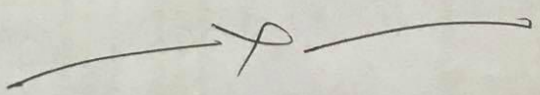
> Entry_POINT LABEL FAR can jump to here from another segment

> NEXT: MOV AL, BL Can not do a far jump directly to a label with a colon.

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.



and tells the assembler that the procedure is far. The PROC directive is used with the ENDP directive to "bracket" a 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. 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

If the label is going to be used to reference a data item, 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.

- WORDS DW 1234H, 3456H Declare an array of 2 words and initialized 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 word 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 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.

- WORDS DW 1234H, 3456H Declare an array of 2 words and initialized 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 word 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 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.

- CODE SEGMENT start of logical segment conta
  CODE ENDS code instruction statements 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.

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 and initialized with the value 437AH when the program is loaded into memory to be run.

example, identifies the

STACK RELATED INSTRUCTIONS

PUSH - PUSH Source

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

POP - POP Destination

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

ENDS (END SEGMENT)

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

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 case 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
JNGE Again Jump to label AGAIN if BL not more positive than or equal to 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

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

If after a compare or some other instructions which effects 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 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)

```
CMP BL, 39H        Compare by subtraction 39H from BL
JGE NEXT           JUMP to label NEXT if BL more positive than or equal to 39H

CMPBL, 39H         Compare by subtracting
JNL NEXT           39H from BL
JNLE NEXT          Jump to label NEXT if BL is not
```

JMP (Unconditional JUMP to SPECIFIED DESTINATION

➤ JMP CONTINUE : This instruction fetches the next instruction from Address at label CONTINUE. If the label is in the same segment, an offset coded as part of the instruction will be added to the instructions pointer to produce the new fetch address. If the label is another segment, then IP and CS will be replaced with value coded in part of the instruction.

➤ JMP BX : This instruction replaces the content of IP with the content of BX. BX must first be loaded with the offset of the destination instruction in CS. This is a near jump.

➤ JMP WORD PTR [BX] This instruction replaces IP with word from a memory location pointed to by BX in DX. This is an indirect near jump.

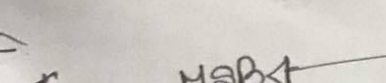
SAR - SAR Destination, Count

This instruction shifts each bit in the specified 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

MSB → MSB . . . LSB → CF

The flag are affected as follows:
CF contains the bit most recently shifted in form 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 Load
- MOV CL, 04 desired number of shifts in CL.



The flags are affected as follows: CF contains the bit most recently shifted out from MSB. For a count of one, OF will be 1 if CF and the current MSB are not the same. For multiple bit shifts, OF is undefined. SF and ZF will be updated to 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 1bit position left, 0 is LSB
- MOV CL, 02h — Load desired number of shifts in CL
  SAL BP, CL — shift word in BP left CL bit positions, 0 is LSBS
- SAL BYTE PTR[BX], 1 — shift byte in DX at offset [BX] 1bit position left, 0 is LSB.

» MOV CL, 4 Load CL for rotating 4 bit position Rotate
RCR BYTE PTR [BX], 4 the byte at offset [BX] in DS
4 bit positions right CF = original
bit 3, Bit 4 - original CF.

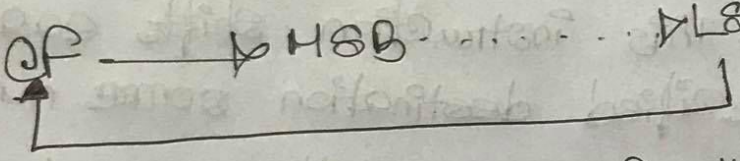
SAL - SAL Destination, Count
SHL - SHL Destination, Count

SAL and SHL are two mnemonics for the same instruction. 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. In the case of multi-bit shift. CF will contain the bit most recently shifted out from the MSB. Bits shifted into CF previously will be lost.

CF ← MSB ← . . . LSB ←

RCR - RCR Destination, Count

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



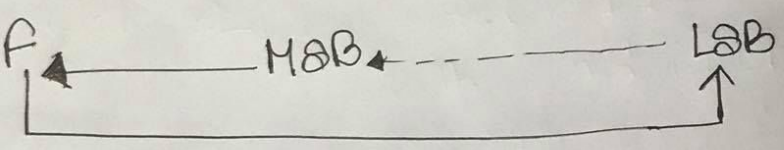
For multi-bit rotate, CF will contain the bit most recently rotated out of the LSB.

RCR affects 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 the multi-bit rotate.

e.g. RCR BX, 1 word in BX right 1bit, CF to MSB, LSB to CF

RCL - RCL Destination, Count

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

RCL affects only Cf and 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.

```
>RCL DX, 1        Word in DX 1 bit left, MSB to Cf, Cf to LSB
> MOV CL,4        Load the number of bit positions to
RCL SUM[BX],CL    rotate into 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
```

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.

⇒ 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.

⇒ XOR word PTR[BX], 00FFH
Exclusive-OR immediate number 00FFH with word at offset [BX] in the Data segment.

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 and destination cannot both be memory locations. CF and OF are both 0 after OR. PF, SF, and ZF are updated by the 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

> Let AL = 0011 0101 (ASCII 5), and BL = 0011 1001 (ASCII 9)

ADD AL, BL

AL = 0110 1110 (6EH, which is incorrect BCD)

AAA

AL = 0000 0100 (unpacked BCD 4)

CF = 1 indicates answer is 14 decimal.

LOGICAL INSTRUCTIONS

AND – AND Destination, Source

The 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.

> AND CX, [SI] AND word in DS at offset [SI] with word in 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 byte, leaves lower byte unchanged

DAA (DECIMAL ADJUST AFTER BCD)

This instruction is used to make sure that 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 06H to AL

> Let AL = 59 BCD, and 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, and 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 and ZF; but OF is undefined.

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

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

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. This means that if an 8-bit destination containing 00H

- 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.
  Decrement a byte if COUNT is declared with a DB.
  Decrement a word if COUNT is declared with a DW.

> DIV SCALE[BX] AX/[BX] if SCALE[BX] is of type byte; or (DX and 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 and 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.

> 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] and [BX+1] in the data segment.

> INC TEMP Increment byte or word named TEMP in the data segment.

DIV - DIV Source

This instruction is used to devide 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 an AH will contain the 8 bit remainder. when double word is divided by a word. the most significant word of the double word must be in AX. After the division AX will contain the 16 bit quotient and DX will contain the 16 bit remainder. If an attempt is made to divide by 0 or if the quotient is too large to fit in the destination, the 8086 will generate a type 0 interupt. All flags are undefined after a DIV instruction.

> DIV BL Divided word in AX by byte in BL; Quotient in AL, remainder in AH

> DIV CX Divided down word in DX and AX by word in CX; Quotient in AX and remainder in DX.

and 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 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 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 to by [BX]
- MUL FACTOR [BX] — Multiply AL with byte at effective address. FACTOR [BX]. If it is declared as type by with DB.

```
MOV AX, MCAND_16    Load 16-bit ⊗ multiplicand in 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 result in DX and 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 form effective address TABLE [BX] and status of CF from CX

➢ SBB TABLE [BX], CX Subtract CX and status 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 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 is put in AX. When a word is multiplied by the content of AX, the result is put in DX

- ADD AL, 74H — Add immediate number 74H to content of AL. Result in AL
- ADC CL, BL — Add content of BL plus carry status 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 status to content of AL
- ADD AL, PRICES [BX] — Add content of memory at effective address PRICE [BX] to AL

SUB – SUB Destination, Source

SBB – SBB Destination, Source

These instructions subtract the number in some source from the number in some destination and put the result in the destination. The SBB instruction also subtracts the content of carry flag from the destination. 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 and fill the upper byte of the word with 0s. flags affected: 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 — Subtract immediate number 3427H from AX.

8086 INSTRUCTION SET

MOV - MOV Destination, Source

The move instruction copies a word or byte from a specified source 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 and destination cannot both be memory locations. They must be of the same type. MOV instruction does not affect any flag.

- MOV CX, 037AH — Put immediate number 037AH to CX
- MOV BL, [437AH] — 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
- MOV RESULT[BP], AX — Copy AX to two memory locations; AL to the 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.

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 indicated 16 bit-register. LEA does not 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]

ARITHMETIC INSTRUCTION

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 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 and the destination in an instruction cannot both be memory locations. The source and the destination must be of the same type. If you want to add a byte to a word, you must copy the byte to a word location and fill the upper byte of the word with 0's before adding. Flags affected: AF, CF, OF, SF, ZF