Chapter 5:

8086 INSTRUCTION SET & PROGRAMMING

Course Content

- Directives
- 8086 Instruction-set
- Arithmetic operations
- logic operations and branch operations
- DOS INT 21
- Programming techniques- looping, counting and indexing
- Additional data transfer and 16-bit instructions

Directives

Assume

- Used to tell the assembler the name of the logical segment it should use for a specified segment.
- You must tell the assembler that what to assume for any segment you use in the program.
- Example, ASSUME: CODE tells the assembler that the instructions for the program are in segment named CODE.

DB – Defined Byte

- Used to declare a byte type variable or to set aside one or more locations of type byte in memory.
- Example, PRICES DB 49H, 98H, 29H: Declare array of 3 bytes named PRICES and initialize 3 bytes as shown.

DD – Define Double Word

- Used to declare a variable of type doubleword or to reserve a memory location which can be accessed as doubleword.

DQ – Define Quadword

- Used to tell the assembler to declare the variable as 4 words of storage in memory.

DT – Define Ten bytes

- Used to tell the assembler to declare the variable which is 10 bytes in length or reserve 10 bytes of storage in memory.

DW – Define Word

- Used to tell the assembler to define a variable type as word or reserve word in memory.

END – End the program

- To tell the assembler to stop fetching the instruction and end the program execution.
- ENDP – it is used to end the procedure.
- ENDS – used to end the segment.

EQU – Equate

- Used to give name to some value or symbol.

EVEN – Align on Even memory address

- Tells the assembler to increment the location counter to the next even address if it is not already at an even address.

EXTRN

- Used to tell the assembler that the name or labels following the directive are in some other assembly module.

GLOBAL – Declares symbols as PUBLIC or EXTRN

- Used to make the symbol available to other modules.
- It can be used in place of EXTRN or PUBLIC keyword.

GROUP – Group related segment

- Used to tell the assembler to group the logical segments named after the directive into one logical segment.
- This allows the content of all the segments to be accessed from the same group.

INCLUDE – include source code from file

- Used to tell the assembler to insert a block of source code from the named file into the current source module. This shortens the source code.

LABEL

- Used to give the name to the current value in the location counter.
- The LABEL directive must be followed by a term which specifies the type you want associated with that name.

LENGTH

- Used to determine the number of items in some data such as string or array.

NAME

- Used to give a specific name to a module when the programs consisting of several modules.

OFFSET

- It is an operator which tells the assembler to determine the offset or displacement of named data item or procedure from the start of the segment which contains it.

ORG – Originate

- Tells the assembler to set the location counter value.
- Example, ORG 7000H sets the location counter value to point to 7000H location in memory.
- $ is often used to symbolically represent the value of the location counter. It is used with ORG to tell the assembler to change the location according to the current value in the location counter. E.g. ORG $+100.

PROC – Procedure

- Used to identify the start of the procedure.

PTR – Pointer

- Used to assign a specific type to a variable or a label.
- It is necessary to do this in any instruction where the type of the operand is not clear.

Public

- It is used to tell the assembler that the specified label or variable is accessible by other modules.
- This is useful in large programs which are generally written in modules.

SEGMENT

- Used to indicate that the start of a logical segment.
- Preceding the segment directive is the name you want to give to the segment.

SHORT

- Used to tell the assembler that only a 1-byte displacement is needed to code a jump instruction.
- If the jump destination is after the jump instruction in the program, the assembler will automatically reserve 2 bytes for the displacement.

TYPE

- Tells the assembler to determine the type of a specified variable.
- The TYPE operator can be used in instruction such as ADD BX, TYPE WORD_ARRAY, where we want to increment BX to point to the next word in an array of words.

8086 Instruction-set

- The 8086 microprocessor supports 8 types of instructions:
- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- String Instructions
- Program Execution Transfer Instructions (Branch & Loop Instructions)
- Processor Control Instructions
- Iteration Control Instructions
- Interrupt Instructions

Data Transfer Instructions

- These instructions are used to transfer the data from the source operand to the destination operand.
- Following are the list of instructions under this group:
- Instruction to transfer a word
- MOV: Used to copy the byte or word from the provided source to the provided destination.
- PPUSH: Used to put a word at the top of the stack.
- POP: Used to get a word from the top of the stack to the provided location.
- PUSHA: Used to put all the registers into the stack.
- POPA: Used to get words from the stack to all registers.
- XCHG: Used to exchange the data from two locations.
- XLAT: Used to translate a byte in AL using a table in the memory.

Instructions for input & output

- IN: Used to read a byte or word from the provided port to the accumulator.
- OUT: Used to send out a byte or word from the accumulator to the provided port.

Instructions to transfer the address

- LEA: Used to load the address of operand into the provided register.
- LDS: Used to load DS register and other provided register from the memory
- LES: Used to load ES register and other provided register from the memory.

OUT:

Instructions to transfer flag registers

- LAHF: Used to load AH with the low byte of the flag register.
- SAHF: Used to store AH register to low byte of the flag register.
- PUSHF: Used to copy the flag register at the top of the stack.
- POPF: Used to copy a word at the top of the stack to the flag register.

Arithmetic Instructions

- These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.
- Following is the list of instructions under this group:
- **Instructions to perform addition**
- ADD: Used to add the provided byte to byte/word to word.
- ADC: Used to add with carry.
- INC: Used to increment the provided byte/word by 1.
- AAA: Used to adjust ASCII after addition.
- DAA: Used to adjust the decimal after the addition/subtraction operation.

Instructions to perform subtraction

- SUB: Used to subtract the byte from byte/word from word.
- SBB: Used to perform subtraction with borrow.
- DEC: Used to decrement the provided byte/word by 1.
- NPG: Used to negate each bit of the provided byte/word and add 1/2's
- complement.
- CMP: Used to compare 2 provided byte/word.
- AAS: Used to adjust ASCII codes after subtraction.
- DAS: Used to adjust decimal after subtraction.

Instruction to perform multiplication

- MUL: Used to multiply unsigned byte by byte/word by word.
- IMUL: Used to multiply signed byte by byte/word by word.
- AAM: Used to adjust ASCII codes after multiplication.

Instructions to perform division

- DIV: Used to divide the unsigned word by byte or unsigned double word by word.
- IDIV: Used to divide the signed word by byte or signed double word by word.
- AAD: Used to adjust ASCII codes after division.
- CBW: Used to fill the upper byte of the word with the copies of sign bit of the lower
- byte.
- CWD: Used to fill the upper word of the double word with the sign bit of the lower
- word.

Bit Manipulation Instructions

- These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc.
- Following is the list of instructions under this group:

Instructions to perform logical operation

NOT: Used to invert each bit of a byte or word.

AND: Used for adding each bit in a byte/word with the corresponding bit in another byte/word.

OR: Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.

XOR: Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.

TEST: Used to add operands to update flags, without affecting operands.

Instructions to perform shift operations

- SHL/SAL: Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
- SHR: Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.
- SAR: Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

Instructions to perform rotate operations

- ROL: Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].
- ROR: Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
- RCR: Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.
- RCL: Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

String Instructions

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

Following is the list of instructions under this group:

- REP: Used to repeat the given instruction till CX≠0.
- REPE/REPZ: Used to repeat the given instruction until CX=0 or zero flag ZF=1.
- REPNE/REPNZ: Used to repeat the given instruction until CX=0 or zero flag ZF=1.
- MOVS/MOVSB/MOVSW: Used to move the byte/word from one string to another.
- COMS/COMPSB/COMPSW: Used to compare two string bytes/words.

String Instructions cont..

- INS/INSB/INSW: Used as an input string/byte/word from the I/O port to the provided memory location.
- OUTS/OUTSB/OUTSW: Used as an output string/byte/word from the provided memory location to the I/O port.
- SCAS/SCASB/SCASW: Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
- LODS/LODSB/LODSW: Used to store the string byte into AL or string word into AX.

Program Execution Transfer Instructions (Branch & Loop Instructions)

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions:

Instructions to transfer the instruction during an execution without any condition:

- CALL: Used to call a procedure and save their return address to the stack.
- RET: Used to return from the procedure to the main program.
- JMP: Used to jump to the provided address to proceed to the next instruction.

Instructions to transfer the instruction during an execution with some conditions:

- JA/JNBE: Used to jump if above/not below/equal instruction satisfies.
- JAE/JNB: Used to jump if above/not below instruction satisfies.
- JBE/JNA: Used to jump if below/equal/ not above instruction satisfies.
- JC: Used to jump if carry flag CF=1
- JE/JZ: Used to jump if equal/zero flag ZF=1
- JG/JNLE: Used to jump if greater/not less than/equal instruction satisfies.
- JGE/JNL: Used to jump if greater than/equal/not less than instruction satisfies.

- JL/JNGE: Used to jump if less than/not greater than/equal instruction satisfies.
- JLE/JNG: Used to jump if less than/equal/if not greater than instruction satisfies.
- JNC: Used to jump if no carry flag (CF=0)
- JNE/JNZ: Used to jump if not equal/zero flag ZF=0
- JNO: Used to jump if no overflow flag OF=0
- JNP/JPO: Used to jump if not parity/parity odd PF=0
- JNS: Used to jump if not sign SF=0
- JO: Used to jump if overflow flag OF=1
- JP/JPE: Used to jump if parity/parity even PF=1
- JS: Used to jump if sign flag SF=1

Processor Control Instructions

These instructions are used to control the processor action by setting/resetting the flag values.

Following are the instructions under this group:

- STC: Used to set carry flag CF to 1
- CLC: Used to clear/reset carry flag CF to 0
- CMC: Used to put complement at the state of carry flag CF.
- STD: Used to set the direction flag DF to 1
- CLD: Used to clear/reset the direction flag DF to 0
- STI: Used to set the interrupt enable flag to 1, i.e., enable INTR input.
- CLI: Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

Iteration Control Instructions

These instructions are used to execute the given instructions for number of times.

Following is the list of instructions under this group:

LOOP: Used to loop a group of instructions until the condition satisfies, i.e., CX=0

LOOPE/LOOPZ: Used to loop a group of instructions till it satisfies ZF=1 & CX=0

LOOPNE/LOOPNZ: Used to loop a group of instructions till it satisfies ZF=0 & CX=0

JCXZ: Used to jump to the provided address if CX=0

Interrupt Instructions

These instructions are used to call the interrupt during program execution.

INT: Used to interrupt the program during execution and calling service specified.

INTO: Used to interrupt the program during execution if OF=1

IRET: Used to return from interrupt service to the main program

DOS INT 21

The INT instruction is somewhat like a FAR call.

– Saves CS:IP and the flags on the stack and goes to the subroutine associated with that interrupt.

Use INT 21H function calls to:

- – Input characters from the keyboard.
- – Output characters to the screen.
- – Input or output strings.

Ex: INT xx ; the interrupt xx can be 00 – FF H

- In x86 processors, 256 interrupts, numbered 00 to FF.

• INT 10H and INT 21H are the most widely used with various functions selected by the value in the AH register.

INT 10H subroutines are burned into the ROM BIOS. – Used to communicate with the computer's screen video.

• Manipulation of screen text/graphics can be done via INT 10H.

• Among the functions associated with INT 10H are changing character or background color, clearing the screen, and changing the location of the cursor. – Chosen by putting a specific value in register AH.

DOS INTERRUPT 21H

- This section uses information inputted from the keyboard, and displayed on the screen.
- A much more dynamic way of processing information.
- When the OS is loaded, INT 21H can be invoked to perform some extremely useful functions.
- Commonly referred to as DOS INT 21H function calls.

DOS INTERRUPT 21H Option 09 outputting a data string on the monitor

- INT 21H can send a set of ASCII data to the monitor.
- Set AH = 09 and DX = offset address of the ASCII data.
- Displays ASCII data string pointed at by DX until it encounters the dollar sign "$".
- The data segment and code segment, to display the message "The earth is but one country" :

```
DATA_ASC    DB    'The earth is but one country','$'

MOV   AH,09            ;option 09 to display string of data
MOV   DX,OFFSET DATA_ASC      ;DX= offset address of data
INT   21H                     ;invoke the interrupt
```

DOS INTERRUPT 21H Option 02 outputting a single character

- To output only a single character, 02 is put in AH, and DL is loaded with the character to be displayed.
- The following displays the letter "J" :

```
MOV    AH,02        ;option 02 displays one character
MOV    DL,'J'       ;DL holds the character to be displayed
INT    21H          ;invoke the interrupt
```

DOS INTERRUPT 21H Option 01 inputting a single character, with echo

- This functions waits until a character is input from the keyboard, then echoes it to the monitor.
- After the interrupt, the input character will be in AL.

```
MOV   AH,01 ;option 01 inputs one character
INT   21H   ;after the interrupt, AL = input character (ASCII)
```

Programming techniques- looping, counting and indexing.

Looping:

- The programming technique used to instruct the microprocessor to repeat tasks is called looping. This task is accomplished by using jump instructions.
- Loops are of 2 types:
- Continuous(repeats a task continuously)
- Conditional(repeats a task until certain data conditions are met)

- Continuous Loop: Repeats a task continuously. A continuous loop is set up by using the unconditional jump instruction. A program with a continuous loop does not stop repeating the tasks until the system is reset.
- Conditional Loop: A conditional loop is set up by a conditional jump instructions. These instructions check flags(Z,CY,P,S) and repeat the tasks if the conditions are satisfied. These loops include counting and indexing.

Conditional Loop And Counter

- A counter is a typical application of the conditional loop.
- A microprocessor needs a counter, flag to accomplish the looping task.
- Counter is set up by loading an appropriate count in a register.
- Counting is performed by either increment or decrement the counter.
- Loop is set up by a conditional jump instruction.
- End of counting is indicated by a flag.

Conditional Loop, Counter And Indexing:

- Another type of loop which includes counter and indexing .
- Indexing: Pointing of referencing objects with sequential numbers. Data bytes are stored in memory locations and those data bytes are referred to by their memory locations.
- Example: Steps to add ten bytes of data stored in memory locations starting ata given location and display the sum.

- The microprocessor needs
- A counter to count 10 data bytes.
- An index or a memory pointer to locate where data bytes are stored.
- To transfer data from a memory location to the microprocessor(ALU)
- To perform addition
- Registers for temporary storage of partial answers
- A flag to indicate the completion of the stack
- To store or output the result.

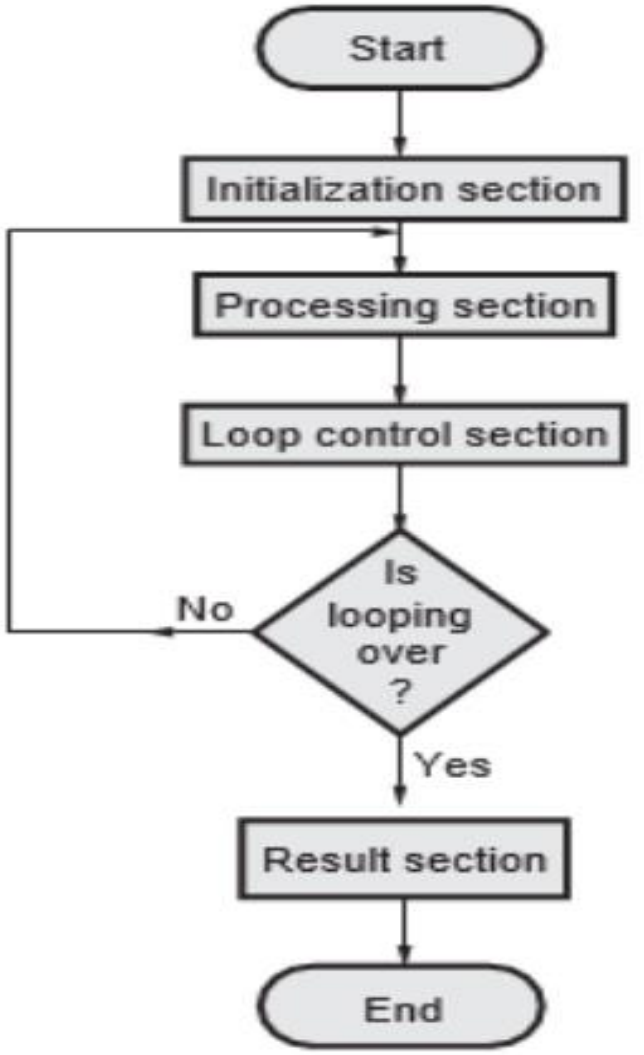


Figure 5.1 Looping flow chart

1. The initialization section establishes the starting values of loop counters for counting how many times loop is executed, Address registers for indexing which give pointers to memory locations and other variables.

2. The actual data manipulation occurs in the processing section. This is the section which does the work.

3. The loop control section updates counters, indices (pointers) for the next iteration.

4. The result section analyzes and stores the results.

Additional data transfer and 16-bit instructions.

16 BIT DATA TRANSFER TO REGISTER PAIR (LXI)

Load register pair immediate

- LXI Reg. pair, 16-bit data.
- The instruction loads 16-bit data in the register pair

designated in the operand.

- Example: LXI H, 2034H or LXI H, XYZ

- Load H and L registers direct
- LHLD 16-bit address
- The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies
- the contents of the next memory location into register H.
- The contents of source memory locations are not altered.

Example: LHLD 2040H

DATA TRANSFER FROM MEMORY TO MICROPROCESSOR

- MOV R,M
- R, M copies data byte from Memory to Register.
- Memory location, its location is specified by the contents of the HL registers.

Example: MOV B, M

- Load accumulator indirect

Arithmetic Instructions in 8086

❖ ADD/ ADC Destination, Source
- This Instruction is used to perform addition
- ADD instruction is just doing addition but ADC instruction will do addition with carry.
- It is compulsory to have same size of source & Destination.
- After addition, answer will be stored in destination Only. Destination must be a register (8 bits or 16 bits) & Source may be register, data or memory location.

```
ADD AL, 20H      ; AL ← AL + 20H
ADD AH, AL       ; AH ← AH + AL
ADD AX, BX       ; AX ← AX + BX
ADC AX, BX       ; AX ← AX + BX + Carry
```

❖ SUB/ SBB Destination, Source
- This Instruction is used to perform Subtraction.
- Similar to ADD/ADC.

```
SUB AL, 20H      ; AL ← AL - 20H
SUB AH, AL       ; AH ← AH - AL
```

❖ INC Destination
- This Instruction adds 1 to destination.
- Destination may be register or memory location.
- It does not effects carry flag.

```
INC AX           ; AX ← AX + 1
INC Byte PTR [BX]  ; [BX] ← [BX]+1
```

❖ DEC Destination
- This Instruction subs 1 to destination.
- Destination may be register or memory location.
- It does not effects carry flag.
- Similar to INC.

❖ MUL Source [Unsigned 8/16 bits data]
- This Instruction performs multiplication.
- If source is of 8 bits, it will multiply with AL and result will be stored in AX register.
- If source is of 16 bits, it will multiply with AX and result will be stored in DX – AX registers.

```
MUL BL           ; AX ← AL * BL
MUL BX           ; DX - AX ← AX * BX
```

- IMUL is also similar to MUL except this instruction performed in signed multiplication.

Arithmetic Instructions in 8086

- DIV Source [Unsigned 8/16 bits data - Divisor]
  - This Instruction performs Division.
  - It Divides a word by byte and a Double word by word.
  - If divisor is of 8 bits, it will divides with AX and result of quotient will be there in AL register & reminder will be stored in AH register.
  - If divisor is of 16 bits, it will divides with DX-AX double word and result of quotient will be there in AX register & reminder will be stored in DX register.

    DIV BL
    DIV BX
  - IDIV is also similar to DIV except that is used for signed number.
- NEG Destination
  - This Instruction performs 2'S Complement.
  - Here destination can be Register or Memory location.

    NEG AL
  - If AL = 46H = 0100 0110 b, then after Execution of NEG instruction, AL = 1011 1001 + 1 = 1011 1010 b = BAH
- CMP Destination, Source
  - This Instruction performs Comparison.
  - If Designation is greater then Source, Carry Flag = 0
  - If Designation is less then Source, Carry Flag = 1
  - If Designation is equal to Source, Zero Flag= 1
  - This instruction does not change the value of destination or Source

    CMP AL, BL
    CMP CL, 25H
    CMP BX, AX
- CBW {Convert Signed Byte to Signed Word}
  - It convert Singed AL into Signed AX.

    CBW
  - If AL = 46H = 0100 0110 b, then after Execution of CBW instruction, AX = 0000 0000 0100 0110 b = 0046H
  - If AL = 96H = 1001 0110 b, then after Execution of CBW instruction, AX = 1111 1111 1001 0110 b = FF96H

Arithmetic Instructions in 8086

- CWD {Convert Signed Word to Signed Double Word}
  - It convert Singed AX into Signed DX-AX.

    CWD
  - If AX = 9746H = 1001 0111 0100 0110 b, then after Execution of CWG instruction,
  - DX - AX = 1111 1111 1111 1111 - 1001 0111 0100 0110 b = FFFF - 0046H
  - Positive numbers are extended by inserting 0's ahead number and negative numbers are extended by inserting 1's ahead numbers.
- ASCII Adjust Instructions
- AAA {ASCII Adjust for Addition}
  - It makes result in unpacked BCD form.
  - ASCII Codes of 0 to 9 are 30H to 39H.
  - AAA should be performed after addition.

    ADD AL, CL

    AAA
  - If AL = 36H {ASCII of 6} and If CL = 37H {ASCII of 7}.
  - After ADD AL,CL result of AL = 6DH.
  - After AAA, result of Carry = 1, AL = 03H

Arithmetic Instructions in 8086

- CWD {Convert Signed Word to Signed Double Word}
  - It convert Singed AX into Signed DX-AX.

    CWD
  - If AX = 9746H = 1001 0111 0100 0110 b, then after Execution of CWG instruction,
  - DX - AX = 1111 1111 1111 1111 - 1001 0111 0100 0110 b = FFFF - 0046H
  - Positive numbers are extended by inserting 0's ahead number and negative numbers are extended by inserting 1's ahead numbers.
- ASCII Adjust Instructions
- AAA {ASCII Adjust for Addition}
  - It makes result in unpacked BCD form.
  - ASCII Codes of 0 to 9 are 30H to 39H.
  - AAA should be performed after addition.

    ADD AL, CL

    AAA
  - If AL = 36H {ASCII of 6} and If CL = 37H {ASCII of 7}.
  - After ADD AL,CL result of AL = 6DH.
  - After AAA, result of Carry = 1, AL = 03H
  - AAA updates only Carry and Auxiliary Carry flag.
- AAS {ASCII Adjust for Subtraction}
  - It makes result in unpacked BCD form.
  - ASCII Codes of 0 to 9 are 30H to 39H.
  - AAS should be performed after subtraction.

    SUB AL, CL

    AAS
  - If AL = 38H {ASCII of 8} and If CL = 32H {ASCII of 2}.
  - After SUB AL,CL result of AL = 06H.
  - After AAS, result of Carry = 0, AL = 06H

Logical Instructions in 8086

❖ NOT Destination
- This Instruction performs 1'S compliment of destination and it stores answer in destination.
- Destination can be register or Memory location

NOT AL
- If AL = 47H = 0100 0111 b
- After execution of instruction AL = 1011 1000 b = B8H

❖ AND Destination, Source
- This Instruction performs Logic AND operation between Destination and Source.
- Destination can be register or Memory location.
- Source can be register or Memory location or Data.

AND AL,BL
- If AL = 47H = 0100 0111 b
- & BL = 36H = 0011 0110 b
- After execution of instruction AL = 0000 0110 b = 06H
- PF, SF & ZF affected, CF = OF = 0 & AF is undefined.

❖ TEST Destination, Source
- This Instruction performs Logic AND operation between Destination and Source, But result is not stored anywhere.

❖ OR Destination, Source
- This Instruction performs Logic OR operation between Destination and Source.
- Destination can be register or Memory location.
- Source can be register or Memory location or Data.

OR AL,BL
- If AL = 47H = 0100 0111 b
- & BL = 36H = 0011 0110 b
- After execution of instruction AL = 0111 0111 b = 77H
- PF, SF & ZF affected, CF = OF = 0 & AF is undefined.

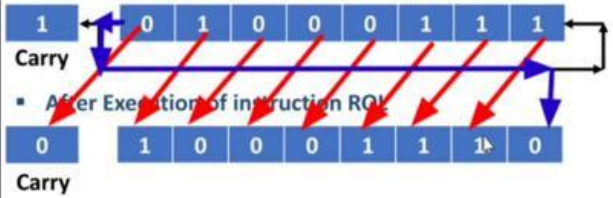
❖ XOR Destination, Source
- This Instruction performs Logic XOR operation between Destination and Source.
- Destination can be register or Memory location.
- Source can be register or Memory location or Data.

XOR AL,BL
- If AL = 47H = 0100 0111 b
- & BL = 36H = 0011 0110 b
- After execution of instruction AL = 0111 0001 b = 71H

Rotate Instructions in 8086

- ROL Destination, Count
  - This Instruction performs Rotate bits Left without carry.
  - If Count is greater then 1 then it has to be stored in CL register. Count 1 can be specified in instruction.

    ROL DH,1
  - If DH = 47H = 0100 0111 b and Carry = 1

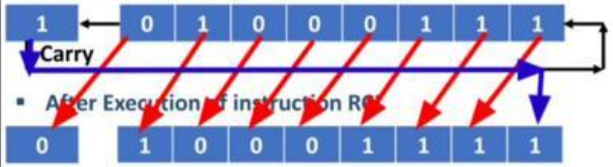


  - After Execution of instruction ROL

Rotate Instructions in 8086

- RCL Destination, Count
  - This Instruction performs Rotate bits Left with carry.
  - If Count is greater then 1 then it has to be stored in CL register. Count 1 can be specified in instruction.

    RCL DH,1
  - If DH = 47H = 0100 0111 b and Carry = 1



  - After Execution of instruction RCL

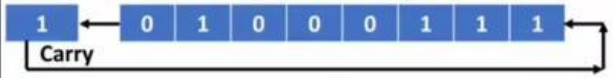
Carry

  - After Rotate Left DH = 8FH, Carry = 0

Rotate Instructions in 8086

- RCL Destination, Count
  - This Instruction performs Rotate bits Left with carry.
  - If Count is greater then 1 then it has to be stored in CL register. Count 1 can be specified in instruction.

    RCL DH,1
  - If DH = 47H = 0100 0111 b and Carry = 1



  - After Execution of instruction RCL

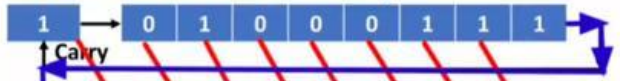
  - After Rotate Left DH = 8FH, Carry = 0
  - Example

    MOV CL,03H

    RCL DH,CL
  - It will Rotate data of DH left by CL = 3 bits.

- RCR Destination, Count
  - This Instruction performs Rotate bits Right with carry.
  - If Count is greater then 1 then it has to be stored in CL register. Count 1 can be specified in instruction.

    RCR DH,1
  - If DH = 47H = 0100 0111 b and Carry = 1



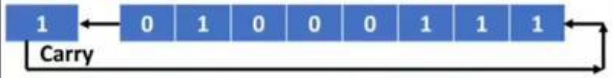
  - After Execution of instruction RCR

Rotate Instructions in 8086

❖ RCL Destination, Count

- This Instruction performs Rotate bits Left with carry.
- If Count is greater then 1 then it has to be stored in CL register. Count 1 can be specified in instruction.

  RCL DH,1
- If DH = 47H = 0100 0111 b and Carry = 1



- After Execution of instruction RCL

- After Rotate Left DH = 8FH, Carry = 0
- Example

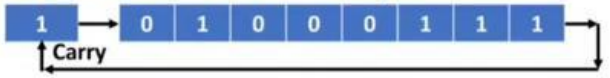
  MOV CL,03H

  RCL DH,CL
- It will Rotate data of DH left by CL = 3 bits.

❖ RCR Destination, Count

- This Instruction performs Rotate bits Right with carry.
- If Count is greater then 1 then it has to be stored in CL register. Count 1 can be specified in instruction.

  RCR DH,1
- If DH = 47H = 0100 0111 b and Carry = 1

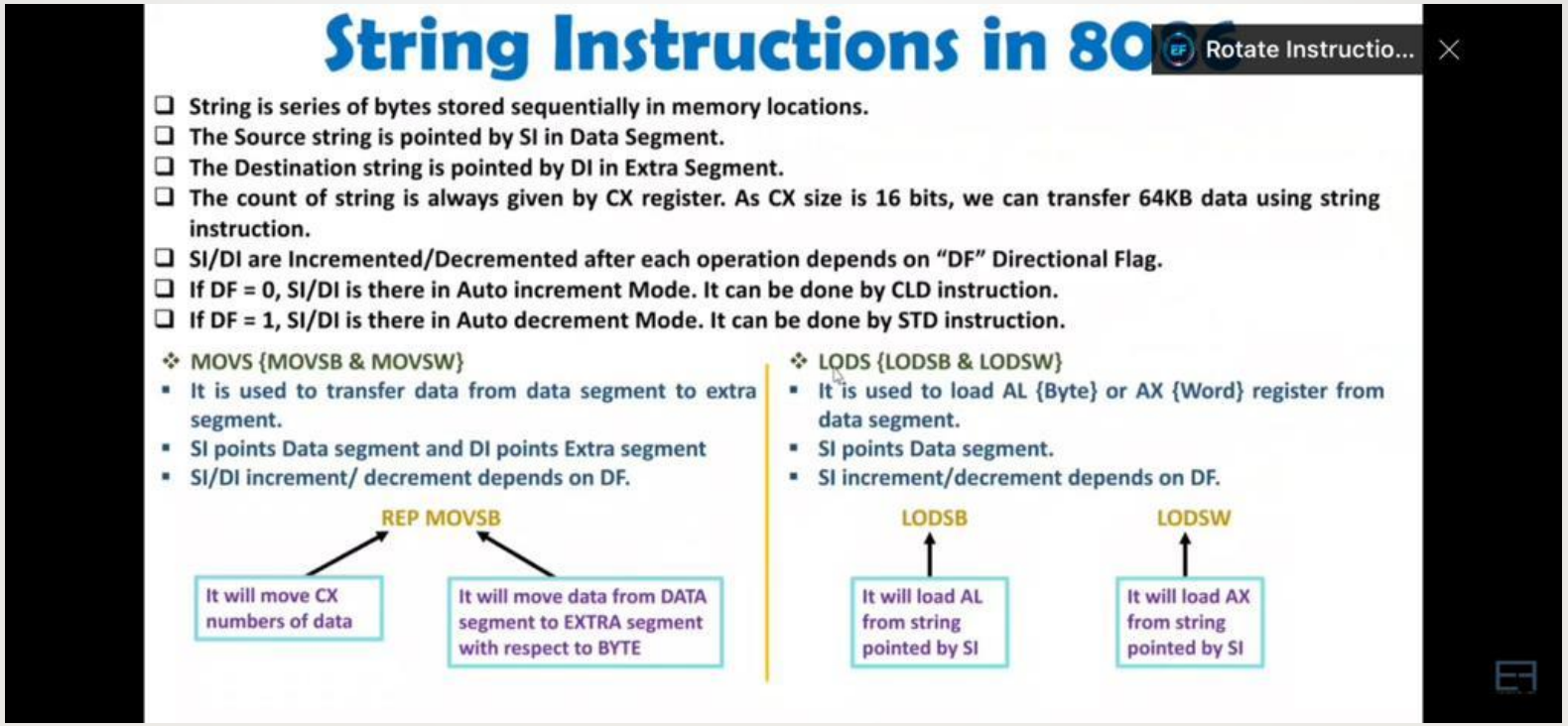


- After Execution of instruction RCR

- After Rotate Right DH = A3H, Carry = 1
- Example

  MOV CL,03H

  RCR DH,CL
- It will Rotate data of DH Right by CL = 3 bits.


String Instructions in 80
Rotate Instructio...
String is series of bytes stored sequentially in memory locations.
The Source string is pointed by SI in Data Segment.
The Destination string is pointed by DI in Extra Segment.
The count of string is always given by CX register. As CX size is 16 bits, we can transfer 64KB data using string instruction.
SI/DI are Incremented/Decremented after each operation depends on "DF" Directional Flag.
If DF = 0, SI/DI is there in Auto increment Mode. It can be done by CLD instruction.
If DF = 1, SI/DI is there in Auto decrement Mode. It can be done by STD instruction.
MOVS {MOVSB & MOVSW}
It is used to transfer data from data segment to extra segment.
SI points Data segment and DI points Extra segment
SI/DI increment/ decrement depends on DF.
REP MOVSB
It will move CX numbers of data
It will move data from DATA segment to EXTRA segment with respect to BYTE
LODS {LODSB & LODSW}
It is used to load AL {Byte} or AX {Word} register from data segment.
SI points Data segment.
SI increment/decrement depends on DF.
LODSB
It will load AL from string pointed by SI
LODSW
It will load AX from string pointed by SI

String Instructions in 8086

❖ STOS {STOSB & STOSW}

- It is used to store AL {Byte} or AX {Word} register on extra segment.
- DI points Extra segment.
- DI increment/decrement depends on DF.

STOSB → It will store AL on string pointed by DI

STOSW → It will store AX on string pointed by DI

❖ CMPS {CMPSB & CMPSW}

- It is used to compare Byte/Word of Data segment with Extra segment.
- SI points Data segment and DI points Extra segment
- SI/DI increment/ decrement depends on DF.

CMPSB → It will compare Byte pointed by SI & DI.

CMPSW → It will compare Word pointed by SI & DI.

❖ SCAS {SCASB & SCASW}

- It is used to compare AL {Byte} or AX {Word} with data of Extra segment.
- DI points Extra segment.
- SI/DI increment/decrement depends on DF.

SCASB → It will compare AL {Byte} data with data pointed by DI.

SCASW → It will compare AX {Word} data with data pointed by DI.

INT 21H (DOS Interrupt) in 8086

- DOS provides various internal interrupts which are used by programmer. The most commonly used is INT 21H.
- This interrupt invokes inbuilt DOS functions [input character, input string, output character output string etc.]
- While calling DOS Interrupt in 8086, we must 1st assign the value to AH register.
- The value in AH register will call different functions of DOS Interrupt [39 functions].

AH	Description
01H	Input character from screen
02H	Display character on screen
0AH	Input string from screen
09H	Display string on screen
4CH	Terminate the program
3CH	Create file
3DH	Open file
3EH	Close file
3FH	Read file
41H	Delete file
56H	Rename file

- Input Character [AH = 01H]

```
MOV AH,01H
INT 21H
```

  - It will take user input from screen.
  - Returns ASCII value in AL register.
  - Example : AL = 65 = 41H, pressed button is A [Uppercase a].

- Input String [AH = 0AH]

```
MOV AH,0AH
LEA DX,string
INT 21H
```

  - The string will be stored from offset address given by DX register.
  - DX will be pointing string from data segment.

- Display Character [AH = 02H]

```
MOV AH,02H
MOV DL,char
INT 21H
```

  - It will display character loaded in DL.

- Display String [AH = 09H]

```
MOV AH,09H
LEA DX,string
INT 21H
```

  - It will display string pointed by DX register.

- Terminate Program [AH = 4CH]

```
MOV AH,4CH
INT 21H
```

  - It will terminate program.