Unit -4

*) Logic and Program control Instruction :-

a) Basic Logic instructions :

The logical instructions in the 8086 include AND, OR, XOR, NOT & TEST.

i) AND :-

The AND instruction performs a logical AND operation between the corresponding bits in the source & destination and stores the result in the destination. The source and destination can be either bytes (or) words. The general form of the AND instruction is AND destination, source.

AND destination, source.

eg:-

i) AND AH, CH

```
AH => F0 = 1111 0000
CH => 9E = 1001 1110
           ---------
           1001 0000
           ---------
      AH = 9     0
```

ii) AND AH, 05

```
AH => F0 = 1111 0000
     05 = 0000 0101
          ---------
          0000 0000
          ---------
     AH = 0     0
```

iii) AND AH, [0002].

```
AH => F0 = 1111 0000
     0A = 0000 1010
          ---------
          0000 0000
          ---------
     AH =  0    0
```

OR instruction :-

The OR instruction performs a logical OR operation between the corresponding bits in the source and destination and stores the result in the destination. The source & destination can be either bytes (or) words. The general form of OR instruction is

OR destination, source

The rules for the source & destination and the way flags are affected are the same as the AND instruction

eg:-

i) OR AH, DH

AH ⇒ F0 H = 1111 0000
DH ⇒ 0E H = 0000 1110
1111 1110
AH = F E

ii) OR AH, 05

AH ⇒ F0H = 1111 0000
05H = 0000 0101
1111 0101
AH = F 5

XOR instruction:-

The XOR instruction performs a logical XOR operation between the corresponding bits in the source and destination and stores the result in the destination.

The source and destination can be either byte (or) words. The general form of the XOR instruction is

XOR destination, source.

eg:-

XOR BL, 05

BL = F0 ⇒ 1111 0000
05 ⇒ 0000 0101
1111 0101
BL = F 5

NOT instruction:-

The NOT instruction inverts each bit (perform is 1's complement) of the byte (or) word at a specified destination. The destination can be a register (or) a memory location. The NOT instruction does not affect any flags.

eg:-

i) NOT BH BH ⇒ F0H

$$\begin{array}{cc} 1111 & 0000 \\ \hline 0000 & 1111 \\ \hline 0 & F \end{array}$$

$\boxed{BH = 0FH}$

ii) NOT BX : Take's 1's complement of BX.

iii) NOT [SI] : Take 1's complement of data in the memory at [SI].

TEST instruction:-

The instruction ANDs content of a source byte (or) word with the content of the specified destination byte (or) word. The flags are updated but neither operand is changed.

The TEST instruction is often used to set flags before a conditional jump instruction.

The general form of Test instruction is

TEST destination, source

It sets zero flag, sign flag & parity flags according to the result.

eg:-

i) TEST AH, CH

AH ⇒ F0 = 1111 0000
CH ⇒ 9E = 1001 1110

$$\begin{array}{cc} 1001 & 0000 \\ \hline 9 & 0 \end{array}$$

Sign flag = 1
zero flag = 0
Parity flag = 1 (even)

ii) TEST AL, 01H

AL ⟹ 01H ⟹ 0000 0001

01 ⟹ 01H ⟹ 0000 0001

0000 0001

0 1

SF = 0

ZF = 0 [Non-zero]

PF = 0 (odd).

Shift and Rotate :-

The shift/rotate instructions perform logical left-shift and right-shift, and arithmetic left-shift and right-shift operations.

The arithmetic-left shift (SAL) and logical left shift (SHL) have the same function,

i) SAL/SHL :-

The general format of SAL/SHL instruction is SAL/SHL destination, count.

The destination can be a register (or) a memory location and a byte (or) a word.

This instruction shifts each bit in the destination a specified number of bit positions to the left.

As a bit is shifted out of the LSB position, a 0 is placed in the LSB position.

The MSB is shifted to the carry flag (CF).

Eg:-

SHL CL, 2.

If CL = 23.

```
Cy0    0 0 1 0 0 0 1 1 ← 0
0      0 1 0 0 0 1 1 0
             8     C
```

$\boxed{CL = 8C}$

Eg:- MOV CL, 05
SAL AX, CL [Shift the left the content of AX by five bit].

CF ← MSB ← LSB ← 0.

ii) SAR:-

The general format of the SAR instruction is SAR destination, Count.

The destination can be a register (or) a memory location and a byte (or) a word.

This Instruction shifts each bit in the destination a specified number of bit positions to the right.

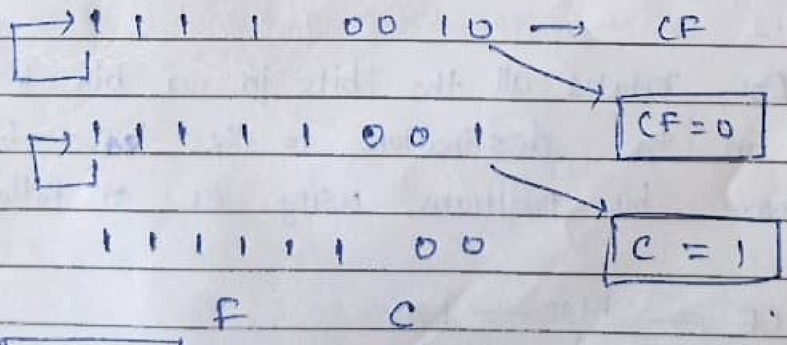
As a bit is shifted out of the MSB position, a copy of the old MSB is put in the MSB position.

The LSB will be shifted into the carry flag as follows.

MSB → MSB →, LSB → CF.

Eg:-

SAR DL, 02 DL = F2



$\boxed{DL = FC}$

iii) SHR :-

The general format of the SHR instruction is SHR destination, ~~source~~ count.

The destination can be a register (or) a memory location and a byte (or) a word.

This instruction shift each bit in the destination a specified number of bit positions to the right.

As a bit is shifted out of the MSB position. a 0 is placed in the MSB position.

The LSB is shifted in to the Carry flag (CF)

$$\begin{tikzcd} 0 \arrow[r] & MSB \arrow[r] & LSB \arrow[r] & CF \end{tikzcd}$$

SHR CL, 2

If CL = 23

```
0 ⟶ 0 0 1 0  0 0 1 1
     0 0 0 1  0 0 0 1   | C= |
           1      1
```

CL = 11

Rotate instruction :-

i) ROL :-

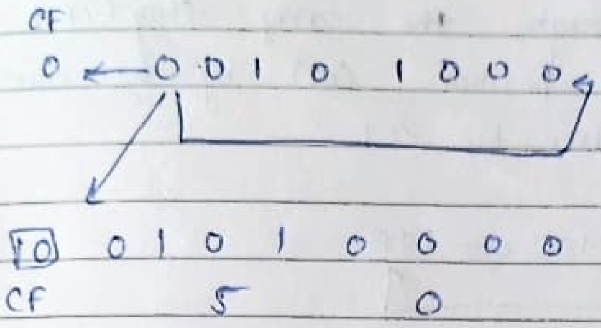
ROL rotates all the bits in a bits in a byte (or) word in the destination to the left, by one (or) more bit positions, using CL as follows.

$$\begin{tikzcd} CF & MSB \arrow[l] \arrow[r, bend right] & LSB \arrow[l] \end{tikzcd}$$

The data bit moved out of the MSB is copied into CF. ROL affects only CF.

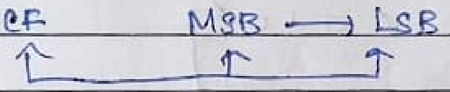
Eg:-

ROL CH, 1 $\boxed{CH=28}$



ii) ROR :-

This instruction rotates all the bits of the specified byte (or) word by a specified number of bit positions to the right. The operation done when ROR is executed.



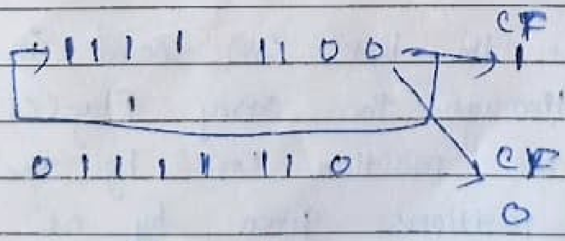
The general format of ROR instruction is
ROR destination, count.

The data bit moved out of the LSB is copied into CF.

CF contains the bit most recently rotated out of the LSB, in the case of a multiple bit rotate operation.

Eg:-

ROR AL, 1 $\boxed{AL = FC}$



7 E ⇒ AL.

iii) RCL :-

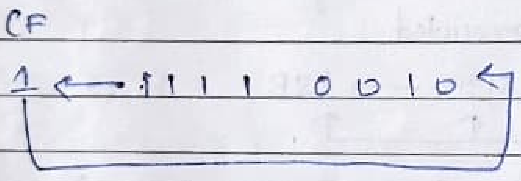
RCL rotates the byte (or) word in the destination left through the carry flag (CF) either by one bit position (or) by the number of bit position given by CL

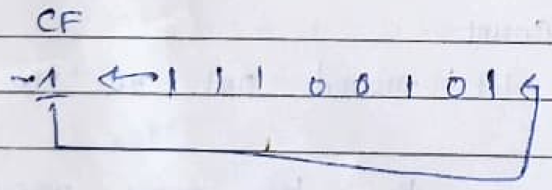
$$\begin{tikzcd} CF & MSB \arrow[l] & LSB \arrow[l] \arrow[from=1-1, to=1-3, bend right] \end{tikzcd}$$

The flags affected are the same as those affected during the execution of ROL.

Eg:-

RCL AH, 2 $\boxed{AH = F2}$





CF

1   1 1 0 0 1 0 1 1

C   B

iv) RCR :-

RCR rotates the byte (or) word in the destination right, through the carry flag (CF), either by one bit position (or) by the number of bit positions given by CL.

$$\begin{tikzcd} CF \arrow[r] & MSB \arrow[r] & LSB \arrow[to=1-1, bend left] \end{tikzcd}$$

iii) RCL :-

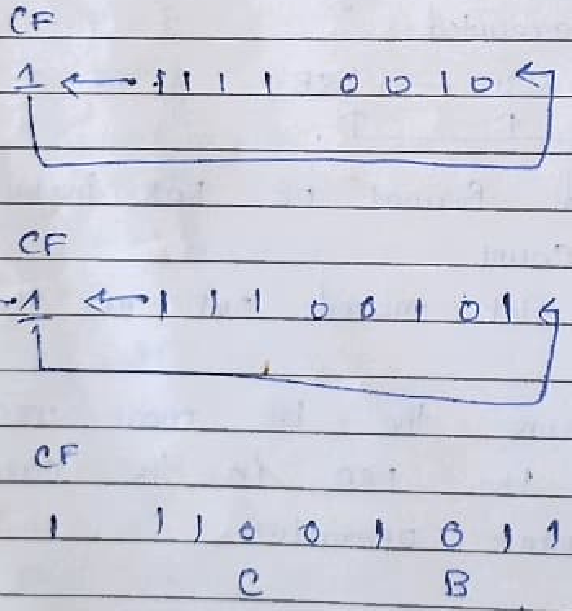
RCL rotates the byte (or) word in the destination left through the carry flag (CF) either by one bit position (or) by the number of bit position given by CL

$$\begin{tikzcd} CF & MSB \arrow[l] & LSB \arrow[l] \\ {} \arrow[u, dash] \arrow[rr] & & {} \arrow[u] \end{tikzcd}$$

The flags affected are the same as those affected during the execution of ROL.

Eg :-

RCL AH, 2 $\boxed{AH = F2}$



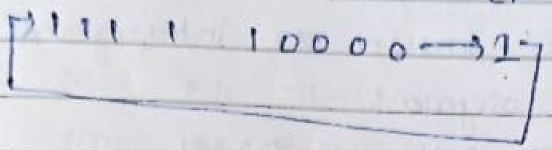
iv) RCR :-

RCR rotates the byte (or) word in the destination right, through the carry Flag (CF), either by one bit position (or) by the number of bit positions given by CL.

$$\begin{tikzcd} CF \arrow[r] & MSB \arrow[r] & LSB \arrow[d, dash] \\ {} \arrow[u] & & {} \arrow[ll] \end{tikzcd}$$

Eg:- RCR CL, 1 $\boxed{CL = F8}$

CF



1111 1 1 000 CF

F 8 0

Jump group and Procedure instructions :-

It is the act of switching execution to a different sequence.

There are two types in jump group instructions

→ unconditional jump

→ conditional jump.

Jump unconditionally :-

The program sequence is transferred to the memory address given in the operand.

JMP address

eg :-

```
         MOV  AH, 20H
         MOV  BL, 30H
         JMP Next
         ADD  BL, AH
         SUB  AL, CH
         .
         .
         .
Next     AND  AL, BL
         RCR  AH, 2
         HLT.
```

There are three types in unconditional jump

i) NEAR :-

Jump to a instruction, within the current code segment also known as intrasegment jump.

```
code segment
MOV  AH, 20H        02301
MOV  BL, 30H        02302
RCR  AH, 2          02303
ADD  BL, AH         02304
JMP  NEXT           02305
ROL  AL, 1          02306.
```

ii) Short :-

Jump to a instruction range limited to -128 to +127 from the current address

Total 256 bytes.

iii) FAR :-

Jump to an instruction located in a different segment reg.

Conditional Jump :-

i) JC address

Jump if carry flag is set.

$\boxed{C=1}$

```
              MOV  AH, 80H          80 = 1000 0000
              ADD  AH, 81H          81 = 1000 0001
                                    ---------------
              JC   NEXT'            [1] 0000 0001
True   False  ADD  BL, AH           carry flag
              SUB  AL, CH
              ROL  AL, 1
Next        · AND  AL, BL
              RCR  AH, 2
              HLT.
```

ii) JNC (Jump if no carry) :-

JNC address $\boxed{C=0}$

```
        MOV  AH, 71H          71 = 0111  0001
        ADD  AH, 71H               0111  0001
        JNC  NEXT.                 ----------
         -                         1110  0010
         -                        carry flag = 0
         -
NEXT        AND  AH, 1BH
            HLT.
```

iii) JZ / JE :- Jump if zero $\boxed{Z=1}$

```
            MOV  AL, F0H          1111  0000
            MOV  BL, F0H          1111  0000
            CMP  AL, BL.          ----------
            JZ   NEXT             0000  0000
             -                    ----------
             -                      Z=1
             -
    NEXT     -
             -
            HLT.
```

JNZ / JNE	Jump if not zero	$\boxed{Z=0}$
JPO / JNP	Jump if parity is odd	$\boxed{P=0}$
JP / JPE	Jump if parity is even	$\boxed{P=1}$
JS	Jump if sign (negative)	$\boxed{S=1}$
JNS	Jump if no sign (positive)	$\boxed{S=0}$
JO	Jump if overflow	$\boxed{O=1}$
JNO	Jump if not overflow	$\boxed{O=0}$

JA   Z=0 & C=0   Jump if above.

```
MOV AL, 35H
MOV BL, 25H
CMP AL, BL
JA address    [AL is above than BL]
HLT
```

JAE :- Jump to address if above (or) equal
$\boxed{CF=0}$

JBE = Jump to address if below (or) equal
CF=1 & ZF=1

JB = Jump if below
$\boxed{C=1}$

JG : Jump to address If greater than second number

JGE : Jump to address if greater (or) equal

JL : Jump to address if first number is Lesser than second.

JLE :- Jump to address if the number is lesser (or) equal.

Loop instruction :-
It is used to loop a group of instruction till zero flag is set that is
$\boxed{Z=1}$. Cx register consist of Count.

LOOPNE / LOOPNZ.
• Loop a group of instruction till the zero flag is not set.
$\boxed{ZF=0}$

*) Machine (or) Processor Control instructions:-

The machine (or) processor control instruction in the 8086 include

HLT, LOCK, NOP, ESC & WAIT.

HLT:-

The HLT instruction stops the execution of all instructions and places the processor in the halt state. An interrupt (or) a Reset signal causes the processor to resume execution from the halt state

LOCK:-

The lock instruction provides the processor an exclusive hold on the use of the system bus.

It activates an external locking signal ($\overline{LOCK}$) of the processor and is placed as a prefix to the instruction for which a lock is to be asserted.

The lock functions only with the XCHG, ADD, OR, ADC, SBB, AND, SUB, XOR, NOT, INC, DEC instructions

NOP :-

No operation :- This instruction is used to insert a delay in software delay programs

ESC:-

This instruction is used to pass instructions to a coprocessor such as 8087, which shares the address bus and data bus with an 8086.

As the 8086 fetches instruction byte from memory, the coprocessor catches these bytes from the data bus and puts them in a queue.

However the coprocessor treats all the normal 8086 instruction as NOP instructions.

When the 8086 fetches an ESC instruction, the coprocessor decodes the instruction and carries out the action specified in the instruction.

v) WAIT :- when this instruction is executed, the 8086 check the status of its $\overline{TEST}$ input pin and is the $\overline{TEST}$ input pin is high, it enters an idle condition during which it does not do any processing.

The 8086 remains in this state until the 8086's $\overline{TEST}$ input is made low (or) an interrupt signal is received on the INTR.

If a valid interrupt occurs, while 8086 is in idle state, it returns to the idle state after the interrupt service routine is executed.

The $\overline{WAIT}$ Instruction does not affect flag.

*) Basic Interrupt processing :-

An interrupt is a condition that halts the microprocessor temporarily to work on a different task and then return to its previous task.

If an interrupt has been requested, the 8086 processes it by performing the series of steps.

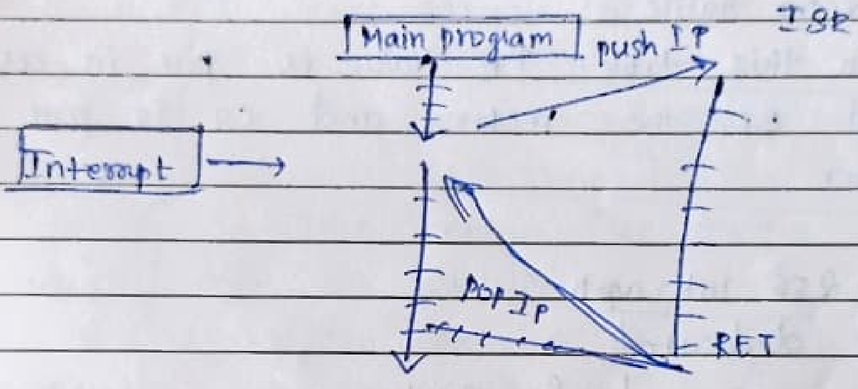
i) Pushes the content of the flag register on to the stack to preserve the status of the interrupt (IF) by decrementing the stack pointer by 2.

ii) Disables the INTR interrupt by clearing IF in the Flag register

iii) Resets TF in the flag register, to disable the single step

iv) Pushes the content of the code segment (CS)

register onto the stack by decrementing SP by 2.
v) Pushes the content of the instruction pointer (IP) onto the stack by decrementing SP by 2.
vi) Perform an indirect jump to the start of the interrupt service routine (ISR) corresponding to the received interrupt.



There are total 256 interrupts in 8086.

When the 8086 responds to an interrupt, it refers to four memory locations present in the Interrupt vector table (IVT), to get the new values of CS and IP.

These memory locations are used to find the starting address of the ISR of the received interrupt in the memory.

In an 8086 system, the first 1KB of memory from the addresses 00000H - 003FFH is set aside as a table called interrupt vector table (IVT) for storing the interrupt vector.

Each interrupt vector indicates the starting address of the ISR of a particular interrupt in the memory. It contain four bytes in which the lower two bytes are called offset and upper two bytes are called segment.

The offset part of the interrupt vector is loaded in the IP register and segment part is loaded in the CS register.

The starting address of an ISR is often called the interrupt vector (or) ~~vect~~ Interrupt pointer. Therefore the table is reffered as interrupt vector table.

In this table, IP value is put in as low word of the vector and CS is put in high vector.

256 interrupts

0-4 dedicated

5-31 reserved for system use

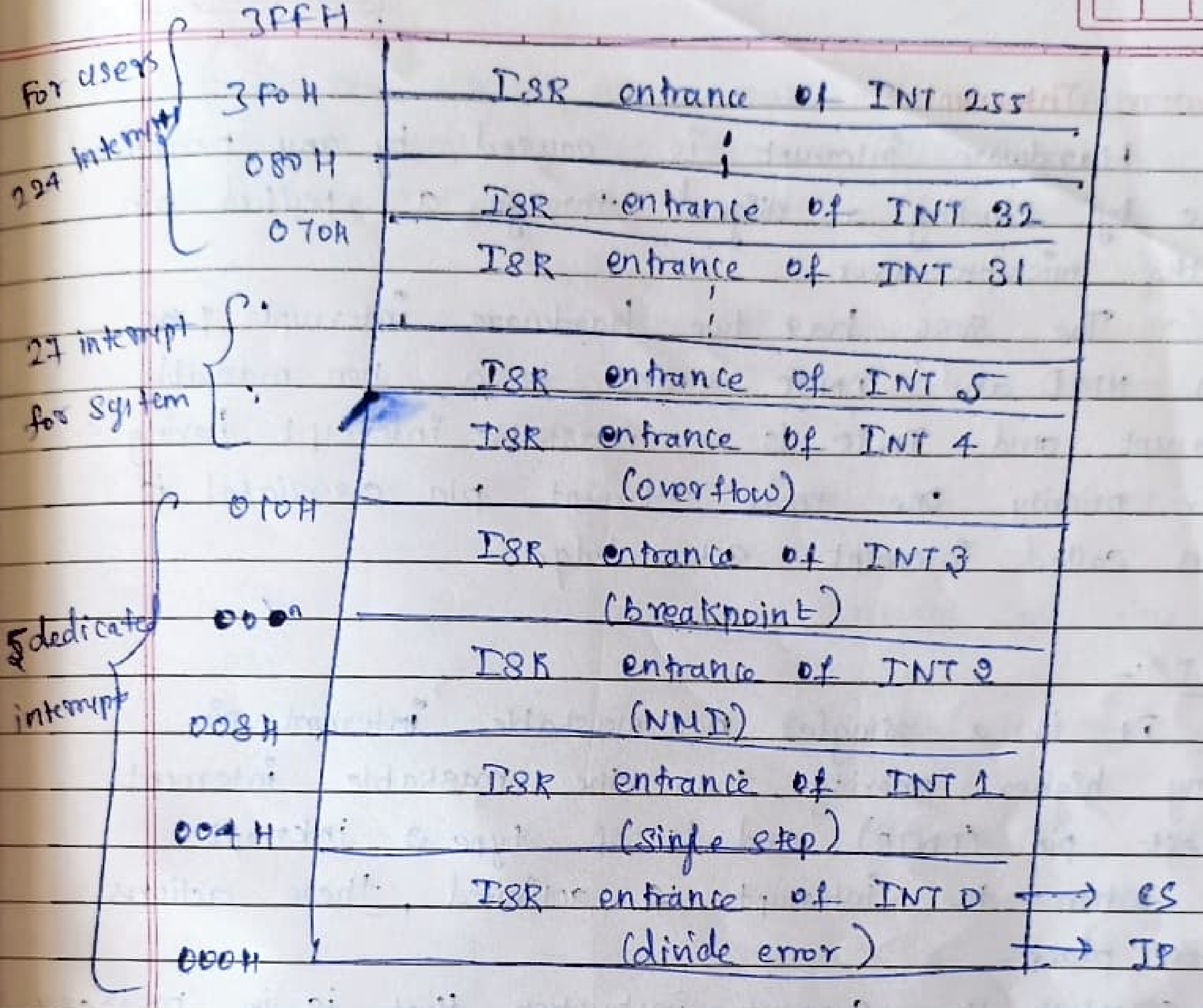
↳ 08H - 0FH : 8259A

↳ 10H - 1FH : BIOS

32 - 255 reserved for users.

↳ 20H - 3FH : DOS

↳ 40H - FFH : Open



In general there are two types of interrupts.

$$\begin{tikzcd}
 & \boxed{\text{Interrupts}} \arrow[dl] \arrow[dr] & \\
\boxed{\text{Hardware interrupts}} \arrow[d] \arrow[dr] & & \boxed{\text{Software interrupts}} \arrow[d] \\
\boxed{\text{Maskable interrupts}} & \boxed{\text{Non-maskable interrupts}} & \boxed{\text{256-Types of Software interrupt}}
\end{tikzcd}$$

DATE

*) Hardware Interrupts :-

Hardware interrupt : is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

The 8086 has two hardware interrupt pins (i.e) NMI and INTR. NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

NMI :-

It is a single non-maskable interrupt pin having higher priority than the maskable interrupt request pin (INTR) and is of type 2 interrupt:

When the interrupt is activated, these actions takes place.

→ Complete the current instruction that is in progress
→ Pushes the flag register values on to the stack
→ Pushes the CS (code segment) value and IP value of the return address to the stack.
→ IP is loaded from the content of the word location
→ CS is loaded from the content of the next word location
→ Interrupt flag and Trap flag are set to 0.

INTR :-

The INTR is a maskable interrupt because the microprocessor only if will be interrupted only if interrupts are enabled using set interrupt flag instructions.

These are the action taken by the microprocessor
→ First completes the current instruction
→ Activate INTA output and receives the interrupt type, say X.
→ Flag registers value, CS value of the return address and IP value of the return address are pushed on to the stack.
→ IP value is loaded from the content of word location
→ CS is loaded from the contents of the next word location
→ Interrupt flag and trap flag is reset to 0.

The processor has the facility for accepting (or) rejecting hardware interrupts. Programming the processor to reject an interrupt is refered to as masking (or) disabling and programming the processor to an interrupt is refered to as unmasking (or) enabling.

In 8086 the interrupt flag (IF) can be set to one to unmask (or) enable all hardware interrupts and IF is cleared to zero (or) mask (or) disable a hardware interrupt except NMI

The interrupts whose request can be either accepted (or) rejected by the microprocessor are called as maskable interrupt. INTR is maskable interrupt.

The interrupt whose request has to be definitely accepted (or) cannot be rejected by the processor are called non-maskable interrupt

Whenever a request is made by non-maskable interrupt, the processor has to definitely accept that request and service that interrupt by suspending its current program and executing an ISR.

The interrupt initiated through NMI pin and all software interrupts are non-maskable.