination.

→ After the content is copied the stack pointer is automatically incremented by two.

(g) IN and OUT

→ IN will copy data from input port to the accumulator.

→ OUT will copy data from accumulator to output port.

eg:- IN AL, 0F8H

copy data from 0F8H to AL

OUT 047H, AL

copy data from AL to 047H

(3) STRING INSTRUCTIONS

DF = 1, address is decremented
DF = 0, address is incremented

→ String is a group of bytes/word and their memory is always allocated in sequential order.

(a) MOVS / MOVSB / MOVSW

- MOVS: Move String (bit by bit)
- MOVSB: Move String Byte (8 bit)
- MOVSW: Move String word (16 bit)

→ These instructions copy a word or byte from a location in the data segment to other location in extra segment.

Note
→ offset of source → source Index
→ offset of destination → Destination Index } for a word/byte.

→ For multiple byte/word transfer the count is stored in the CX register.

→ When direction flage is 0, SI and DI are incremented.
→ When DF = 1, SI and DI are decremented.

example :-

```
CLD      { clear the direction } then SI and DI is incremented.
MOV AX, 0000H
MOV DS, AX      } DS carry source & ES carry destination offset address.
MOV ES, AX
MOV SI, 2000H
MOV DI, 2400H
MOV CX, 04H     length of string = 04
REP MOVSB       { Repeated till count reg. becomes to zero.
```

## REP/REPE/REPZ/REPNE/REPNZ

→ It repeats an instruction until the specified condition becomes false.

eg:- REP → until CX = 0 becomes false
REPE/REPZ → until CX = 0 or ZF = 1 " "
REPNE/REPNZ → until CX = 0 or ZF = 0 " "

## (c) LODS / LODSB / LODSW

→ copies a bytes from a string location pointed by SI to AL or a word from a string location pointed by SI to AX.

Byte → AL (8-bit)
word → AX (16-bit)

example:

```
CLD          } clear direction flag.
LODS S_STRING
```

## (d) STOS / STOSB / STOSW

→ used to store a byte/word contained in AL/AX to destination (offset contained in DI).
→ does not affect any flag.
→ DI → incremented/decremented based on direction flag.

example:

```
MOV DI , OFFSET (D_STRING)
STOS D_STRING.
```

## (e) CMPS / CMPSB / CMPSW

→ Compare the strings, byte size or word size
→ comparision is affected by subtraction of content pointed by DI from that pointed by SI.

→ AC, CF, OF, SF, ZF get affected.

example :-

```
MOV SI , OFFSET (string)
MOV DI , OFFSET (F_string)
MOV CX , 0AH
CLD
REPE CMPSB
```

compare string byte size.

(f) SCAS / SCASB / SCASW

→ scan a string and compare its byte with a byte in AL or string word with a word in AX.

(g) OUT / OUTSB / OUTSW

→ output string / byte / word from the memory location to the I/O port.

(4) Flag Manipulation Instructions

→ CLC - Clear carry flag
CMC - complement carry flag
STC - set carry flag
CLD - Clear direction flag
STD - set direction flag
CLI → Clear Interrupt flag.
STI → set interrupt flag.

Q. Write an 8086 assembly language program to set the contents of lower byte of the flag as follows:
SF=0, ZF=0, AC=1, PF=1, CY=0

## #5 Machine Control Instructions

(a) Wait
→ these instructions are inserted with other program instruction until the TEST (input) pin goes low (logic 0).

(b) HLT (Halt) (Idle state)
→ procesor comes out from idle when triggered by external interrupt.

(c) NOP (No operation)

(d) ESC (Escape)
→ control goes from uP to Co-procesor.

(e) LOCK
→ ~~thus~~ when this is executed, the uP gains complete control of buses until the LOCK prefix instruction is executed.

LOCK MOV AX, BX } After the instruction is completed uP buses can be used for other purposes.

## (6) Branch Instruction

### (a) JUMP

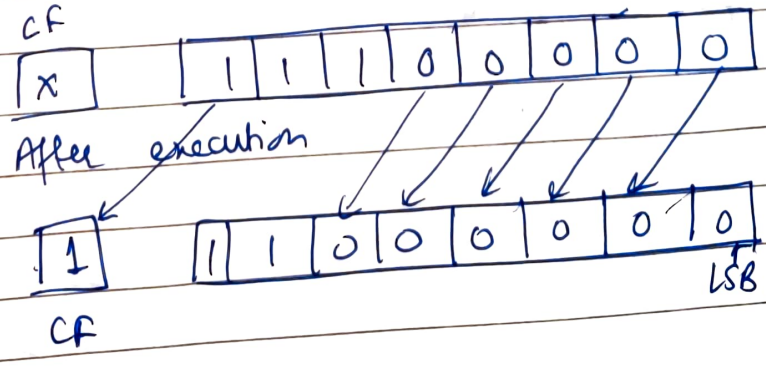
JZ → Jump when Z = 1
JNZ → Jump when Z = 0
JC → Jump when CY = 1
JNC → Jump when CY = 0
JPE → Jump if P = 1 (even parity)
JPO → Jump if P = 0 (odd parity)
JM → Jump if S = 1 (negative)
JP → Jump if S = 0. (positive)

(b) CALL
(C) Return
} Same as 8085 instructions.

## (7) Shift and Rotate Instruction

(a) SAL / SHL : Arithmetic/logical left shift.
→ Each bit is shifted one position to left.
→ LSB is cleared to 0
→ MSB is moved to CF.

example

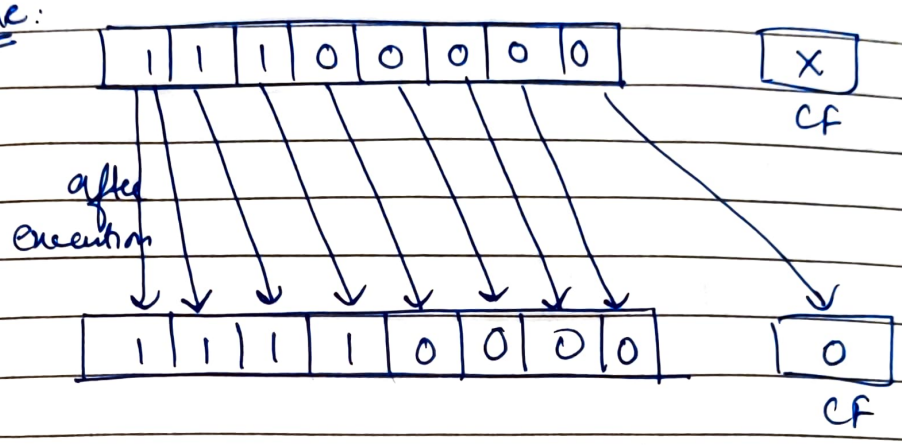


(b) SAR Shift Arithmetic Right

→ MSB of destination is filled with its previous value

→ LSB of Destination is moved to CF.

example:

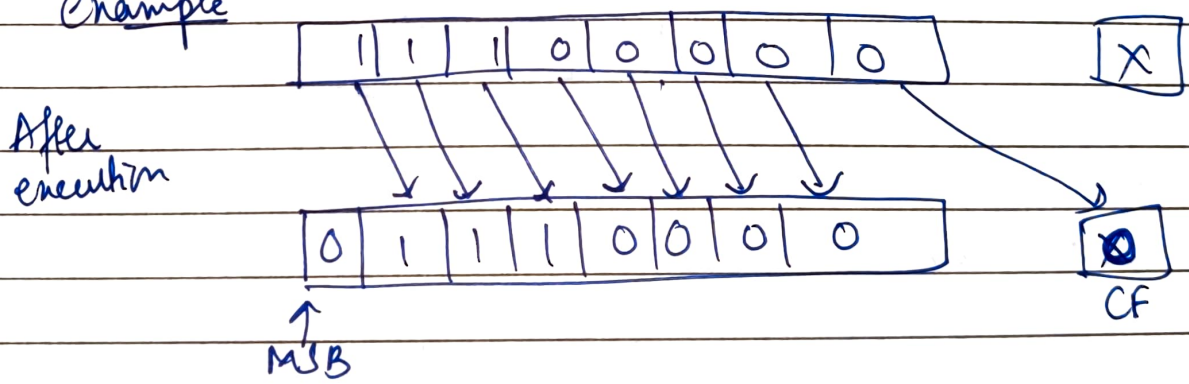



(c) SHR : Shift logical Right

→ MSB of destination = 0

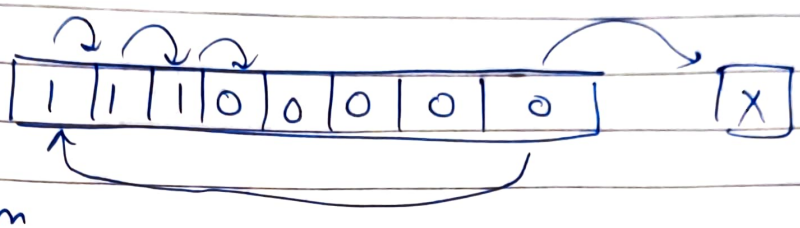
→ LSB of destination is moved into CF.

Example

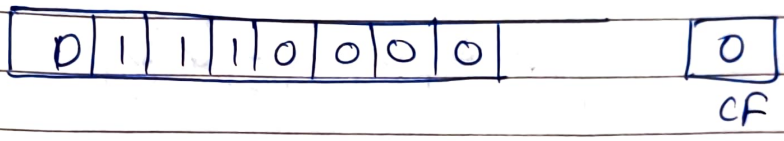


(d) ROR : Rotate right without carry
→ Rotate bits to the right.
→ CF get a copy of LSB.

example

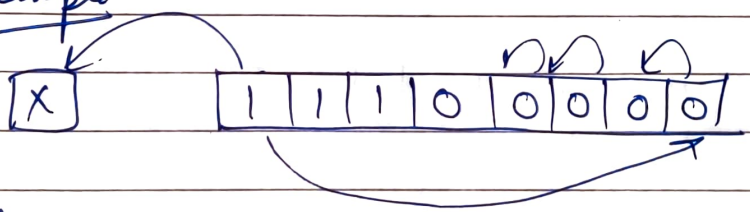



After execution

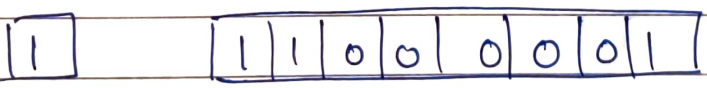



(e) ROL : Rotate left without carry.
→ Rotate bits to the left
→ CF gets a copy of MSB.

example

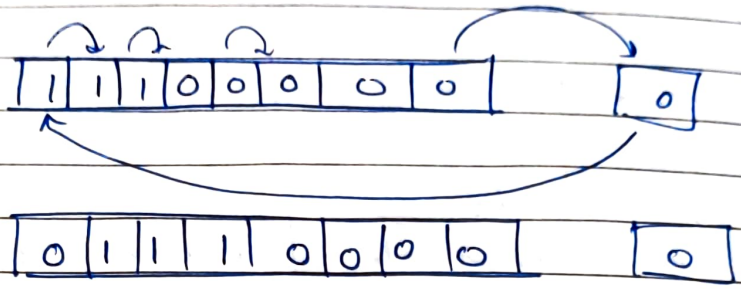



After execution

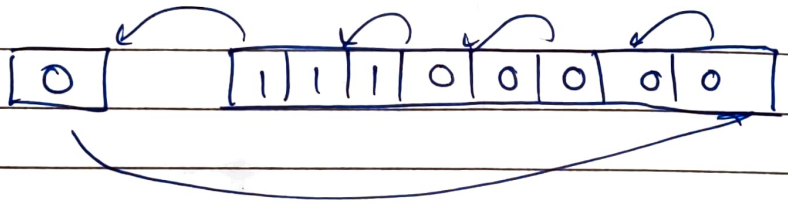


(f) RCR: Rotate Right through carry
→ Rotate bits to right
→ Cf gets a copy of LSB.

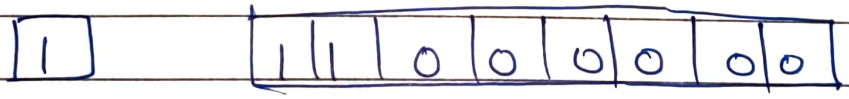
Example




(g) RCL: Rotate left through carry
→ Rotate bits to the left.
→ Cf gets a copy of MSB.



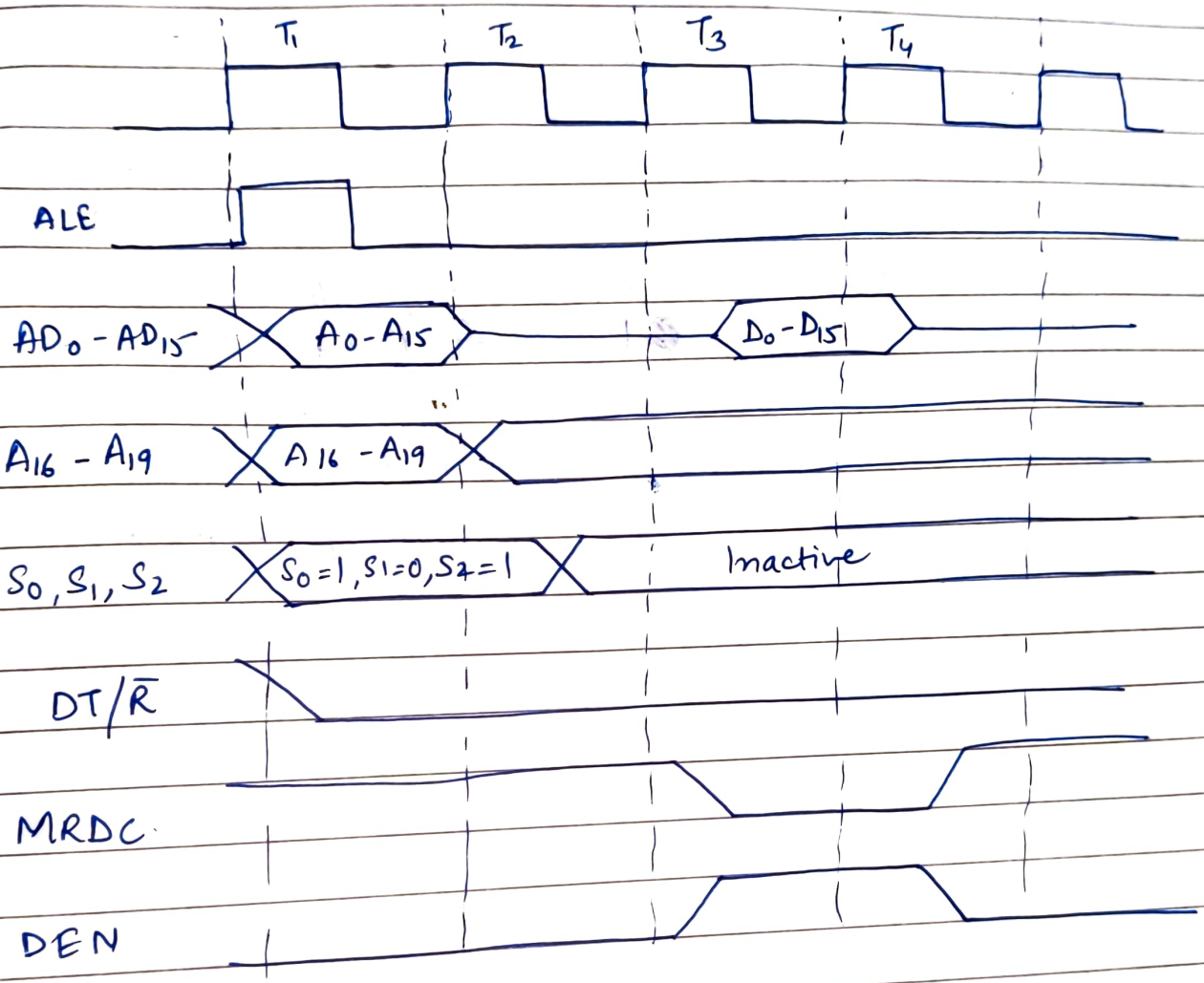

after execution




(d)

# Timing Diagram of 8086

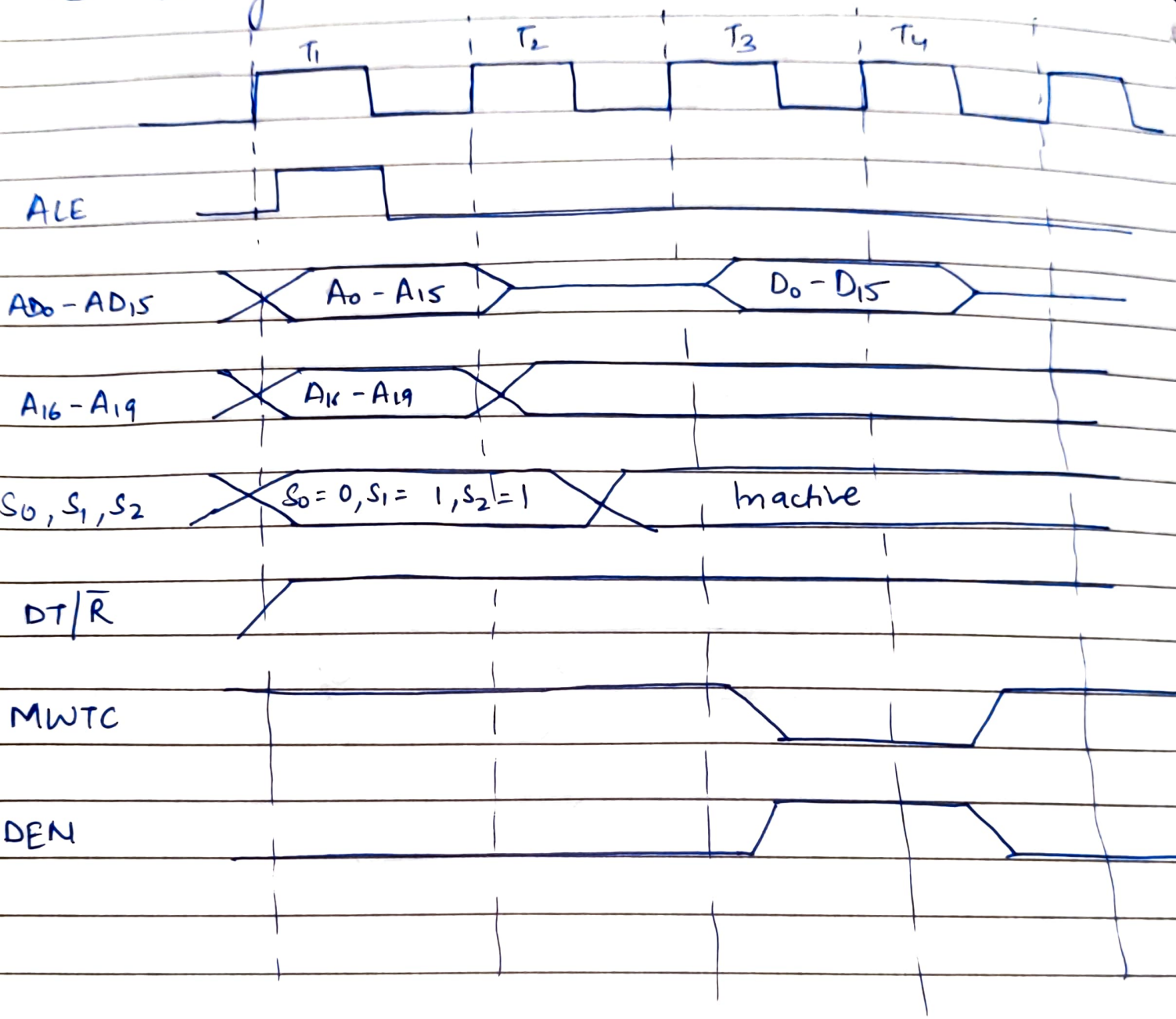
① Memory Read operation in Maximum Mode.



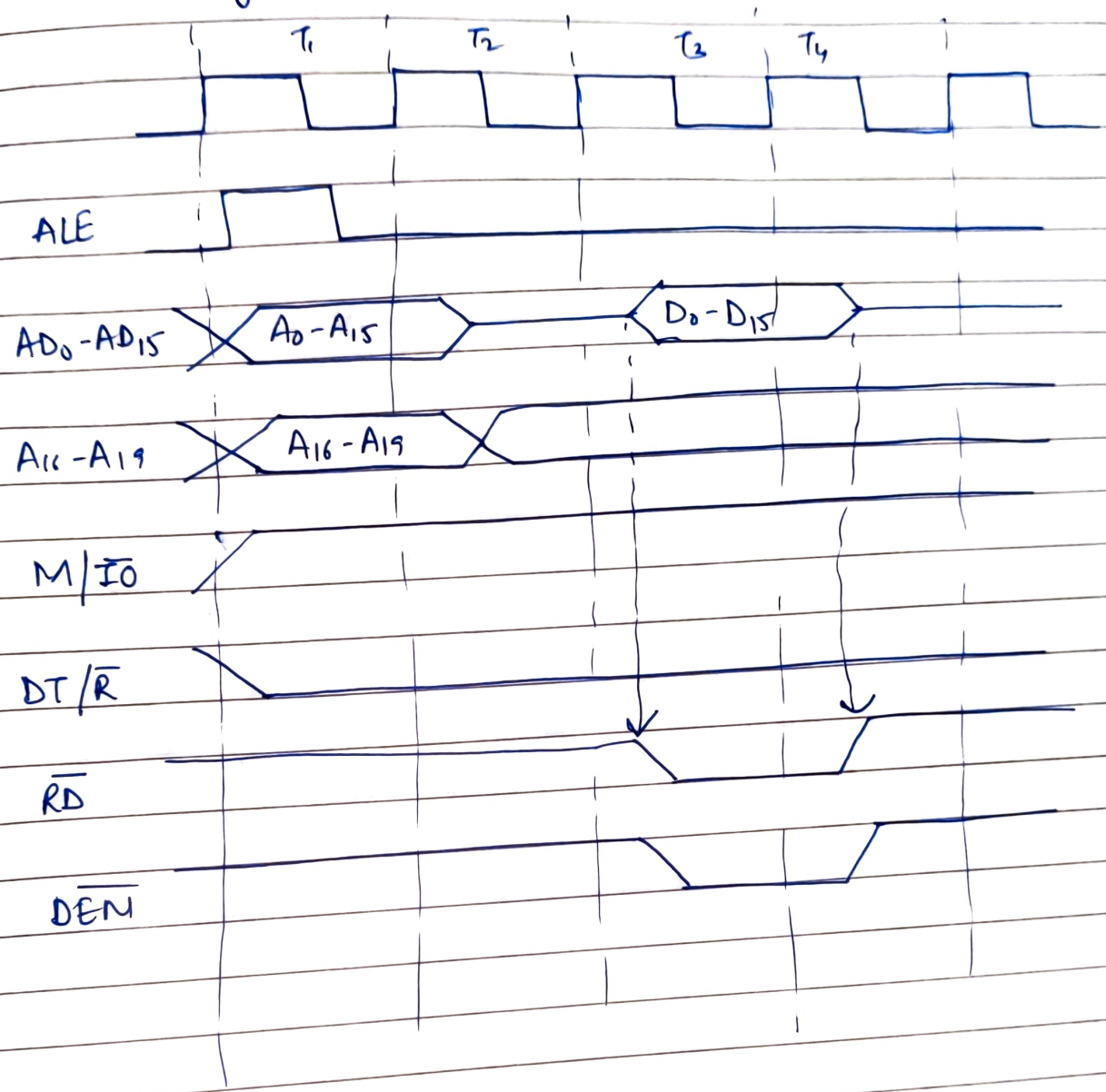

X why this signal, because it is a combination of 0 and 1.

$S_0 = 1, S_1 = 0, S_2 = 1$ why.
This is used for memory read operation.

## (2) Memory write operation in maximum mode.



## ③ Memory Read in minimum mode -



## (4) Memory write in minimum mode