Flag Operations

STC – Set Carry Flag

Example: STC

This instruction is used to set the carry flag.

Mnemonic: **STC**

Operation : **CF = 1**

Addressing Mode : Implied addressing mode

Flags : Except carry flag no other flags are affected.

CLC – Clear Carry Flag

Example: CLC

This instruction is used to clear the carry flag.

Mnemonic: **CLC**

Operation : **CF = 0**

Addressing Mode : Implied addressing mode

Flags : Except carry flag no other flags are affected.

CMC – Complement Carry Flag

Example: CMC

This instruction is used to complement the carry flag.

Mnemonic: **CMC**

Operation : $CF = \overline{CF}$

If CF = 0 then after execution of CMC instruction CF = 1

If CF = 1 then after execution of CMC instruction CF = 0

Addressing Mode : Implied addressing mode

Flags : Except carry flag no other flags are affected.

STD – Set Direction Flag

- This instruction is used to set the direction flag.
- DF flag is used in string instruction.
- If DF= 1 then in case of string instructions SI and DI automatically decremented.

Example: STD

Mnemonic: **STD**

Operation : **DF = 1**

1

U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

Addressing Mode : Implied addressing mode

Flags : Except Direction flag no other flags are affected.

CTD – Clear Direction Flag

- This instruction is used to clear the direction flag.
- DF flag is used in string instruction.
- If DF= 0 then in case of string instructions SI and DI automatically incremented.

Example: CTD

Mnemonic: **CTD**

Operation : **DF = 0**

0

U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

Addressing Mode : Implied addressing mode

Flags : Except Direction flag no other flags are affected.

STI – Set Interrupt Enable Flag

- This instruction sets the interrupt flag to 1.
- This enables INTR interrupt of the 8086.

Example: STI

Mnemonic: **STI**

Operation : **IF = 1**

1

U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

CLI – Clear Interrupt Enable Flag

- This instruction resets the interrupt flag to zero.
- If the interrupt flag is reset, the 8086 will not respond to an interrupt signal on its INTR input.

Example: STI

Mnemonic: **CTI**

Operation : **IF = 0**

0

U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

Addressing Mode : Implied addressing mode

Flags : Except Interrupt flag no other flags are affected.

No Operations

NOP – No Operation

Example: NOP

- The execution of this instruction causes the CPU to do nothing.
- This instruction uses three clock cycles and increments the instruction pointer to point to the next instruction.
- It can be used to increase the delay of delay loop.

Mnemonic: **NOP**

Operation : **Do nothing**

Addressing Mode : Implied addressing mode

Flags : Dose not affect any flag.

External synchronization

HALT – Halt until interrupt or reset

- The HLT instruction will cause the 8086 to stop fetching and executing instructions. The 8086 enters into a halt state. To come out of the halt state, there are 3 ways given below
  (i) Interrupt signal on INTR pin
  (ii) Interrupt signal on NMI pin
  (iii) Reset signal on reset pin.
- It may be used as an alternative to an endless software loop in situations where a program must wait for an interrupt

Mnemonic: **HLT**

Addressing Mode : Implied addressing mode

Flags : Dose not affect any flag.

WAIT – Wait for test pin active

When this instruction executes, the 8086 enters on idle condition in which it is doing no processing.

- The 8086 will stay in this idle state until 8086 $\overline{\text{TEST}}$ input pin is made low or on interrupt signal is received on the INTR or NMI interrupt pins.
- If a valid interrupt occurs while the 8086 is in the idle state, the 8086 will return to idle state after the interrupt service procedure executes.
- It is used to synchronize the 8086 with external hardware. Such as 8087 math processor.

Mnemonic: **WAIT**

Addressing Mode : Implied addressing mode

Flags : Dose not affect any flag.

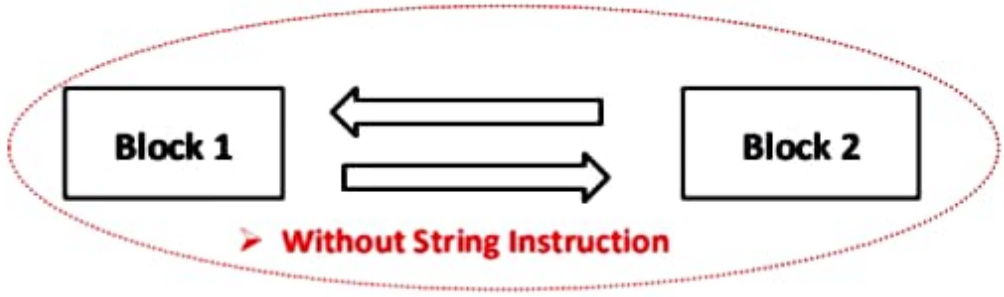
3. ESC – Escape to external processor

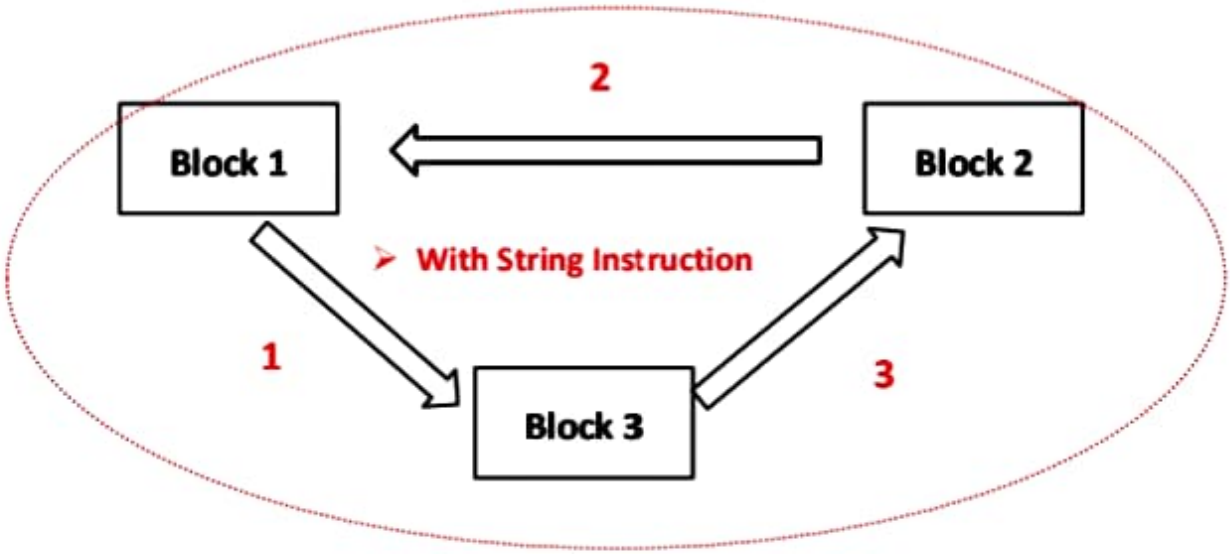
Mnemonic	ESC external – opcode, source.
Operation	This instruction is used to pass instruction to a coprocessor, such as 8087 math co-processor which shares the address and data bus with on 8086. • The instruction for the Coprocessor are represented by a 6 bit code embedded in the escape instruction. • When the 8086 fetches on ESC instruction, the coprocessor decodes the instruction and carries out the action specified by the 6 bit code specified in the instruction. • In most cases 8086 treats the ESC instruction as a NOP in some cases 8086 will access a data item in memory for co-processor.

4. LOCK – Lock bus during next instruction

Mnemonic	LOCK
Operation	Many multiprocessor systems contain several microprocessors. Each microprocessor has its own local buses and memory. The individual microprocessors are connected together by a shared system bus so that each can access system resources such as disk drives or memory. • Each microprocessor takes control of the system bus only when it needs to access some resource. • Lock prefix allows a microprocessor to make sure that another processor does not take control of the system bus. • While it is in the middle of a critical instruction which uses the system bus when an instruction with lock prefix executes the 8086 will assert its bus lock signal output. This signal is connected to an external bus controller, which then prevents any other processor from taking over the system bus.
Example	LOCK XCHG SEMAPHORE, AL. The XCHG instruction requires two bus accesses. The lock prefix prevents another processor from taking control of system bus between two accesses.

Block Exchange





➢ Without String Instruction

➢ With String Instruction

Block Exchange without string instruction

WAP to exchange 05 data bytes present at two memory blocks 1000 and 2000 onwards respectively without using string instructions

Block 1

SI →	1000	01
SI+1 →	1001	02
	1003	03
	1004	04
	1005	05

Block 2

DI →	2000	05
DI+1 →	2001	04
	2003	03
	2004	02
	2005	01

[SI] = 01 & [DI] = 05

Mov content of SI in AL & content of DI in AH

AL = 01 & AH = 05

Exchange the content of AL and AH

AL = 05 & AH = 01

Store the exchanged value into the blocks

Increment SI and and DI by 1 to point next memory location

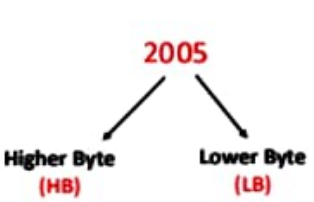
1. Immediate addressing mode (Data in Instruction)

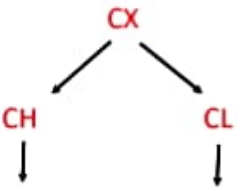
In immediate addressing mode the data to be used is immediately given in the instruction.

Example :

MOV CL, 02 H ⟶ 02 (8 bit data) is transfer into reg. CL

MOV CX, 2005 H ⟶ 2005 (16 bit data) is transfer into reg. CX in following manner :





2. Register addressing mode (Data in register)

In register addressing mode data to be operand is in general purpose register

Example :

MOV CL, BL ⟶ Content (8 bit data) of reg. BL is transferred into reg. CL

MOV CX, BX ⟶ Content (16 bit data) of reg. BX is transferred into reg. CX

MOV CL, BX ✗ It is not possible to perform 16 bit operation with 8 bit reg or vice versa.

3. Direct addressing mode (Address in Instruction)

In direct addressing mode operand is given by a direct address where the data is present.

Anything in [ ] refers address

Example 1 :

MOV CL, [2000] ⟶ The Content which present at memory location 2000 transfer into reg. CL

CL ⟵ DS : [2000]

DS

0000	1A
0001	08
.	
.	
2000	04
2001	05
2002	06
.	
FFFF	

⟶ CL = 04

Note :

- Data is always refer from data segment (DS)
- DS has starting address
- From 0000 to FFFF these are the offset
- BIU section of 8086 generate 20 bit physical address using following formula :

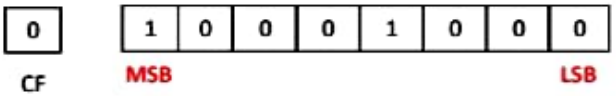
PA = Seg address * 10 h + offset

1. SHL/SAL : Shift bits to left

Example: MOV BL, 88 h
MOV CL,01h
SHL BL, CL

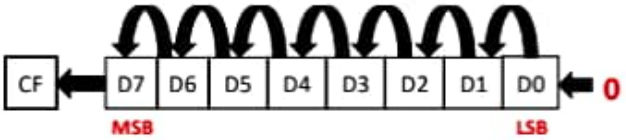
1. This instruction is used to shift 8 bits and 16 bits to the left.
2. MSB shifted into the carry.
3. LSB gets a 0.

Before Execution BL= 88 h

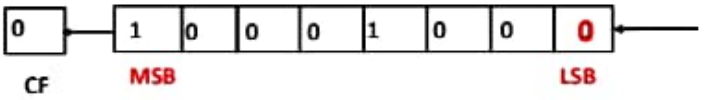


Mnemonic: **SHL/SAL destination , Count**

Operation :



After Execution BL= 10 h

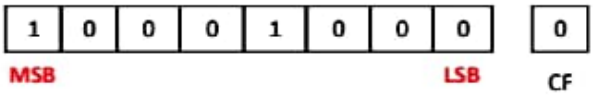


2. SHR : Shift bits to right

Example: MOVE CL, 01h
SHR BL, CL

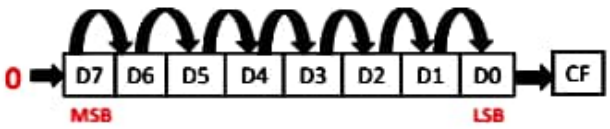
1. This instruction is used to shift 8 bits and 16 bits to the right.
2. LSB shifted into carry flag.
3. MSB gets 0.

Before Execution BL= 88 h

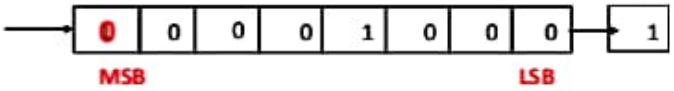


Mnemonic: **SHR destination , Count**

Operation :



After Execution BL= 44 h



Hence this instruction numbers are assumed to be unsigned

By putting 0 in MSB, going to change the sign of number i.e. makes the no. +ve

If we don't want to change the sign of number which means if +ve then +ve and if –ve then –ve (signed numbers) then use next shift instruction.

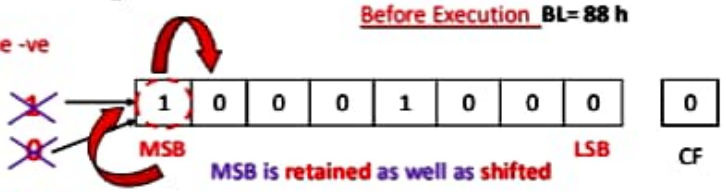
3. SAR : Shift Arithmetic right

Example: MOVE CL, 01h
SHR BL, CL

1. This instruction is used to shift 8 bits and 16 bits to the right.
2. LSB shifted into carry flag.
3. Retain and shift the MSB

Number will become -ve

Before Execution BL= 88 h



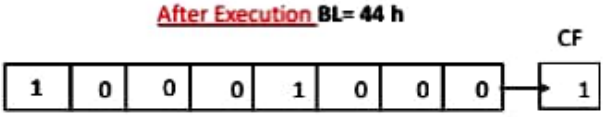
MSB is retained as well as shifted

Mnemonic: **SAR destination , Count**

Number will become +ve

Operation :

After Execution BL= 44 h



1. NOT : Destination

This instruction forms the 1's complement of destination and result stores into destination

Mnemonic: **NOT destination**

Operation :

Destination ⟵ $\overline{\text{Destination}}$

Example: MOVE CL, 01h
NOT CL

Before Execution **CL= 88 h**

1	0	0	0	1	0	0	0

After Execution **CL= 77 h**

0	1	1	1	0	1	1	1

2. AND : Destination , Source

This instruction is used to logically AND's the content of source with destination and result will store into the destination

Mnemonic: **AND destination , Source**

Operation :

Destination ⟵ **Destination** ∧ **Source**

Example: AND BL, CL

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Anything AND with 0 will become 0

Anything AND with 1 will remains same

MOV CL, 35 h
AND CL, F0 h

0	0	1	1	0	1	0	1
1	1	1	1	0	0	0	0
0	0	1	1	0	0	0	0

Clear Lower Nibble

NOTE : data is always starts with number
MOV CL, 35 h
AND CL, 0F0 h

2. OR : Destination , Source

This instruction is used to logically OR's the content of source with destination and result will store into the destination

Mnemonic: **OR destination , Source**

Operation :

Destination ⟵ **Destination** ∨ **Source**

Example: OR BL, CL

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Anything OR with 0 will remains same

Anything AND with 1 will become 1

MOV CL, 35 h
OR CL, 0F h

0	0	1	1	0	1	0	1
0	0	0	0	1	1	1	1
0	0	1	1	1	1	1	1

Set Lower Nibble

2. XOR : Destination , Source

This instruction is used to logically OR's the content of source with destination and result will store into the destination

Mnemonic: **XOR destination , Source**

Operation :

Destination ⟵ **Destination** ⊕ **Source**

Example: OR BL, CL

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

Anything XOR with 0 will remains same

Anything XOR with 1 will give complement

MOV CL, 35 h
XOR CL, 0F h

0	0	1	1	0	1	0	1
0	0	0	0	1	1	1	1
0	0	1	1	1	0	1	0

Complement Lower Nibble

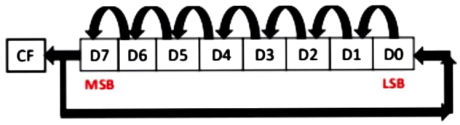
1. ROL : Rotate bits to left

Example: MOV CL, 01h
ROL BL, CL

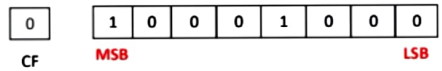
1. This instruction is used to rotate 8 bits and 16 bits to the left.
2. MSB moves into the LSB
3. MSB also copies into the carry flag

Mnemonic: **ROL destination , Count**

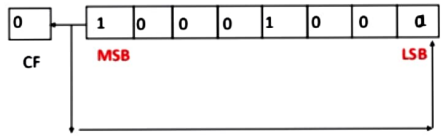
Operation :



Before Execution **BL= 88 h**



After Execution **BL= 11 h**



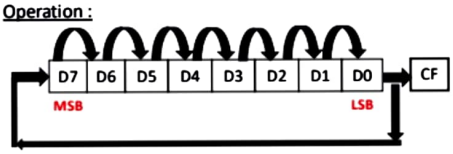
2. ROR : Rotate bits to right

Example: MOV CL, 01h
ROR BL, CL

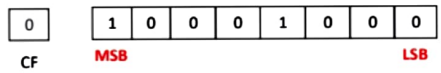
1. This instruction is used to rotate 8 bits and 16 bits to the right.
2. LSB moves into the MSB
3. LSB also copies into the carry flag

Mnemonic: **ROR destination , Count**

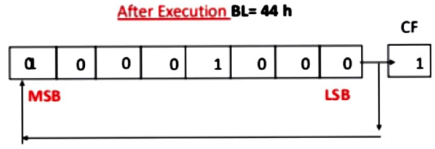
Operation :



Before Execution **BL= 88 h**



After Execution **BL= 44 h**



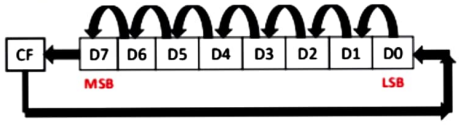
3. RCL : Rotate bits to left with carry

Example: MOVE CL, 01h
RCL BL, CL

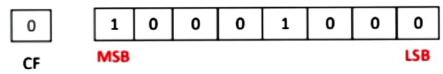
1. This instruction is used to rotate 8 bits and 16 bits to the left along with carry
2. LSB moves into the carry flag
3. Previous carry flag moves into the LSB

Mnemonic: **RCL destination , Count**

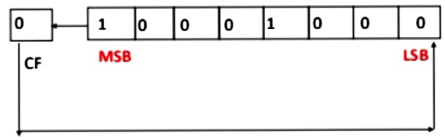
Operation :



Before Execution **BL= 88 h**



After Execution **BL= 10 h**



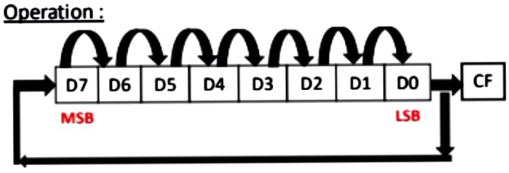
4. RCR : Rotate bits to right with carry

Example: MOVE CL, 01h
ROR BL, CL

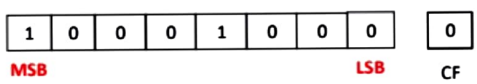
1. This instruction is used to rotate 8 bits and 16 bits to the right.
2. LSB moves into the MSB
3. LSB also copies into the carry flag

Mnemonic: **RCR destination , Count**

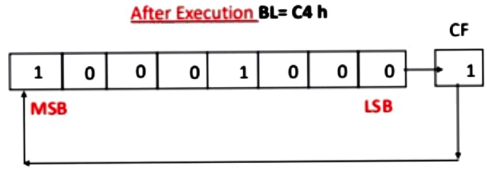
Operation :



Before Execution **BL= 88 h**



After Execution **BL= C4 h**



Prefix used with string instructions:

1. **REP – Repeat**
2. **REPE – Repeat if equal**
3. **REPNE – Repeat if not equal**

MOVS B/W/D :

This instruction is used to transfer the contents of source to destination.

Operation : ES: [DI] ← DS: [SI]

Types :

1. **MOVSB : Moves 8 bit content of source to destination : SI = SI +1 and DI = DI + 1 if DF =0**
   SI = SI -1 and DI = DI - 1 if DF =1
2. **MOVSW : Moves 16 bit content of source to destination : SI = SI +2 and DI = DI + 2 if DF =0**
   SI = SI - 2 and DI = DI - 2 if DF =1
3. **MOVSD : Moves 32 bit content of source to destination : SI = SI +4 and DI = DI + 4 if DF =0**
   SI = SI - 4 and DI = DI - 4 if DF =1

LODS B/W/D :

This instruction is used to load string byte into AL and string word into AX register. This instruction copies a byte or word from a string location pointed by SI into the AL/AX register.

Operation : AL ← DS: [SI]

Types :

1. **LODSB : Load 8 bit content of AL from source : SI = SI +1 if DF =0**
   AL ← DS: [SI] SI = SI -1 if DF =1
2. **LODSW : Load 16 bit content of AX from source : SI = SI +2 if DF =0**
   AX ← DS: [SI] SI = SI - 2 if DF =1
3. **LODSD : Load 32 bit content of EAX from source : SI = SI +4 if DF =0**
   EAX ← DS: [SI] (80386) SI = SI - 4 if DF =1

STOS B/W/D :

This instruction is used to load string byte from AL and string word from AX register. This instruction copies a byte or word from a AL/AX reg into string location pointed by DI in extra segment.

Operation : AL → ES: [DI]

Types :

1. **STOSB : Store 8 bit content of AL into destination : DI = DI + 1 if DF =0**
   DI = DI - 1 if DF =1
   AL → ES: [DI]
2. **STOSW : Store 16 bit content of AX into destination : DI = DI + 2 if DF =0**
   DI = DI - 2 if DF =1
   AX → ES: [DI]
3. **STOSD : Store 32 bit content of EAX into destination : DI = DI + 4 if DF =0**
   DI = DI - 4 if DF =1
   EAX → DS: [DI] (80386)

CMPS B/W/D :

This instruction is used to compare a byte/word into source string of DS pointed by SI with a byte/word pointed by DI in ES.

Operation : Compare DS:[SI] with ES: [DI]

Types :

1. **CMPSB : Compare 8 bit content of source with destination : SI = SI +1 and DI = DI + 1 if DF =0**
   SI = SI -1 and DI = DI - 1 if DF =1
2. **CMPSW : Compare 16 bit content of source with destination : SI = SI +1 and DI = DI + 1 if DF =0**
   SI = SI -1 and DI = DI - 1 if DF =1
3. **CMPSD : Compare 32 bit content of source with destination : SI = SI +1 and DI = DI + 1 if DF =0**
   SI = SI -1 and DI = DI - 1 if DF =1

CMPS B/W/D :

This instruction is used to compare a byte/word into source string of DS pointed by SI with a byte/word pointed by DI in ES.

Operation : Compare DS:[SI] with ES: [DI]

Types :

1. CMPSB : Compare 8 bit content of source with destination : SI = SI +1 and DI = DI + 1 if DF =0
   SI = SI -1 and DI = DI - 1 if DF =1

2. CMPSW : Compare 16 bit content of source with destination : SI = SI +1 and DI = DI + 1 if DF =0
   SI = SI -1 and DI = DI - 1 if DF =1

3. CMPSD : Compare 32 bit content of source with destination : SI = SI +1 and DI = DI + 1 if DF =0
   SI = SI -1 and DI = DI - 1 if DF =1

SCAS B/W/D :

This instruction is used to compare a byte/word into AL/AX with a byte pointed by DI in ES.

Operation : Compare AL/AX with ES: [DI]

Types :

1. SCASB : Compare 8 bit content of AL with destination : DI = DI + 1 if DF =0
   DI = DI - 1 if DF =1

2. SCASW : Compare 16 bit content of AX with destination : DI = DI + 2 if DF =0
   DI = DI - 2 if DF =1

3. SCASD : Compare 32 bit content of EAX with destination : DI = DI + 4 if DF =0
   DI = DI - 4 if DF =1