LAMISA TASNIM
ID: 1811472642

Homework - 4

8086 Instruction Set

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. The source & destination cannot both be memory locations. They must both be of the same type (bytes or words). MOV instruction does not affect any flag.

→ MOV CX, 037 AH ; put immediate number 037AH to CX
→ MOV BL, [437 AH] ; copy byte in DS at offset 437AH 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

LAMISA TASNIM
ID: 1811472642

→ MOV RESULT [BP], AX ; Copy AX to two memory locations;
AL to the first location,
AH to the second location;
EA of the first memory
location is sum of the
displacement represented by
RESULTS & 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.

LEA – LEA Register, Source:

This instruction determines the offset of the variable or memory location named as the source & puts this offset in the indicated 16-bit register. LEA does not affect any flag.

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

LAMISA TASNIM
ID: 1911472642

ARITHMETIC INSTRUCTIONS

ADD - ADD Destination, Source:
ADC - ADC Destination, Source:

These instructions add a number from some source to a number in some destination & put the result in the specified destination. The ADC also adds the status of the carry flag to the result. The source may be an immediate number, a register, or a memory location. The destination may be a register or a memory location. The source & the destination in an instruction cannot both be memory locations. The source & the destination must be of same type (bytes or words). If you want to add a byte to word, you must copy the byte to a word location to fill the upper byte of the word with 0's before adding. Flags affected: AF, CF, OF, S

→ ADD AL, 74H ; Add immediate number 74H to content of AL, Result i
→ ADC CL, BL ; Add content of BL plus carry status to conte
→ ADD DX, BX ; Add content of BX to content of DX
→ ADD DX, [SI] ; Add word from memory at offset
→ ADC AL, PRICES [BX] ; Add byte from effective address
→ ADD AL, PRICES [BX] ; Add content of memory at effectiv address PRICES [BX] to AL

LAMISA TASNIM
ID: 181147264

SUB - SUB Destination, Source:

SBB - SBB Destination, Source:

These instructions subtract the number in some source from the number in some destination & put the result in the destination. The SBB instruction also subtracts the content of carry flag from the destination. The source may be an immediate number, a register or memory location. The destination can also be a register or a memory location. However, the source & the destination cannot both be memory location. The source & the destination must both be of the same type (bytes or words). If you want to subtract a byte from a word, you must first move the byte to a word location such as a 16-bit register & fill the upper byte of the word with 0's. Flags affected: AF, CF, OF, PF, SF, ZF.

→ SUB CX, BX ; CX - BX; Result in CX

→ SBB CH, AL ; Subtract content of AL & content of CF from content of CH. Result in CH

→ SUB AX, 3427H ; Subtract immediate number 3427H from AX.

LAMISA TASNIM
ID: 1811472642

→ SBB BX, [3427H] ; Subtract word at displacement 3427H in DS & 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[B & status of CF from CX.

→ SBB TABLE [BX], CX ; Subtract CX & status of CF from word in memory at effective address TABLE [BX].

TAMISA TASNIM
ID: 1811472642

MUL - MUL Source:

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

If you want to multiply a byte with a word, you must first move the byte to a word location such as an extended register & fill the upper byte of the word with all 0's. You cannot use the CBW instruction for this, because the CBW instruction fills the upper byte with copies of most significant bit of the lower byte.

LAMISA TASNIM
ID: 191647262

→ MUL BH ; Multiply AL with BH ; result in AX

→ 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 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 & AX.

LAMISA TASNIM
ID: 181114726 42

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 a word is divided by a byte, the word must be in the AX register. The divisor can be in a register or a memory location. After the division, AL will contain the 8-bit quotient, & AH will contain the 8-bit remainder. When a double word is divided by a word, the most significant word of the double word must be in DX, & the least significant word of the double word must be in AX. After the division, AX will contain the 16-bit quotient & DX will contain 16-bit remainder. If an attempt is made to divide by 0 or if the quotient is too large to fit in the destinatio (greater than FFH/FFFFFH), the 8086 will generate a type 0 interrupt. All flags are undefined after a DIV instruction.

LAMISA TASNIM
ID: 1844732642

If you want to divide a byte by a byte, you must first put the divided byte in AL & AH with all 0's. Likewise, if you want to divide a word by another word, then put the dividend word in AX & fill DX with all 0's.

→ DIV BL ; Divide word in AX by byte in BL; Quotient in AL, remainder in AH.

→ DIV CX ; Divide down word in DX & AX by word in CX;

→ DIV SCALE[BX] ; AX / (byte at effective address SCALE[BX]) is of type byte; or (DX & AX) / (word at effective address SCALE[BX] if SCALE[BX] is of type word.

INC - INC Destination:

The INC instruction adds 1 to a specified register or to a memory location. AF, OF, PF, SF & ZF are updated, but CF is not affected. This means that if an 8-bit destination containing FFH or a 16-bit destination containing FFFFH is incremented, the result will be all 0's with no carry.

LAMISA TASNIM
ID: 18114726 42

→ INC BL ; Add 1 to contains of BL register

→ INC CX ; Add 1 to contains of CX register

→ INC BYTE PTR [BX] ; Increment byte in data segment at offset contained in BX.

→ INC WORD PTR [BX] ; Increment the word at offset of [BX] & [BX+1] in the data segment

→ INC TEMP ; Increment byte or word named TEMP in the data segment.
Increment byte if MAX-TEMP declared with DB.
Increment word if MAX-TEMP is declared with DW.

→ INC PRICES [BX] ; Increment element pointed to by [BX] in array PRICES.
Increment a word if PRICES is declared as an array of words.
Increment a byte if PRICES is declared as an array of bytes.

LAMISA TASNIM
ID: 1811472642

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 & ZF are updated, but CF is not affected. This means that if an 8-bit destination containing 00H or 16-bit destination containing 0000H is decremented, the result will be FFH or FFFFH with no carry (borrow).

→ DEC CL ; Subtract 1 from content of CL register

→ DEC BP ; Subtract 1 from byte at offset [BX] in DS.

→ 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.

Decrements byte if COUNT is declared with a DB

Decrement a word if COUNT is declared with DW.

LAMISA TASNIM
ID: 1811472642

DAA (DECIMAL ADJUST AFTER BCD ADDITION):

This instruction is used to make sure the result of adding two packed BCD numbers is adjusted to be a legal BCD number. The result of the addition must be in AL for DAA to work correctly. If the lower nibble in AL after an addition is greater than 9 or AF was set by the addition, then the DAA instruction will add 6 to lower nibble in AL. If the result in the upper nibble of AL is now greater than 9 or if the carry flag was set by the addition or correction, then the DAA instruction will add 60H to AL.

→ Let AL = 59 BCD, & BL = 35 BCD

```
ADD  AL, BL      ; AL = 8EH ; lower nibble > 9, add 06H to AL
DAA              AL = 94 BCD, CF = 0
```

→ Let AL = 88 BCD, & BL = 49 BCD

```
ADD  AL, BL      ; AL = D1H ; AF = 1, add 06H to AL
DAA              AL = D7H ; upper nibble > 9, add 60H to AL
                 AL = 37 BCD, CF = 1
```

The DAA instruction updates AF, CF, SF, PF & ZF. But OF is undefined.

LAMISA TASNIM
ID: 18114726 42

AAA (ASCII ADJUST FOR ADDITION):

Numerical data coming into a computer from a terminal is usually in ASCII code. In this code, the numbers 0 to 9 are represented by the ASCII codes 30H to 39H. The 8086 allows you to add the ASCII codes for two decimal digits without masking off the "3" in the upper nibble of each. After the addition, the AAA instruction is used to make sure the result is the correct unpacked BCD.

→ Let AL = 00110101 (ASCII 5), & BL = 0011 1001 (ASCII 9)

```
ADD  AL, BL
AAA
```

AL = 0111 1110 (6EH)

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 & CF; but OF, PF, SF & ZF are left undefined.

LAMISA TASNIM
ID: 1811472642

LOGICAL INSTRUCTIONS

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 result is put in the specified destination. The content of the specified source is not changed.

The source can be an immediate number, the content of a register, or the content of a memory location. The destination can be a register or a memory location. The source & the destination cannot both be memory locations. CF & OF are both 0 after AND.

PF, SF & ZF are updated by the AND instruction. AF is undefined. PF has meaning only for an 8-bit operand.

LAMISA TASNIM
ID: 1811472642

→ AND CX, [SI] ; AND word in DS at offset [SI] with word in CX register; Result in CX.

→ 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 result is put in the specified destination. The content of the specified source is not changed.

The source can be an immediate number, the content of a register, or the content of a memory location. The destination can be ~~greater~~ register or memory location. The source & destination cannot both be memory location. CF & OF are both 0 after OR. PF, SF & ZF are updated by the OR instruction. AF is undefined. PF has meaning only for an 8-bit operand.

TAMISA TASNIM
ID: 1811472042

→ 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 change

→ 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. The result is put in the specified destination. The content of the specified source is not changed.

The source can be an immediate number, the content of a register, or the content of a memory location. The destination can be a register or a memory location. The source & destination cannot both be memory locations. CF & OF are both 0 after XOR, PF, SF & ZF are updated. PF has meaning only for an 8-bit operand. AF is undefined.

LAMISA TASNIM
ID: 1811472642

→ 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 the specified destination. The source can be an immediate number, a register, or a memory location. The destination can be a register or a memory location. However, the source & the destination cannot both be memory locations. The comparison is actually done by subtracting the source byte or word from the destination byte or word. The source & the destination are not changed, but the flags are set to indicate the results of the comparison. AF, OF, SF, ZF, PF & CF are updated by CMP instruction. For the instruction CMP CX, BX, the values of CF, ZF & SF will be as follows:

LAMISA TASNIM
ID: 1811472642

	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 A

→ 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 ~~ADDS~~ ANDs the byte/word in the specified source with the byte/word in the specified destination. Flags are updated, but neither operand is changed. The test instruction is often used to set flags before a conditional jump instruction.

LAMISA TASNIM
ID: 181147642

The source can be an immediate number, the content of a register, or the content of a memory location. The destination can be a register or a memory location. The source & the destination cannot both be memory locations. CF & OF are both 0's after TEST. PF, SF & ZF will be updated to show the results of the destination. AF is to be 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

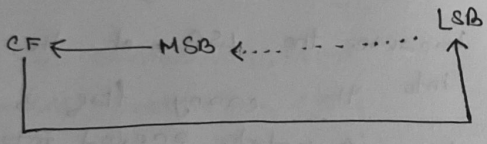
→ TEST BP, [BX][DI]; AND word are offset [BX][DI] in DS with word in BP. No result stored. Update PF, SF, ZF.

LAMISA TASNIM
ID: 1811472642

ROTATE & SHIFT INSTRUCTIONS

#RCL - RCL Destination, ~~Source~~ Count :

This instruction rotates all the bits in a specified word or byte some number of bit positions to the left. The operation circular because the MSB of the operand in rotated into the carry flag & the bit in the carry flag is rotated around into LSB of the operand.



For multi-bit rotates CF will contain the bit most recently rotated out of the MSB.

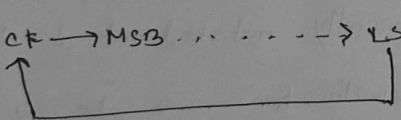
The destination can be a register or a memory location. RCL affects only CF & OF. OF will be a 1 after a single bit RCL if the MSB was changed by the rotate. OF is undefined after the multi-bit rotate.

LAMICA TASNIM
ID: 1811472042

→ RCL BX, 1 ; Word in DX 1 bit left, MSB to CF, CF to LS
→ MOV CL, 4 ; load the number of bit position to rotate i
RCL SUM[BX], CL; Rotate byte or word at effecti
address SUM[BX] 4 bits left
Original bit 4 now in CF, original
CF now in bit 3.

RCR – RCR a Destination, Count:

This instruction rotates all the bits in a specified word or byte some number of bit positions to the right. The operation circular because the LSB of the operand is rotated into the carry flag & the bit in the carry flag is rotate around into MSB of the operand



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

The destination can be a registter or a memory location. If you want to rotate the operand by one bit position, you can specify this by putting 1 in the count position of the instruction.

LAMISA TASNIM
ID:2211472642

RCR affects only CF & OF. OF will be a 1 after a single bit RCR if the MSB was changed by the rotate. OF is undefined after the 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 positions

RCR BYTE PTR[BX], 4 : Rotate the byte at offset [BX] in DS 4 bit positions right CF = original bit 3, Bit 4 = original CF.

SAL - SAL Destination, Count
SHL - SHL Destination, Count :

SAL & SHL are two mnemonics for the same instructions. This instruction shifts each bit in the specified destination some number of bit positions to the left. As a bit is shifted out of the LSB operation, a 0 is put in the LSB position. The MSB will be shifted into CF.

LAMISA TASNIM
ID: 1811432642

CF ← MSB ← ..... ← LSB ← 0

The destination operand can be a byte or a word. It can be in a register or in a memory. If you want to shift the operand by one bit position, you can specify this by putting a 1 in the count position of the instruction.

CF, ZF, SF & OF are affected. AF is undefined.

→ SAL BX, 1 ; shift word in BX 1 bit position left, 0 in LSB

→ MOV CL, 02H ; Load desired number of shifts in CL
SAL BP, CL ; shift word in BP left CL bit positions, 0 in LSB

→ SAL BYTE PTR [BX], 1; shift byte in DX at offset [BX]

SAR – SAR Destination, Count: 1 bit position left, 0 in LSB.

This instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted out of the MSB position, a copy of the old MSB is put in the MSB position. In other words, sign bit is copied into the MSB. The LSB will be shifted into CF.

LANISA TASNIM
ID: 1811472642

MSB ⟶ NSB . . . . . → LSB ⟶ CF

The destination operand can be a byte or a word. It can be in a register or in a memory location. If you want to shift the operand by one bit position, you can specify this by putting a 1 in the count position of the instruction.

CF, OF, SF & ZF will be affected.

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
~~MOV~~ WORD PTR [BP], CL; shift word at offset [BP] in stack segment right by two bit positions, the two MSBs are now copies of original LSB.

LAMISA TASNIM
ID: 1811472642

SHR - SHR Destination, count:

This instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted out of the MSB position, a 0 is put in it's place. The bit shifted out the LSB position goes to CF. Bits shifted into CF previously will be lost.

0 ⟶ MSB .....> LSB ⟶ CF

The destination operand can be a byte or a word in a register or in a memory location. If you want to shift the operand by one bit position, you can specify this by putting 1 in the count position of the instruction

AF is undefined.

→ SHR BP, 1 ; Shift word in BP one bit position right, 0 in MSB.

→ MOV CL, 03H ; Load desired number of shifts in CL
SHR BYTE PTR [BX] ; Shift byte in DS at offset [BX] 3 bit right; 0's in 3 MSBS.

LAMISA TASNIM
ID: 1811472642

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 JMP instruction. If the destination is in the same code segment as the JMP instruction, then only the instruction pointer will be changed to get the destination location. This is referred to as a near jump. If the destination for the jump instruction is in a segment with a name different from that of the segment containing the JMP instruction. The JMP instruction does not affect any flag.

→ JMP CONTINUE
→ JMP BX
→ JMP WORD PTR [BX]
→ JMP DWORD PTR [SI]

LAMISA TASNIM
ID: 1811473242

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

If, after a compare or some other instructions which affect the 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 & 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 is 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 & 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 than 39H
→ CMP BL, 39H ; Compare by subtracting 39H from BL
JNLE NEXT ; Jump to label NEXT if BL is not less than or equal to 39H.

LAMISA TASNIM
ID: 1211472642

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, 29H ; 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 the zero flag is set, or if the sign flag not equal to the overflow flag.

→ CMP BL, 39H ; Compare by subtracting 39H from BL
JLE NEXT ; Jump to label NEXT if BL more negative or equal 39H

→ CMP BL, 39H ; Compare by subtracting 39H from BL
JNG NEXT ; Jump to label NEXT if BL not more positive than 39H.

LAMIA TASNIM
ID : IR11473C642

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

This instruction is used after a comparc 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 0.

INPUT-OUTPUT INSTRUCTIONS

IN-IN Accumulator, Port

The IN instruction copies data from a port to the 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.

The IN instruction has two possible formats, fixed port & variable port. For fixed port type, the 8-bit port address is specified directly in the instruction. With this form, any one of 256 possible ports can be addressed.

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

LAMISA TASNIM
ID: 1811472642

For the variable-port form of the IN instruction, the port address is loaded into the DX register before the IN instruction. Since DX is a 16-bit register, the port address can be any number between 0000H & FFFFH. Therefore, upto 65,536 ports are addressable in this mode.

→ MOV DX, 0FF78H ; Initialize DX to point 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 variable-port IN instruction has advantage that the port address can be computed or dynamically determined in the program. Suppose, for example, than an 8086-based computer needs to input data from 10 terminals, each having its own port address. Instead of having a separate procedure to input data from each port, you can write one generalized input procedure & simply pass the address of the desired port.

The IN instrion does not change any flags.

LAMISA TASNIM
ID: 1811472642

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 & variable port.

For the fixed port, the 8-bit port address is specified directly in the instruction. With this form, any one of the 256 possible ports can be addressed.

→ OUT 3BH, AL; Copy the content of AL to port 3BH

→ OUT 2CH, AX; Copy the content of AX to port 2CH

→ 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.

LAMISA TASNIM
ID: 1811472642

STACK RELATED INSTRUCTIONS

PUSH - PUSH Source

The PUSH instruction decrements the stack pointer by 2 & copies a word from a specified source to the location in the stack segment to which the stack pointer points. The source of the word can be general-purpose register, segment register, or memory. The stack segment register & the stack pointer must be initialized before this instruction can be used. PUSH can be used to save data on the stack so that it will not be destroyed by a procedure. This instruction does not affect any flag.

→ PUSH BX ; Decrement SP by 2; copy BX to stack
→ PUSH DS; Decrement SP by 2; Copy DS to stack
→ PUSH BL; Illegal; must push a word
→ PUSH TABLE[BX]; Decrement SP by 2, & copy word from memory in DS at EA = TABLE + [BX] to stack

ZAMKA TASNIM
ID : 1811472642

POP – POP Destination

The POP instruction copies a word from the stack location pointed by the stack pointer to a destination specified in the instruction. The destination can be a general-purpose register, a segment register, or a memory location. The data in stack is not changed. After the word is copied to the specified destination, the stack pointer is automatically incremented by 2 to the point to the next word on the stack. The POP instruction does not affect any flag.

→ POPDX; Copy a word from top of stack to DX; increment SP by 2

→ POP DS; Copy a word from top of stack to DS; increment SP by 2

→ POP [TABLE [BX]; Copy a word from TOP of stack to memory in DS with EA = TABLE + [BX]; Increment SP by 2.

LAMIRA TASNIM
ID: IR11472647

8086 ASSEMBLER DIRECTIVES

ENDS (END SEGMENT)

This directive is used with the name of a 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. The assembler will ignore any statements after an END directive, so you should make sure to use only one END directive at the very end of your program module. A carriage return is required after the END directive.

LAMISA TASNIM
ID: IRN4732642

DW (DEFINE WORD)

The DW directive is used to tell the assembler to define a variable of type word or to reserve storage locations of type word in memory. The statement MULTIPLIER DW 437AH, for example, declares a variable of type word named MULTIPLIER & initialized with the value 437AH when the program is loaded into memory to be run.

→ WORDS DW 1234H, 34564

→ STORAGE DW 100 DUP(0)

→ STORAGE DW 100 DUP(?)

PROC (PROCEDURE)

The PROC directive is used to identify the start of a procedure. The PROC directive follows a name you give the procedure. After the PROC directive, the term near or the term far is used to specify the type of the procedure. The statement DIVIDE PROC FAR, for example, identifies the start of a procedure named DIVIDE & tells the assembler that the procedure is far. The PROC directive is used with the ENDP directive to "bracket" a procedure.

LAMISA TASNIM
ID: 1811472642

ENDP (END PROCEDURE):

The directive is used ~~also~~ along with the name of the procedure to indicate the end of a procedure to the assembler. The directive, together with the procedure directive, PROC, is used to "Bracket" a procedure.

→ SQUARE_ROOT PROC ; start of procedure
— SQUARE_ROOT ENDP ; End of Procedure.

LABEL:

As an assembler assembles a section of a data declarations or instructions statements, it uses a location counter to be keep track of how many byters it is from the start of a segment at any time. The LABEL directive is used to give a name to the current value in the location counter. The LABEL direction must be followed by a term that specifies the type you want to associate with that name. If the label is going to be used as the destination for a jump or a call, then the label must be specified as type near or type far. If the label is going to be used to reference a data item,

LAMIRA TASNIM
ID: 1811472692

then the label must be specified as type byte, type word, or type double word. Here's how we use the LABEL directive for a jump address.

→ ENTRY_POINT LABEL FAR; can jump to here from another segme
NEXT: MOV AL, BL ; can not do a far jump directly to a label with a colon.

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

→ STACK_SEG SEGMENT STACK
DW 100. DUP(0)
STACK_TOP LABEL WORD
STACK_SEG ENDS

To initialize stack pointer, 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.

The End
— x —