MODULE 2

There are eight addressing modes in 8086 MPU. These modes are:

- Immediate Addressing Mode
- Register Addressing Mode
- Direct Addressing Mode
- Register Indirect Addressing Mode
- Based Addressing Mode
- Indexed Addressing Mode
- Based-Index Addressing Mode
- Based-Index with displacement addressing mode

Instruction Format

1. Operation Destination, Source
   Opcode Operand, Operand

2. Operation Source
   Opcode Operand

3. Operation
   Opcode

Immediate Addressing

The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode.

```
MOV CX, 4929 H, ADD AX, 2387 H, MOV AL, FFH
```

Register Addressing

It means that the register is the source of an operand for an instruction.

```
MOV CX, AX; copies the contents of the 16-bit AX register into
; the 16-bit CX register),
ADD BX, AX
```

Direct Addressing

The addressing mode in which the effective address of the memory location is written directly in the instruction.

```
MOV AX, [1592H], MOV AL, [0300H]
```

Register Indirect Addressing

This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI & SI.

```
MOV AX, [BX]; Suppose the register BX contains 4895H, then the contents

; 4895H are moved to AX
```

Based Addressing

In this addressing mode, the offset address of the operand is given by the sum of contents of the BX/BP registers and 8-bit/16-bit displacement.

```
MOV DX, [BX+04], ADD CL, [BX+08]
```

Indexed Addressing

In this addressing mode, the operands offset address is found by adding the contents of SI or DI register and 8-bit/16-bit displacements.

```
MOV BX, [SI+16], ADD AL, [DI+16]
```

Based Index Addressing

In this addressing mode, the offset address of the operand is computed by summing the base register to the contents of an Index register.

```
ADD CX, [AX+SI], MOV AX, [AX+DI]
```

Based Index with Displacement Addressing

In this addressing mode, the operands offset is computed by adding the base register contents. An Index registers contents and 8 or 16-bit displacement.

```
MOV AX, [BX+DI+08], ADD CX, [BX+SI+16]
```

8086/8088 Instructions - Categories

1. **Data Copy/Transfer** Instructions
   - Used to transfer data from source operand to destination operand.
   - All the store, move, load, exchange, input and output instructions belong to this category.
2. **Arithmetic** and **Logical Instructions**
   - Instructions performing arithmetic, logical, increment, decrement, compare and scan
     - The arithmetic instructions affect all the condition code flags.
     - The operands are either the registers or memory locations or immediate data depending on addressing mode
3. **Branch** Instructions
   - Transfer control of execution to the specified address.
   - All the call, jump, interrupt and return instructions belong to this class.
4. **Loop** Instructions
   - Used to implement unconditional and conditional loops.
   - The **loop** instructions (LOOP, LOOPNZ, LOOPZ) belong to this category.

8086/8088 Instructions

5. **Machine Control** Instructions
   - These instructions control the machine status. NOP, HLT, WAIT and LOCK instructions belong to this class.
6. **Flag Manipulation** Instructions
   - Instructions which directly affect the flag register
   - Instructions like CLC, STC, CLD,STD,CLI,STI, etc. belong to this category of instructions.
7. **Shift** and **Rotate** Instructions
   - Involve the bitwise shifting or rotation in either direction with or without a count in CX.
8. **String** Instructions
   - These instructions involve various string manipulation operations like load, move, scan, compare, store, etc.

Data Copy/Transfer Instructions

1. MOV: (Move)

- Transfers data from one register/memory location to another register/memory location.
  - Source - any one of the segment registers or other general or special purpose registers or a memory location and,
  - Destination - another register or memory location.
    - In case of immediate addressing mode, a segment register cannot be a destination register.
    - To load the segment registers with immediate data, one will have to load any general purpose register with the data and then it will have to be moved to that particular segment register.
    - Both the source and destination operands cannot be memory locations (except for string instructions)

MOV- Examples

- MOV AX, 5000H; Immediate
- MOV AX, [2000H]; Direct
- MOV AX, BX; Register
- MOV AX, [BX]; Register Indirect
- MOV AX, 50H[BX]; Based relative, 50H Displacement

- MOV DS, 5000H; Not permitted (invalid)
- MOV AX, 5000H
  MOV DS, AX Alternative (Valid)

Data Copy/Transfer Instructions

2. PUSH: Push to Stack

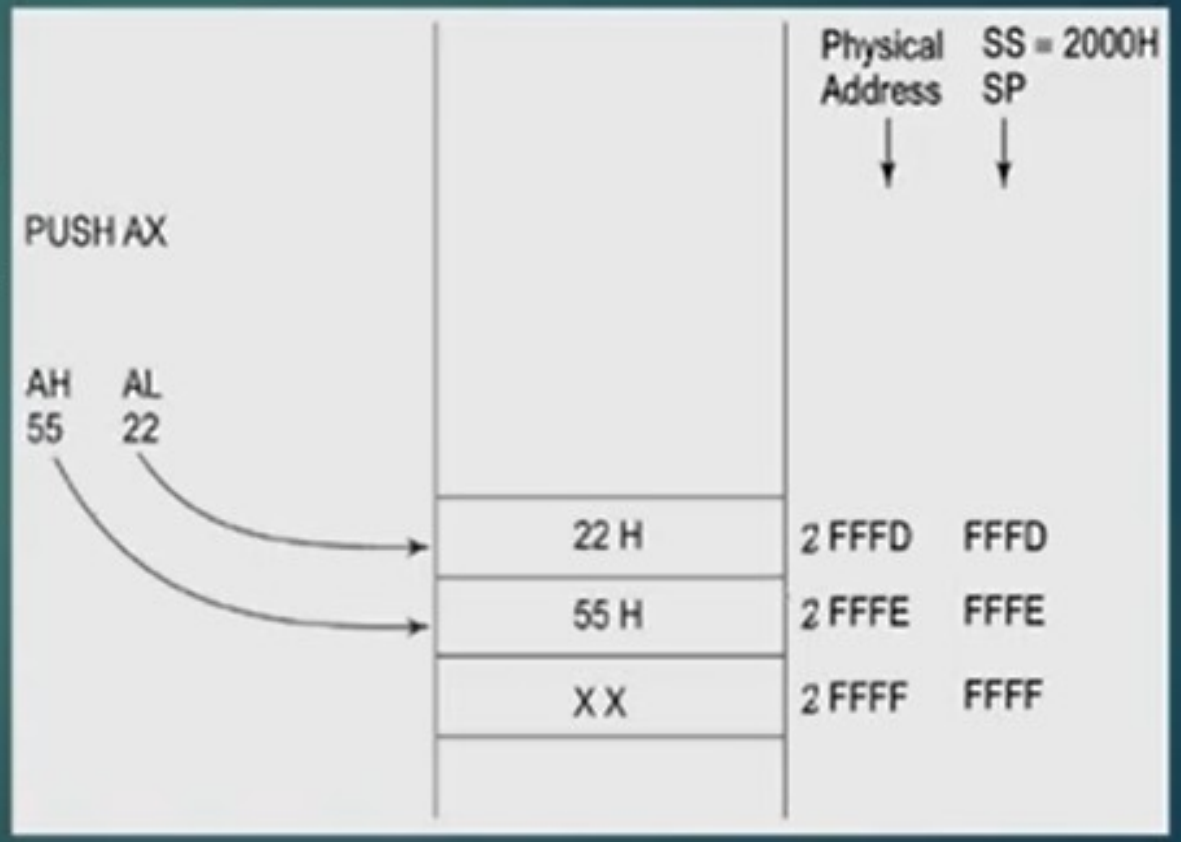
- This instruction pushes the contents of the specified register/memory location on to the stack.
  - Push operation decrements SP by 2 and then stores the two byte contents of the operand onto the stack
    - The higher byte is pushed first (Higher Address)and then the lower byte (Lower Address).

Examples

- PUSH AX
- PUSH DS
- PUSH [5000H];

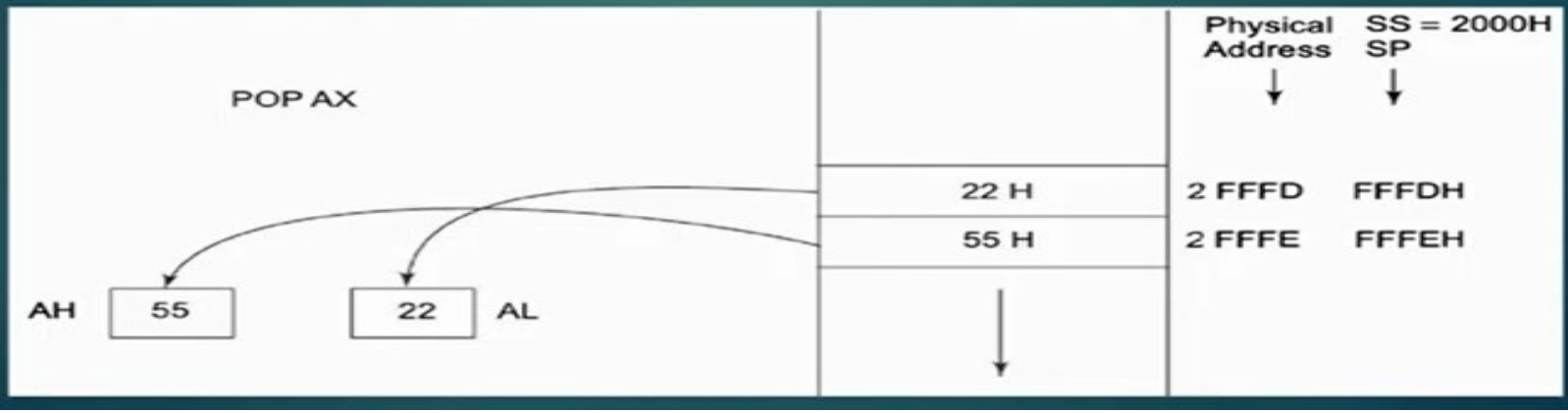
PUSH

- SS : SP points to the stack top and AH, AL contains data to be pushed.
- The sequence of operation as below:
  - Current stack top is already occupied so decrement SP by one then store AH into the address pointed to by SP.
  - Further decrement SP by one and store AL into the location pointed to by SP.
    - Thus SP is decremented by 2 and AH—AL contents are stored in stack memory
    - Contents of SP points to a new stack top.



POP

- 16-bit contents of current stack top are poped into the specified operand as follows.



Data Copy/Transfer Instructions

4. XCHG: Exchange

- Exchanges the contents of the specified source and destination operands
  - May be registers or one of them may be a memory location.
  - Exchange of contents of two memory locations is not permitted.
  - Immediate data is also not allowed in these instructions.

Examples

- XCHG [5000H] , AX
  - Exchanges data between AX and a memory location [5000H] in the data segment.
- XCHG BX , AX
  - Exchanges data between AX and BX.

Data Copy/Transfer Instructions

5. IN: Input the Port

- For reading an input port.
  - The address of the input port specified in the instruction directly or indirectly.
  - AL and AX are the allowed destinations for 8 and 16-bit input operations.
  - DX is the only register (implicit) which is allowed to carry the port address (16 bit)

Examples

- IN AL , 03H --Reads data from an 8-bit port whose address is 03H and stores it in AL
- IN AX, DX --Reads data from a 16-bit port whose address is in DX and stores it in AX
- MOV DX,0800H --The 16-bit address is taken in DX

  IN AX, DX --Read the content of the port in AX

Data Copy/Transfer Instructions

6. OUT: Output to the Port

- For writing to an output port.
  - The address of the output port may be specified in the instruction directly or implicitly in DX (If the port address is of 16 bits)
  - Contents of AX (16 bit) or AL (8 bit) are transferred to a directly or indirectly addressed port
  - The data to an odd addressed port is transferred on D8-D15 while that to an even addressed port is transferred on D0-D7.

Example

```
OUT   03H , AL      ; This sends data available in AL to a port whose address is 03H.
OUT   DX , AX       ; This sends data available in AX to a port whose address is specified implicitly in DX.
MOV DX , 0300H      ; The 16-bit port address is taken in DX.
OUT DX , AX         ; Write the content of AX to a port of which address is in DX.
```

Data Copy/Transfer Instructions

7. XLAT: Translate

- Used for finding out the codes in case of code conversion problems, using look up table technique.
  - Suppose, a hexadecimal key pad having 16 keys from 0 to F is interfaced with 8086 and the number is displayed in 7 segment display device
  - A look up table of codes is prepared to store the 7-segment codes for 0 to F at 16 locations (e.g. 2000H to 200FH) sequentially
- The following sequence of instructions perform the task.
  - MOV AX, SEG TABLE --Address of the segment containing look-up-table
  - MOV DS,AX --is transferred in DS
  - MOV AL, CODE --Code of the pressed key is transferred in AL
  - MOV BX , OFFSET TABLE --Offset of the code look up table in BX
  - XLAT --Find the equivalent code and store in AL

Data Copy/Transfer Instructions

7. LEA: Load Effective Address

- Loads the effective address formed by destination operand into the specified source register.

Examples

- LEA BX, ADR : Offset of the Label-ADR will be transferred to Register BX.
- LEA SI, ADR [BX]: offset of Label ADR will be added to content of BX to form effective address and it will be loaded in SI

8. LDS/LES: Load Pointer to DS/ES

- Copies a word from memory locations into the register specified in the instruction.
- Then copies a word from the next memory locations into the DS(ES) register.
- Useful for pointing to SI(DI) and DS(ES) in string operation.

Example

- LDS BX, 5000H
- LES BX, 5000H

Data Copy/Transfer Instructions

9. LAHF : load AH from Lower Byte of Flag

- Loads the AH register with the lower byte of the flag register.
- Used to observe the status of all the condition code flags (except over flow) at a time.

10. SAHF: Store AH to Lower Byte of Flag Register

- Sets or resets the condition code flags (except overflow) in the lower byte of the flag register depending upon the corresponding bit positions in AH.

11. PUSHF: Push Flags to Stack

- Pushes the flag register on to the stack
  - first the upper byte and then the lower byte is pushed on to it.

12. POPF: Pop Flags from Stack

- The pop flags instruction loads the flag register (2 bytes) memory location addressed by SP and SS

Arithmetic Instructions

ADD: Add

- Adds an immediate data or contents of a memory location specified in the instruction or a register (source) to the contents of another register (destination) or memory location.
  - Both the source and destination operands cannot be memory operands.
    - Memory to memory addition is not possible.
  - Contents of the segment registers cannot be added using this instruction.
  - All the condition code flags are affected, depending upon the result.

Examples

ADD AX, 0100H	Immediate
ADD AX, BX	Register
ADD AX, [SI]	Register indirect
ADD AX, [5000H]	Direct
ADD [5000H], 0100H	Immediate
ADD 0100H	Destination AX (implicit)

Arithmetic Instructions

ADC: Add with Carry

- Same operation as ADD instruction, but adds the carry flag bit to the result
  - carry flag bit may be set as a result of the previous calculations
  - All the condition code flags are affected by this instruction.

Example

ADC 0100H	Immediate (AX implicit)
ADC AX, BX	Register
ADC AX, [SI]	Register indirect
ADC AX, [5000H]	Direct
ADC [5000H], 0100H	Immediate

Arithmetic Instructions

INC: Increment / DEC: Decrement

- Increases/ Decreases the contents of the specified register or memory location by 1 [Data±1]
  - All the condition code flags are affected except the carry flag CF.
  - Immediate data cannot be operand of this instruction.

Examples

```
INC     AX        Register
INC     [BX]      Register indirect
INC     [5000H]   Direct
```

```
DEC      AX        Register
DEC      [5000H]   Direct
```

Arithmetic Instructions

SBB: Subtract with Borrow

- Subtracts the source operand and the borrow flag (CF) which may reflect the result of the previous calculations, from the destination operand.
  - Subtraction with borrow, here means subtracting 1 from the subtraction obtained by SUB, if carry (borrow) flag is set.
  - The result is stored in the destination operand.
  - All the flags are affected (Condition code) by this instruction.

Examples

- SBB AX, 0100H Immediate [destination AX]

Arithmetic Instructions

CMP: Compare

- Compares the source operand (a register or an immediate data or a memory location), with a destination operand (may be a register or a memory location)
  - For comparison, it subtracts the source operand from the destination operand but does not store the result anywhere.
  - The flags are affected depending upon the result of the subtraction.
    - If both of the operands are equal, zero flag is set.
    - If the source operand is greater than the destination operand, carry flag is set
    - If the source operand is less than the destination operand , carry flag is reset.
- The examples of this instruction are as follows:
  - CMP AX, 0100H — Immediate
  - CMP [5000H], 0100H — Direct
  - CMP BX, [SI] — Register indirect
  - CMP BX, CX — Register

07:00

Arithmetic Instructions

MUL: Unsigned Multiplication Byte or Word

- Multiplies an unsigned byte or word (from a general purpose registers or memory locations) by the contents of AL or AX.
  - The most significant word of the result is stored in DX
  - The least significant word of the result is stored in AX.
  - All the flags are modified depending upon the result.
  - Immediate operand is not allowed in this instruction.
  - If the most significant byte or word of the result is '0'
    - CF and OF both will be set.

```
1. MUL BH               ; (AX) ← (AL) × (BH)
2. MUL CX               ; (DX) (AX) ← (AX) × (CX)
3. MUL WORD PTR [SI]    ; (DX) (AX) ← (AX) × ([SI])
```

Arithmetic Instructions

IMUL: Signed Multiplication

- Multiplies a signed byte or word in source operand by a signed byte in AL or AX.
  - In case of 32-bit results, the higher order word (MSW) is stored in DX and the lower order word (LSW) is stored in AX.
  - The AF, PF, SF, and ZF flags are undefined after IMUL.
  - If AH and DX contain parts of 16 and 32-bit result respectively,
    - CF and OF both will be set.
  - The AL and AX are the implicit operands in case of 8 bits and 16 bits multiplications respectively.
  - The unused higher bits of the result are filled by sign bit and CF, AF are cleared.

```
1. IMUL     BH
2. IMUL     CX
3. IMUL     [SI]
```

12:00

Arithmetic Instructions

DIV: Unsigned Division

- It divides an unsigned word or double word by a 16-bit or 8-bit operand.
- The dividend must be in AX for 16-bit operation and divisor may be specified using any one of the addressing modes except immediate.
  - The result will be in AL (quotient) while AH will contain the remainder.
  - If the result is too big to fit in AL, type 0 (divide by zero) interrupt is generated.
- In case of a double word dividend (32-bit)- The higher word in DX and lower word in AX.
  - The quotient and the remainder, in this case, will be in AX and DX respectively.
- This instruction does not affect any flag.

IDIV: Signed Division

- Same operation as the DIV instruction, but with signed operands and signed Results

14:53

Arithmetic Instructions

NEG: Negate

- The negate instruction forms 2's complement of the specified destination in the instruction.
- For obtaining 2's complement, it subtracts the contents of destination from zero.
- The result is stored back in the destination operand which may be a register or a memory location.
- If OF is set, it indicates that the operation could not be completed successfully.
- This instruction affects all the condition code flags.

Arithmetic Instructions

CBW: Convert Signed Byte to Word

- Copies the sign bit of a byte to be converted to all the bits in the higher byte of the result word.
- The byte to be converted must be in AL. The result will be in AX.
- It does not affect any flag.

CWD: Convert Signed Word to Double Word

- This instruction copies the sign bit of AX to all the bits of the DX register.
- This operation is to be done before signed division.
- It does not affect any flag.

Arithmetic Instructions

DAA: Decimal Adjust Accumulator

- Used to convert the result of the addition of two packed BCD numbers to a valid BCD number.
  - The result has to be only in AL.
- If the lower nibble is greater than 9, or if AF is set
  - Will add 06 to the lower nibble in AL.
  - After adding 06 in the lower nibble, if the upper nibble of AL is greater than 9 or if CF=1

    DAA instruction adds 60H to AL.
- The instruction DAA affects AF, CF, PF, and ZF flags.

```
(i) AL = 53        CL = 29
    ADD AL, CL     ; AL ← (AL) + (CL)
                   ; AL ← 53 + 29
                   ; AL ← 7C
    DAA            ; AL ← 7C + 06 (as C>9)
                   ; AL ← 82
```

```
(ii) AL = 73       CL = 29
     ADD AL, CL    ; AL ← AL + CL
                   ; AL ← 73 + 29
                   ; AL ← 9C
     DAA           ; AL ← 02 and CF = 1
                     AL = 7 3
                        +
                   CL = 2 9
                   ---------
                        9 C
                        + 6
                   ---------
                        A 2
                      + 6 0
                   ---------
                   CF = 1  0 2  in AL
```

Arithmetic Instructions

DAS: Decimal Adjust after Subtraction

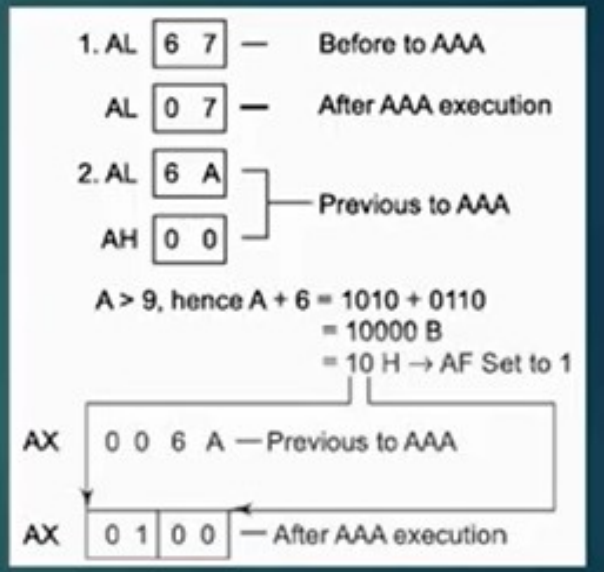
- This instruction converts the result of subtraction of two packed BCD numbers to a valid BCD number.
  - The subtraction has to be in AL only.
- If the lower nibble of AL is greater than 9
  - Will subtract 06 from lower nibble of AL.
- If the result of subtraction sets the carry flag or if upper nibble is greater than 9,
  - Will subtracts 60H from AL.
- Modifies the AF, CF, SF, PF &ZF flags.
  - The OF is undefined after DAS.

```
(i) AL = 75        BH = 46
    SUB AL, BH     ; AL ← 2 F = (AL) - (BH)
                   ; AF = 1
    DAS            ; AL ← 2 9 (as F > 9, F - 6 = 9)
(ii) AL = 38       CH = 6 1
    SUB AL, CH     ; AL ← D 7  CF = 1 (borrow)
    DAS            ; AL ← 7 7  (as D > 9, D - 6 = 7)
                   ; CF = 1 (borrow)
```

Arithmetic Instructions

AAA: ASCII Adjust After Addition

- Performed after an addition of two ASCII coded operands to give a byte of result in AL.
  - Converts the resulting contents of AL to unpacked decimal digits.
- When lower 4 bits of AL is checked for a valid BCD number in the range 0 to 9
- If it is between 0 to 9 and AF is zero
  - AAA sets the 4 high order bits of AL to 0 [The AH must be cleared before addition]
- If the lower digit of AL is between 0 to 9 and AF is set
  - *06 is added to AL*, The upper 4 bits of AL are cleared and AH is incremented by 1.
- If the value in the lower nibble of AL is greater than 9 then
  - AL is incremented by 06, AH is incremented by 1
  - AF and CF flags are set to 1, The remaining flags are unaffected
  - Higher 4 bits of AL are cleared to 0.
  - The AH is modified as sum of previous contents (usually 00) and the **carry** from the adjustment
- This instruction does not give exact ASCII codes of the sum, but they can be obtained by adding 3030H to AX.



Arithmetic Instructions

AAS: ASCII Adjust AL after Subtraction

- AAS instruction corrects the result in AL register after subtracting two unpacked ASCII operands.
  - The result is in unpacked decimal format.
- If the lower 4 bits of AL register are greater than 9 or if the AF flag is 1
  - AL is decremented by 6 and AH register is decremented by 1
  - CF and AF are set to 1.
- Otherwise,
  - CF and AF are set to 0,
  - Result needs no correction.
  - As a result, the upper nibble of AL is 00 and the lower nibble may be any number from 0 to 9.
- The procedure is similar to the AAA instruction except for the subtraction of 06 from AL.
  - AH is modified as difference of the previous contents (usually zero) of AH and the borrow for adjustment.

AAD: ASCII Adjust before Division

- Converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL.
  - This adjustment must be made before dividing the two unpacked BCD digits in AX by an unpacked BCD byte.
  - PF, SF, ZF are modified while AF, CF, OF are undefined, after the execution of the instruction AAD.

Logical Instructions

AND: logical AND

- This instruction bit by bit ANDs the source operand to the destination operand
  - Source may be an immediate, a register or a memory location
  - Destination may be a register or a memory location.
    - At least one of the operands should be a register or a memory operand.
    - Both the operands cannot be memory locations or immediate operands.
    - An immediate operand can- not be a destination operand.

```
1.AND AX, 0008H
2.AND AX, BX
3.AND AX, [5000H]
4.AND [5000H], DX
```

If the content of AX is 3F0FH, the first example instruction will carry out the operation as given below. The result 3F9FH will be stored in the AX register.

```
0 0 1 1   1 1 1 1   0 0 0 0   1 1 1 1   = 3F0F H [AX]
↓ ↓ ↓ ↓   ↓ ↓ ↓ ↓   ↓ ↓ ↓ ↓   ↓ ↓ ↓ ↓     AND
0 0 0 0   0 0 0 0   0 0 0 0   1 0 0 0   = 0008 H
-------------------------------------------------
0 0 0 0   0 0 0 0   0 0 0 0   1 0 0 0   = 0008 H [AX]
```

The result 0008H will be in AX.

```
1.OR AX, 0098H
2.OR AX, BX
3.OR AX, [5000H]
4.OR [5000H], 0008H
```

The contents of AX are say 3F0FH, then the first example instruction will be carried out as given below.

```
  0 0 1 1      1 1 1 1      0 0 0 0      1 1 1 1      = 3F0F H
  ↓ ↓ ↓ ↓      ↓ ↓ ↓ ↓      ↓ ↓ ↓ ↓      ↓ ↓ ↓ ↓        OR
  0 0 0 0      0 0 0 0      1 0 0 1      1 0 0 0      = 0098 H
  ---------------------------------------------------------------
  0 0 1 1      1 1 1 1      1 0 0 1      1 1 1 1      = 3F9F H
```

Thus the result 3F9FH will be stored in the AX register.

Logical Instructions

NOT: logical Invert

- The NOT instruction complements (inverts) the contents of an operand register or a memory location, bit by bit.

```
NOT AX
NOT [5000H]
```

If the content of AX is 200FH, the first example instruction will be executed as shown.

```
   AX      = 0 0 1 0      0 0 0 0      0 0 0 0      1 1 1 1
invert       ↓ ↓ ↓ ↓      ↓ ↓ ↓ ↓      ↓ ↓ ↓ ↓      ↓ ↓ ↓ ↓
             1 1 0 1      1 1 1 1      1 1 1 1      0 0 0 0

Result
in AX =          D            F            F            0
```

The result DFF0H will be stored in the destination register AX.

Logical Instructions

XOR: Logical Exclusive OR

- The XOR operation is again carried out in a similar way to the AND and OR operation.
  - The XOR operation gives a high output, when the 2 input bits are dissimilar.
  - Otherwise, the output is zero.

```
1.XOR    AX, 0098H
2.XOR    AX, BX
3.XOR    AX, [5000H]
```

If the content of AX is 3F0FH, then the first example instruction will be executed as explained. The result 3F97H will be stored in AX.

```
 AX = 3F0FH =      0 0 1 1     1 1 1 1     0 0 0 0     1 1 1 1
        XOR        ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓
      0098H =      0 0 0 0     0 0 0 0     1 0 0 1     1 0 0 0
AX = Result =     ---------------------------------------------
                   0 0 1 1     1 1 1 1     1 0 0 1     0 1 1 1
             = 3F97H
```

Logical Instructions

TEST: Logical Compare Instruction

- The TEST instruction performs a **bit** by bit logical AND operation on the two operands.
  - if the corresponding bits of both operands are 1, Each bit of the result is then set to 1, else the result bit is reset to 0.
  - The result of this ANDing operation is not available for further use
    - But flags (OF, CF, SF, ZF and PF) are affected.
  - The operands may be registers, memory or immediate data.

```
1.TEST     AX, BX
2.TEST     [0500], 06H
3.TEST     [BX] [DI], CX
```

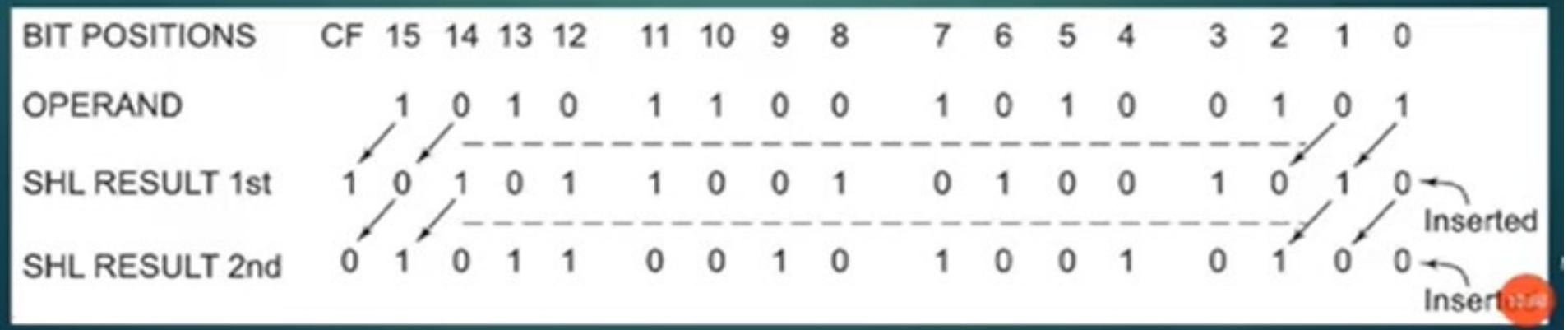
SHIFT and ROTATE instructions

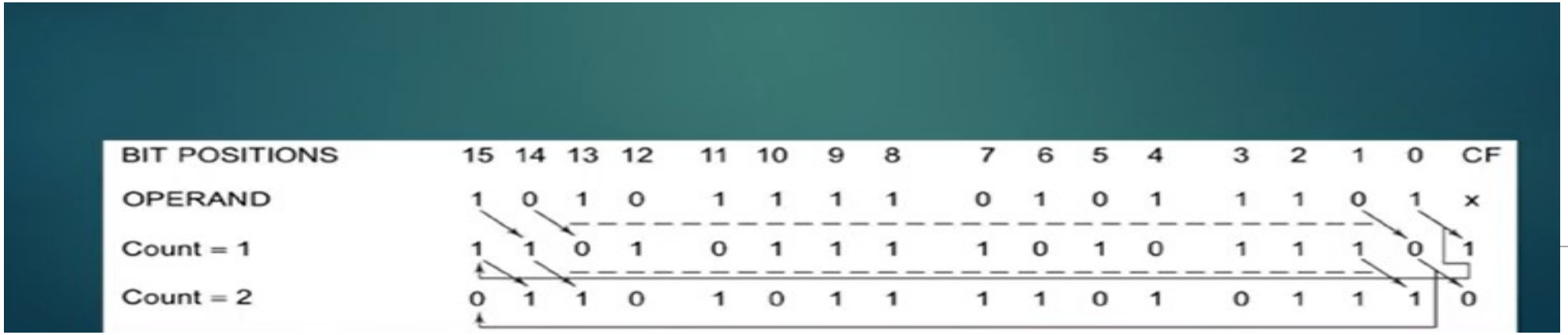
- In case of all the SHIFT and ROTATE instructions, the count is either 1 or specified by register CL.
- Operand may be a register or a memory location by the specified count in the instruction.
- Immediate operand is not allowed
- The result is stored in the destination operand
- All the condition code flags are affected by the shift operation.
- The PF, SF, and ZF flags are left unchanged by the rotate operation.

Logical Instructions

SHL/SAL: Shift logical/Arithmetic left

- Shift the operand word or byte, bit by bit to the left
  - Insert zeros in the newly introduced least significant bits.
  - Shift operation is through carry flag.

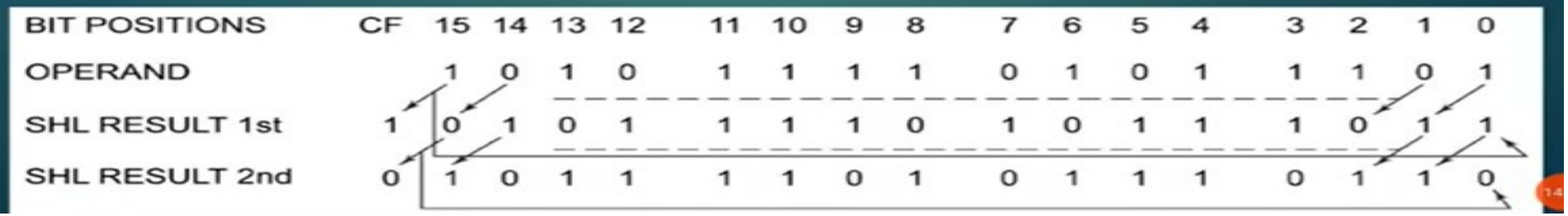




Logical Instructions

ROL: Rotate left without Carry

- This instruction rotates the content of the destination operand to the left by the specified count (bit-wise) excluding carry.
  - The most significant bit is pushed into the carry flag as well as the least significant bit position at each operation.
    - The remaining bits are shifted left subsequently by the specified count positions.



String Manipulation Instructions

Byte strings or *Word strings*: A series of data bytes or words available in memory at consecutive locations.

- For example, a string of characters may be located in consecutive memory locations, where each character may be represented by its ASCII equivalent.

- For referring to a string, two parameters are required,
  a) starting or end address of the string and
  b) length of the string. [The length of a string is usually stored as count in the CX register].
- The incrementing or decrementing of the pointer, depends upon the Direction Flag (DF) status.
  - If it is a byte string operation, the index registers are updated by one.

Previous Year Questions-3 Marks Questions

- List the registers used in 8086 microprocessor.
- Describe function of the following signals of 8086.
  - INTR
  - READY
  - HOLD
- What are the different flag bits available within the flag register of 8086

Previous Year Questions-3 Marks Questions

- With the help of a timing diagram show the transition of control signals involved in the I/O read operation of 8086 in minimum mode.
- How does the 8086 processor access a word from an odd memory location? How many memory cycles does it take?
- Draw the timing diagram for the 8086 minimum mode memory write operation
- List any six features of 8088 microprocessor.
- What are the differences between 8086 and 8088 microprocessors?

Previous Year Questions- 4/ 5 Marks Questions

- What are the different information conveyed by the Queue status signals QS0 and QS1 of 8086 in maximum mode?
- With a neat diagram describe how 8086 memory is organised at physical level
- Give the architectural and signal differences between 8086 and 8088

Previous Year Questions-9 Mark Questions

- Explain register set of 8086.
- Draw and explain the internal block diagram of 8086
- Draw the Memory Read timing diagram of 8086 in Minimum mode. Describe the status of the relevant signals during each clock period.
- Explain minimum mode configuration of 8086.
- Explain the physical and logical memory organization of 8086

3 Mark Questions

1. Describe any three addressing modes used in 8086
2. Explain the following addressing modes of 8086 with suitable examples.

   a) Immediate b)Register Indirect

4. With the help of an example state the differences in the functioning aspects of the instructions SHR and SAR of 8086.
5. Describe the use of 8086 instructions: PUSH, POP and PUSHF

7. What is the difference in the execution of an 8086 inter-segment and intra segment CALL instruction?

3 Mark Questions

8. Find the physical address of the destination operands referred in the following instructions, if DS:0223H, DI:0CCCH and SI:1234H

   a) MOV [DI], AL b) MOV [SI][56H], BL

9. Find the physical address of the memory locations referred by the instructions, when DS:BCOOH, SI:0023H, BX:0012H

   a)MOV AL,[SI] b) MOV [BX][SI],DL

3. Write an ALP to find the largest number from an unordered array of 8-bit numbers?

```
        MOV CX, [2000H]     ; Set the length of array in CX
        MOV SI, 3000H       ; Point the first element of array with SI
        MOV AL, [SI]        ; Select the first element
L1:     INC SI              ; Increment SI to select second number
        CMP AL, [SI]        ; Compare the second number with content of AL
        JNC L2              ; If AL remains the larger number, repeat the comparison with next number
        MOV AL, [SI]        ; If 2nd number is larger, move to step L2 to update the content of AL with larger number
L2:     LOOP L1             ; Repeat the step [CX]times
        MOV [5000H], AL     ; Store Result in the location 5000H
```

9 Marks Questions

2. Write an 8086 assembly language program to find the sum of all numbers less than 50 in an array of n numbers

```
    MOV CX, [2000H]     ; Set the length of array in CX
    MOV SI, 3000H       ; Point first element using SI
    MOV AX, 0000H       ; Clear Accumulator
    MOV BX, 50D         ; Set the value 50 in BX
L1: CMP BX,[SI]         ; Compare the number with 50
    JNB L2              ; If number is not less than 50, continue with next numbers
    ADD AX, [SI]        ; If number is less than 50, add with AX
L2: INC SI              ; Select next number
    DEC CX              ; Decrement count for repeat operation
    JNZ L1              ; Repeat
    MOV [5000H], AX     : Store result in the location 5000H
```

3. Write an 8086 ALP to find the count of even and odd numbers from a set of 10 sixteen bit numbers stored in location starting from a known address. Store the results in two different locations.

6 Marks/ 5 Marks Questions

1. What are the five different addressing modes available in 8051 microcontroller
2. Describe any four control transfer instructions of 8051?
3. What is the difference between LCALL and ACALL instructions

4. Is "DIV A, R1" a valid instruction? Justify your answer.
5. What is the use of following 8051 instructions :

   ADDC, SUBB, CPL, RLC and SWAP?

6. Explain the working of the following instructions with suitable example.

   a) MOVX b) XCHD c) AJMP d) SWAP

10 Marks Questions

1. Describe the addressing modes of 8051 with one example for each
2. What are the five different categories of 8051 instruction set? Explain each category with appropriate examples.