8086 INSTRUCTION SET

DATA TRANSFER INSTRUCTIONS

MOV - MOV Destination, Source

The MOV instruction copies a word or byte of data from a specified destination. The destination can be a register or a memory location. The source can be a register, a memory location or an immediate number. MOV instruction doesn't affect any flag.

- MOV CX, 037AH - Put immediate number 037AH to CX
- MOV BL, [437AH] - Copy byte in DS at offset 437H to BL
- MOV AX, BX - Copy content of register BX to AX
- MOV DL, [BX] - Copy byte from memory at [BX] to DL
- MOV DS, BX - Copy word from BX to DS register.
- MOV RESULT[BP], AX - Copy AX to two memory location; AL to first location, AH to the second; EA of the first memory location is sum of the displacement represented by RESULTS and content of BP
  Physical address: EA + SS
- MOV ES: RESULTS[BP], AX - Same as the above instruction, but physical address = EA + ES, because of the segment override prefix ES.

XCHG - XCHG Destination, Source

The XCHG instruction exchanges the content of a register with the content of another register or with the content of memory location(s). It can not directly exchange the content of two memory locations. The source and destination must both be of the same type. The segment registers can not be used in this instruction. This instruction does not affect any flag.

- XCHG AX, DX — Exchange word in AX with word in DX
- XCHG BL, CH — Exchange byte in BL with byte in CH.
- XCHG AL, PRICES[BX] - Exchange byte in AL with byte in memory at EA = PRICE[BX] in DS

LEA - LEA Register, Source

This instruction determines the offset of the variable or memory location named as the source and puts this offset in the indica-ted 16-bit register. LEA doesn't affect any flag.

- LEA BX, PRICES - Load BX with offset of PRICE in DS
- LEA BP, SS:STACK_TOP - Load BP with offset of STACK_TOP in SS
- LEA CX, [BX][DI] - Load CX with EA = [BX] + [DI]

LDS - LDS Register, Memory address of the first word

This instruction loads new values into the specified register and into the DS register from four successive memory locations. It does not affect any flag.

- LDS BX, [4926] - Copy content of memory at displacement 4926H in DS to BL, content of 4927H to BH. Copy content at displacement of 4928H and 4929H in DS to DS register.
- LDS SI, SPTR - Copy content of memory at displacement SPTR and SPTR+1 in DS to SI register. Copy content of memory at displacement SPTR+2 and SPTR+3 in DS to DS register. DS: SI now points at start of the desired string.

LES - LES Register, Memory address of the first word

This instruction loads new values into the specified register and into the ES register from four successive memory locations. It does not affect any flag.

- LES BX, [789AH] - Copy content of memory at displacement 789AH in DS to BL, content of 789BH to BH, content of memory at displacement 789CH and 789DH in DS is copied to ES register.
- LES DI, [BX] - Copy content of memory at offset [BX] and offset [BX]+1 in DS to DI register. copy content at offset [BX]+2 and [BX]+3 to ES register.

ARITHMETIC INSTRUCTIONS

ADD- ADD Destination, Source

ADC- ADC Destination, Source

These instructions ADD a number from some source to a number in some destination and put the result in the specified destination. The ADC also adds the states of the carry flag to the result. The source and destination in an instruction can not both be memory locations. Flags affected : AF, CF, OF, SF, ZF.

- ADD AL, 74H – Add immediate number 74H to content of AL. Result in [AL]
- ADD CL, BL – Add content of BL plus carry states to content of CL
- ADD DX, BX – Add content of BX to content of DX
- ADD DX, [SI] – Add word from memory at offset [SI] in DS to content of DX.
- ADC AL, PRICES[BX] – Add byte from effective address PRICES[BX] plus carry states to content of AL.
- ADD AL, PRICES[BX] – Add content of memory at effective address PRICES [BX] to AL.

SUB - SUB Destination, Source

SBB - SBB Destination, Source

These instructions substract the number in some source from the number in some destination and put the result in the destination. The SBB instruction also substract the content of carry flag from the destination. The source and destination can not both be memory location. Flags effected : AF, CF, OF, PF, SF, ZF

- SUB CX, BX – CX - BX ; Result in CX
- SBB CH, AL – Subtract content of AL and content of CF from content of CH. Result in CH.
- SUB AX, 3427H – Subtrat immediate number 3427H from AX.
- SBB BX, [3427H] – Subtract word at displacement 3427H in DS and content of CF from BX

- SUB PRICES[BX], 04H – Subtract 04 from byte at effective address PRICES[BX], if PRICES is declared with DB; Subtract 04 from word at effective address PRICES[BX], if it is declared with DW.
- SBB CX, TABLE[BX] – Subtract word from effective address TABLE[BX] and states of CF from CX.
- SBB TABLE[BX], CX – Subtract CX and states of CF from word in memory at effective address TABLE[BX]

MUL - MUL Source

This instruction multiplies an unsigned byte in some source with an AL register or an unsigned word in some source with AX register. The source can be register or a memory location. If the most significant byte of a 16-bit result or the most significant word of a 32-bit result is 0, CF and OF will both be 0's. AF, PF, SF and ZF are undefined after a MUL instruction.

- MUL BH – Multiply AL with BH; result in BX.
- MUL CX – Multiply AX with CX; result high word in DX, Low word in AX.
- MUL BYTE PTR[BX] – Multiply AL with byte in DS pointed to by [BX]
- MUL FACTOR[BX] – Multiply AL with byte at effective address FACTOR [BX] if it is declared as type byte with DB. Multiply AX with word at effective address FACTOR[BX], if it is declared as type word with DW.
- MOV AX, MCAND_16 – Load 16-bit multiplicand into AX.
  MOV CL, MPLIER_8 – Load 8-bit multiplier into CL
  MOV CH, 00H – Set upper byte of CX to all 0's
  MUL CX – AX times CX; 32-bit result in DX and AX

IMUL - IMUL Source

This instruction multiplies a signed byte from source with a signed byte in AL or a signed word from some source with a signed word in AX. The source can be a register or a memory location

- IMUL BH — Multiply signed byte in AL with signed byte in BH; result in AX.
- IMUL AX — Multiply AX times AX; result in DX and AX.
- MOV CX, MULTIPLIER — Load signed word in CX
  MOV AL, MULTIPLICAND — Load signed byte in AL
  CBW — Extend sign of AL into AH
  IMUL CX — Multiply CX with AX; Result in DX and AX.

DIV - DIV Source

This instruction is used to divide an unsigned word by a byte or to divide an unsigned double word (32 bits) by a word. When the word is divided by a byte, the word must be in the AX register. All flags are undefined after a DIV instruction.

- DIV BL — Divide word in AX by byte in BL; Quotient in AL; remainder in AH.
- DIV CX — Divide down word in DX and AX by word in CX; Quotient in AX, and Remainder in DX.
- DIV SCALE[BX] — AX / if SCALE[BX] is of type byte; or (DX and AX) / if SCALE[BX] is of type word.

IDIV - IDIV Source

This instruction is used to divide a signed word by a signed byte, or to divide a signed double word by a signed word. All flags are undefined after an IDIV.

- IDIV BL — Signed word in AX / signed byte in BL.
- IDIV BP — Signed double word in DX and AX / signed word in BP.
- IDIV BYTE PTR[BX] — AX / byte at offset [BX] in DS

INC - INC Destination

The INC instruction adds 1 to a specified register or to a memory Location. AF, OF, PF, SF and ZF are updated, but CF is not affected.

- INC BL — Add 1 to contains of BL register.
- INC CX — Add 1 to contains of CX register.
- INC BYTE PTR[BX] — Increament byte in data segment at offset contained in BX.

- INC WORD PTR[BX] – Increament the word at offset of [BX] and [BX+1] in the data segment.
- INC TEMP – Increament byte or word named TEMP in the data Segment.
  Increament byte if MAX_TEMP declared with DB
  Increament word if MAX_TEMP declared with DW.
- INC PRICES[BX] – Increament element pointed to by [BX] in array PRICES
  Increament a word if PRICES is declared as an an array of words.
  Increament a byte if PRICES is declared as an array of byte.

DAA ~~DEC Destination~~

This instruction is used to make sure that result of adding two packed BCD numbers is adjusted to be a legal BCD number.

- Let AL = 59 BCD and BL = 35 BCD
  ADD AL, BL
  DAA
  AL: 8EH ; Lower nibble > 9, add 06H to AL
  AL : 94 BCD, CF: 0
- Let AL = 88 BCD, and BL = 49 BCD
  ADD AL, BL
  DAA
  AL : D1H ; AF : 1, add 06H to AL
  AL : D7H ; upper nibble > 9 ; add 60H to AL
  AL = 37 BCD, CF : 1

It updates AF, CF, SF, PF, and ZF, but OF is undefined.

DEC – DEC Destination

This instruction subtracts 1 from the destination word or byte. The destination can be a register or a memory location. AF, OF, SF, PF, and ZF are updated, but CF is not affected.

- DEC CL – Subtract 1 from content of CL register.
- DEC BP – Subtract 1 from content of BP register
- DEC BYTE PTR[BX] – Subtract 1 from byte at offset [BX] in DS.
- DEC WORD PTR[BP] – Subtract 1 from a word at offset [BP] in SS.
- DEC COUNT – Subtract 1 from byte or word named COUNT in DS
  Determine a byte if COUNT is declared with a DB
  Decreament a word if COUNT is declared with DW

DAS (DECIMAL ADJUST AFTER BCD SUBTRACTION

This instruction is used after subtracting one packed BCD number from another packed BCD number, to make sure the result is correct packed BCD.

- Let AL = 86 BCD, and BH = 57 BCD
  SUB AL, BH
  AL = 2FH; lower nibble > 9, subtract 06H from AL
  AL = 29 BCD, CF = 0

- Let AL = 49 BCD, and BH = 72 BCD
  SUB AL, BH
  DAS
  AL = D7H; upper nibble > 9, subtract 60H from AL
  AL = 77 BCD, CF = 1 (borrow is needed)

It updates AF, CF, SF, PF, and ZF, but OF is undefined.

CBW (CONVERT SIGNED BYTE TO SIGNED WORD)

This instruction copies the sign bit of the byte in AL to all the bits in AH. CBW does not effect any flag.

- Let AX = 0000 0000 1001 1011 (-155 decimal)
  CBW
  Convert signed byte in AL to signed word in AX
  AX = 1111 1111 1001 1011 (-155 decimal)

CWD (CONVERT SIGNED WORD TO SIGNED DOUBLE WORD

This instruction copies the sign bit of a word in AX to all the bits of the DX register. CWD affects no flags.

- Let DX = 0000 0000 0000 0000 and AX = 1111 0000 1100 0111 (-3897 decimal)
  CWD
  Convert signed word in AX to signed double word in DX:AX
  DX = 1111 1111 1111 1111
  AX = 1111 0000 1100 0111 (-3897 decimal)

AAA (ASCII ADJUST FOR ADDITION

Numerical data coming into a computer from a terminal is usually in ASCII code. In this the number 0 to 9 are represented by the ASCII codes 30H to 39H.

- Let, AL = 0011 0101 (ASCII 5), and BL = 0011 1001 (ASCII 9)
  ADD AL, BL
  AAA
  AL = 0110 1110 (6EH, which is incorrect BCD)
  AL = 0000 0100 (Unpacked BCD 4)
  CF = 1 indicates answer is 14 decimal

The AAA instruction works only on the AL register. The AAA instruction updates AF and CF, but OF, PF, SF and ZF are left undefined.

AAS (ASCII ADJUST FOR SUBTRACTION)

Numerical data coming into a computer from a terminal is usually in an ASCII code.

- Let AL: 0011 1001 (39H or ASCII 9), and BL: 0011 0101 (35H or ASCII 5)
  SUB AL, BL
  AAS
  AL: 0000 0100 (BCD 04), and CF=0
  AL: 0000 0100 (BCD 04), and CF:0 (no borrow required)

- Let, AL: 0011 0101 (35H or ASCII 5) and BL: 0011 1001 (39H or ASCII 9)
  SUB AL, BL
  AAS
  AL: 1111 1100 (-4 in 2's complement form), and CF=1
  AL: 00000100 (BCD 04), and CF:1 (borrow required)

The AAS instruction only works on the AL register. It updates ZF and CF; but OF, PF, SF, AF are left undefined.

AAM (BCD ADJUST AFTER MULTIPLY)

AAM instruction is used to adjust the product of two unpacked BCD digits in AX. It works only the operand in AL. AAM updates PF, SF, and ZF but AF, CF and OF are left undefined.

- Let AL: 0000 0101 (unpacked BCD 5, and BH: 0000 1001 (unpacked BCD 9)
  MUL BH
  AAM
  AX BH: 0000 0000 0010 1101 = 002DH
  AX: 0000 0100 0000 0101 = 0405H (unpacked BCD 45)

AAD (BCD-TO-BINARY CONVERT BEFORE DIVISION)

AAD converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL. AAD updates PF, SF and ZF; AF, CF and OF are left undefined.

- Let AX: 0607 (unpacked BCD for 67 decimal), and CH: 09H
  AAD
  DIV CH
  AX: 0043 (43H: 67 decimal)
  AL: 07; AH: 04; Flags undefined after DIV

If an attempt is made to divide by 0, the 8086 will generate a type 0 interrupt

LOGICAL INSTRUCTION

AND - AND Destination, Source

This instruction ANDs each bit in a source byte or word with the same numbered bit in a destination byte or word. The source and destination cannot both be memory locations. CF and OF are both 0 after AND. PF, SF, and ZF are updated by the AND instruction. AF is undefined. PF has meaning only for an 8-bit operand.

- AND CX, [SI] — AND word in DS at offset [SI] with word CX register; Result in CX register.
- AND BH, CL — AND byte in CL with byte in BH; Result in BH.
- AND BX, 00FFH — 00FFH Masks upper byte, leaves lower byte unchanged.

OR - OR Destination, Source

This instruction ORs each bit in a source byte or word with the same numbered bit in a destination byte or word. The source and destination cannot both be memory locations. CF and OF are both 0 after OR. PF, SF and ZF are updated by OR instruction. AF is undefined. PF has meaning only for an 8-bit operand.

- OR AH, CL — CL ORed with AH, result in AH, CL not changed.
- OR BP, SI — SI ORed with BP, result in BP, SI not changed
- OR SI, BP — BP ORed with SI, result in SI, BP not changed
- OR BL, 80H — BL ORed with immediate number 80H; sets MSB of BL to 1.
- OR CX, TABLE[SI] — CX ORed with word from effective address TABLE[SI]; content of memory is not changed.

XOR - XOR Destination, Source

This instruction Exclusive-ORs each bit in a source byte or word with the same numbered bit in a destination byte or word.

CF and OF are both 0 after XOR. PF, SF and ZF are updated. PF has meaning only for an 8-bit operand. AF is undefined.

- XOR CL, BH — Byte in BH exclusive-ORed with byte in CL. Result in CL. BH not changed.
- XOR BP, DI — Word in DI exclusive-ORed with word in BP. Result in BP. DI not changed.
- XOR WORD PTR [BX], 00FFH — Exclusive-OR immediate number 00FFH with word at offset [BX] in the data segment. Result in memory location [BX]

CMP- CMP Destination, Source

This instruction compares a byte / word in the specified source with a byte/ word in specified destination. AF, OF, SF, ZF, PF, and CF are updated by the CMP instruction. For the instruction CMP CX, BX, the values of CF, ZF and SF will be as follows:

	CF	ZF	SF	
CX = BX	0	1	0	Result of subtraction is 0
CX > BX	0	0	0	No borrow required, so CF = 0
CX < BX	1	0	1	Subtraction requires borrow, so CF = 1

- CMP AL, 01H — Compare immediate number 01H with byte in AL
- CMP BH, CL — Compare byte in CL with byte in BH.
- CMP CX, TEMP — Compare word in DS at displacement TEMP with word at CX.
- CMP PRICES [BX], 49H — Compare immediate number 49H with byte at offset [BX] in array PRICES

TEST- TEST Destination, Source

This instruction ANDs the byte/ word in specified source with the byte/ word in the specified destination. Flags are updated, but neither operand is changed.

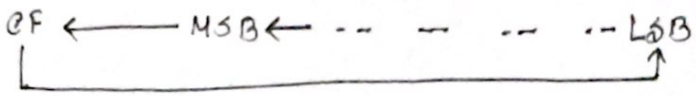
CF, and OF are both 0's after TEST. PF, SF, and ZF will be updated to show the result of the destination. AF is undefined.

- TEST AL, BH — AND BH with AL. No result stored; Update PF, SF, ZF.
- TEST CX, 0001H — AND CX with immediate number 0001H; No result stored; Update PF, SF, ZF.
- TEST BP, [BX] [DI] — AND words are offset [BX] [DI] in DS with word in BP. No result stored. Update PF, SF, and ZF

ROTATE AND SHIFT INSTRUCTIONS

RCL- RCL Destination, Count

This instruction rotate all the bits in a specified word or byte some number of bit position to the left.



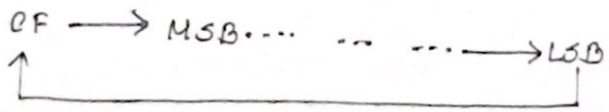
For multibit rotates, CF will contain the bit most recently rotated out of MSB

RCL affects only CF and OF. OF will be a 1 after single bit RCL if the MSB was changed by the rotate. OF is undefined after multi-bit rotate.

- RCL DX, 1 — Word in DX 1 bit left, MSB to CF, CF to LSB.
- MOV CL, 4 — Load the number of bit position to rotate into CL
  RCL SUM[BX], CL — Rotate byte or word at effective address SUM[BX] 4 bits left.
  Original bit 4 now in CF, original CF now in bit 3.

RCR- RCR Destination, Count

This instruction rotates all the bits in a specified word or byte some number of bit positions to the right.



For multi-bit rotate, CF will contain the most recently rotated out of the LSB.

RCR effects only CF and OF. OF will be a 1 after a single bit RCR if the MSB was changed by the rotate. OF is undefined after multi-bit rotate.

- RCR BX, 1 — Word in BX right 1 bit, CF to MSB, LSB to CF.
- MOV CL, 4 — Load CL for rotating 4-bit position
- RCR BYTE PTR[BX], 4 — Rotate the byte at offset[BX] in DS 4 bit positions right.
  CF = original bit3, Bit 4- original CF

SAL - SAL Destination, Count

SHL - SHL Destination, Count

SAL and SHL are two mnemonics for the same instruction. This instruction shifts each bit in a specified destination some number of bit positions to the left.

CF ← MSB ← -- -- -- LSB ← 0

The flags are affected as follow: CF contains the bit most recently shifted out from MSB. For a count of 1, OF will be 1 if CF and the current MSB are not same. For multiple-bit shifts, OF is un-defined. SF and ZF will be updated the reflect the condition of the destination. PF will have meaning only for an operand in AL, AF is undefined.

- SAL BX, 1 — Shift word in BX 1 bit position left, 0 in LSB.
- MOV CL, 02h — Load desired number of shift in CL
  SAL BP, CL — Shift word in BP left CL bit positions, 0 in LSBs.
- SAL BYTE PTR[BX], 1 — Shift byte in DX at offset [BX] 1 bit position left, 0 in LSB

SAR - SAR Destination, Count

This instruction shifts each bit in the specified destination some number of bit position to the right.

MSB → MSB -- -- -- → LSB → CF

The flags are affected as follow: CF contains the bit most recently shifted in from LSB. For a count of one, OF will be 1 if the two MSBs are not the same. After a multi-bit SAR, OF will be 0. SF and ZF will be updated to show the condition of the destination. PF will have meaning only for an 8-bit destination. AF will be undefined after SAR.

- SAR DX, 1 — Shift word in DI one bit position right, new MSB = old MSB.
- MOV CL, 02H — Load desired number of shifts in CL.
- SAR WORD PTR[BP], CL — Shift word at offset [BP] in stack segment right by two bit position, the two MSBs are now copies of original LSB.

SHR - SHR Destination, Count

This instruction shifts each bit in the specified destination some number of bit position to the right

0 ⟶ MSB- - - - - - - - ⟶ LSB ⟶ CF

The flags are affected by SHR as follow: CF contains the bit most recently shifted out from LSB. For a count of one, OF will be 1 if the two MSBs are not both 0's. For multiple-bit shifts, OF will be meaningless. SF and ZF will be updated to show the condition of the destination. PF will have meaning only for an 8-bit destination. AF is undefined.

- SHR BP, 1 — shift word in BP one bit position right, 0 in MSB.
- MOV CL, 08H — Load desired number of shift into CL.
  SHR BYTE PTR [BX] — Shift byte in DS at offset [BX] 3 bits right; 0's in 3 MSBs.

TRANSFER - OF - CONTROL - INSTRUCTIONS

JMP (UNCONDITIONAL JUMP TO SPECIFIED DESTINATION

This instruction will fetch the next instruction from the location specified in the instruction rather than from the next location after the JMP instruction. The JMP instruction does not affect any flag.

- JMP CONTINUE — This instruction fetches the next instruction from address at label CONTINUE.
- JMP BX — This instruction replaces the content IP with the content of BX.
- JMP WORD PTR [BX] — This instruction replaces IP with word from a memory location pointed to by BX in DX.
- JMP DWORD PTR [SI] — This instruction replaces IP with word pointed to by SI in DS. It replaces CS with a word pointed by SI+2 in DS.

JBE / JNA (JUMP IF BELOW OR EQUAL / JUMP IF NOT ABOVE)

If, after a compare or some other instructions which affect flags, either the zero flag or the carry flag is 1, this instruction will cause execution to jump to a label given in the instruction. If CF and ZF are both 0, the instruction will have no effect on program execution.

- CMP AX, 4371H — Compare (AX - 4371H)
  JBE NEXT — Jump to label NEXT if AX is below or equal to 4371H
- CMP AX, 4371H — Compare (AX - 4371H)
  JNA NEXT — Jump to label NEXT if AX not above 4371H.

JG / JNLE (JUMP IF GREATER / JUMP IF NOT LESS THAN OR EQUAL)

This instruction is usually used after a compare instruction. The instruction will cause a jump to the label given in the instruction if the zero flag is 0 and the carry flag is the same as the overflow flag.

- CMP BL, 39H — Compare by subtracting 39H from BL.
  JG NEXT — Jump to label NEXT if BL more positive than 39H.
- CMP BL, 39H — Compare by subtracting 39H from BL
  JNLE — Jump to label NEXT if BL is not less than or equal to 39H.

JL / JNGE (JUMP IF LESS THAN / JUMP IF NOT GREATER THAN OR EQUAL)

This instruction is usually used after a compare instruction. The instruction will cause a jump to the label given in the instruction if the sign flag is not equal to the overflow flag.

- CMP BL, 39H — Compare by subtracting 39H from BL
  JL AGAIN — Jump to label AGAIN if BL more negative than 39H.
- CMP BL, 39H — Compare by subtracting 39H from BL
  JNGE AGAIN — Jump to label AGAIN if BL not more positive than or equal to 39H.

JLE/JNG (JUMP IF LESS THAN OR EQUAL / JUMP IF NOT GREATER)

This instruction is usually used after a compare instruction. The instruction will cause a jump to the label given in the instruction if zero flag is set, or if the sign flag not equal to over flow flag.

- CMP BL, 39H — Compare by subtracting 39H from BL
  JLE NEXT — Jump to label NEXT if BL more negative than or equal to 39H.
- CMP BL, 39H — Compare by subtracting 39H from BL
  JNG NEXT — Jump to label NEXT if BL not more positive than 39H.

JE/JZ (JUMP IF EQUAL / JUMP IF ZERO)

This instruction is usually used after a compare instruction If the zero flag is set, then this instruction will cause a jump to the label given in the instruction

- CMP BX, DX — Compare (BX-DX)
  JE DONE — Jump to DONE if BX=DX
- IN AL, 30H — Read data from port 8FH
  SUB AL, 30H — Subtract the minimum value.
  JZ START — Jump to label START if the result of subtraction is 0.

JNE/JNZ (JUMP NOT EQUAL/ JUMP IF NOT ZERO)

This instruction usually used after a compare instruction. If the zero flag is 0, then this instruction will cause a jump to the label given in the instruction.

- IN AL, 0F8H — Read data value from port
  CMP AL, 72 — Compare (AL-72)
  JNE NEXT — Jump to label NEXT if AL≠72
- ADD AX, 0002H — Add count factor 0002H to AX
  DEC BX — Decreament BX
  JNZ NEXT — Jump to label NEXT if BX≠0

STACK RELATED INSTRUCTIONS

PUSH- PUSH Source

The PUSH instruction decrements the stack pointer by 2 and copies a word from a specified source to the location in the stack segment to which the stack pointer points. PUSH can be used to save data on the stack so that it will not destroyed by a procedure. This instruction doesn't affect any flag.

- PUSH BX — Decreament SP by 2, copy BX to stack.
- PUSH DS — Decreament SP by 2, copy DS to stack
- PUSH BL — Illegal; must push word
- PUSH TABLE[BX] — Decreament SP by 2, and copy word from memory in DS at EA = TABLE + [BX] to stack.

POP- POP Destination

The POP instruction copies a word from the stack location pointed to by the stack pointer to a destination specified in the instruction. The POP instruction does not affect any flag.

- POP DX — Copy a word from top of the stack to DX; increament SP by 2.
- POP DS — Copy a word from top of the stack to DS; increament SP by 2.
- POP TABLE [DX] — Copy a word from top of the stack to memory in DS with EA = TABLE + [BX]; increament SP by 2.

INPUT- OUTPUT INSTRUCTIONS

IN- IN Accumulator, Port

The IN instruction copies data from a port to AL or AX register. If an 8-bit port is read, the data will go to AL. If a 16-bit port is read, the data will go to AX.

- IN AL, 0C8H — Input a byte from port 0C8H to AL.
- IN AX, 34H — Input a word from port 34H to AX.

For the variable-port form of the IN instruction, the port address is loaded into the DX register before the IN instruction.

- MOV DX, 0FF78H — Initialize DX to points to port
  IN AL, DX — Input a byte from 8-bit port 0FF78H to AL
  IN AX, DX — Input a word from 16-bit port 0FF78H to AX.

The IN instruction does not change any flag.

OUT- OUT Port, Accumulator

The OUT instruction copies a byte from AL or a word from AX to the specified port. The OUT instruction has two possible forms, fixed port and variable port.

For the fixed port form, the 8 bit address is specified directly in the instruction.

- OUT 3BH, AL — Copy the content of AL to port 3BH
- OUT 2CH, AX — Copy the content of AX to port 2CH

For variable port form of the OUT instruction, the content of AL or AX will be copied to the port at an address contained in DX.

- MOV DX, 0FFF8H — Load desired port address in DX.
  OUT DX, AL — Copy content of AL to port FFF8H
  OUT DX, AX — Copy content of AX to port FFF8H

The OUT instruction does not affect any flag.

8086 ASSEMBLER DIRECTIVES

ENDS (END SEGMENT)

This directive is used with the name of segment to indicate the end of that logical segment.

- CODE SEGMENT – Start of logical segment containing code instruction statements.

  CODE ENDS – End of segment named CODE

END (END PROCEDURE)

The END directive is put after the last statement of a program to tell the assembler that this is the end of the program module.

DW (DEFINE WORD)

The DW directive is used to tell the assembler to define a variable of type word or to reserve storage location of type word in memory.

- WORDS DW 1234H, 3456H – Declare an array of 2 words and initialize them with the specified values.
- STORAGE DW 100 DUP(0) – Reserve an array of 100 words of memory and initialize all 100 words with 0000. Array is named as STORAGE
- STORAGE DW 100 DUP(?) – Reserve 100 words of storage in memory and give it the name STORAGE, but leave the words un-initialized.

PROC (PROCEDURE)

The PROC directive is used to identify the start of a procedure. The PROC directive follows a name given to the procedure. After the PROC directive, the term near or the term far is used to specify the type of the procedure.

ENDP (END PROCEDURE)

The directive is used along with the name of the procedure to indicate the end of a procedure to the assembler.

- SQUARE_ROOT PROC – Start of procedure

  SQUARE_ROOT ENDP – End of procedure.

LABEL

As an assembler assembles a section of a data declaration or instruction statements, it uses a location counter to be keep track of how many bytes it is from the start of the segment at any time. The LABEL directive is used to give a name to the current value in the location counter.

Here's how we use the LABEL directive for a jump address.

```
• ENTRY_POINT LABEL FAR      - Can jump to here from another
                                segment.
  NEXT: MOV AL, BL           - Can not do a far jump directly
                                to a label with a colon.
```

The following examples shows how we use the label directive for a data reference

```
• STACK_SEG SEGMENT STACK
  DW 100 DUP(0)              - Set aside 100 words for stack.
  STACK_TOP LABEL WORD       - Give name to next location after
                                last word in stack.
  STACK_SEG ENDS
```

To initialize stack pointer, we use MOV SP, OFFSET STACK_TOP

INCLUDE (INCLUDE SOURCE CODE FROM FILE)

This directive is used to tell the assembler to insert a block of source code from the named file into the current source module.