MOV - MOV Destination, Source

The MOV instruction copies a word or byte of data from a specified source to a specific 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 both be of the same type (bytes or words). MOV 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]
MOV DS, BX
MOV RESULT [BP], AX   copy AX to two memory
MOV ES: RESULT[BP], AX   locations. EA+ES
```

LEA - LEA Register, source

The 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.

~~XCHG AX, DX~~ Exchange word in AX with word in DX

~~XCHG BL, BH~~ Exchange byte in BL with byte in BH

~~XCHG AL, PRICE[BX]~~ EA = PRICE [BX] in DS

ADD - ADD Destination, Source

ADC - ADC Destination, Source

This 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. 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.

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]

SUB - SUB Destination, source

SBB - SBB Destination, source

This 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. 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 and the destination must both be of the same type. 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 0's. Flag Affected : AF, CF, OF, PF, SF, ZF.

SUB CX, BX CX-BX ; Result in CX

SBB CH, AX Subtract content of AL and content of CF from content of CH

SUB AX, 3427H Subtract immediate number 3427H

SBB BX[3427H]

SUB PRICES [BX], 04H Subtract 04 from byte at effective address PRICES [BX]

SBB CX, TABLE [BX] Subtract word from 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 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 and 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 is multiplied by the content of AX, the result is put in DX and AX registers. AF, PF, SF and 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 and fill the upper byte of the word with all 0's.

```
MUL BH
MUL CX
MUL BYTE PTR [BX]
MUL FACTOR [BX]
MOV AX, MCAND_16
MOV CL, MPLIER_8
MOV CH, 00H
MUL CX
```

DIV – DIV Source

This instruction is used to devide an unsigned word by on 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 and AH will contain the 8-bit remainder. When a double word is divided by a word, the most significant word of the ~~division~~ double word must be in DX, and the least significant word of a double word must be in AX. If an attempt is made to devide by 0 or if the quotient is too large to fit in the destination (greater than FFH/FFFFH), the 8086 will generate a type 0 interrupt. All flags are undefined after a DIV instruction.

If you want to divide a byte by a byte, you must first put the dividend byte in AL and fill AH with all 0's.

```
DIV BL
DIV CX
DIV SCALE [BX]
```

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 FFFFH is incremented, the result will be all 0's with no carry.

```
INC BL      'add 1 to contains BL register
INC CX
INC BYTE PTR [BX]
```

INC WORD PTR [BX]

INC TEMP

INC PRICES [BX]

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 the lower nibble in AL. If the carry flag was set by the addition or correction, then the DAA instruction will add 60H to AL.

Let AL = 59 BCD, and BL = 35 BCD

```
ADD AL, BL
DAA
```

Let AL = 88 BCD and BL = 90 BCD

ADD AL, BL

DAA

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. 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 = 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

CF = 1 indicates answer is 14 decimal

AAA instruction works only on the AL register

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 and the source cannot both be memory location. CF and OF are

both 0 after AND. PF has meaning only for an 8-bit operand

```
AND CX, [SI]
AND BH, CL
AND BX, 00FFH
```

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 can't both be memory locations. CF and OF are both 0 after OR. PF has meaning only for an 8-bit operand

```
OR AH, CL
OR BP, SI
OR SI, BP
```

OR BL

OR SX, TABLE [SI]

XOR -

This instruction Exclusive-ORs each bit in a source byte or word with the same numbered bit in a destination can be a register or a memory location. The source and destination cannot both be memory locations. CF and OF are both 0 after XOR.

XOR CL, BH

XOR BP, DI

XOR WORD PTR [BX], 00FFH

CMP -

This instruction compares a byte/word in the specified source with a byte/word in the specified source with a byte/word in destination. The source can be an

immediate number, a register or a memory location. the comparison is actually done by subtracting the source. 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
CX > BX	0	0	0
CX < BX	1	0	1

```
CMP  AL, 01H
CMP  BH, CL
CMP  CX, TEMP
CMP  PRICES [BX], 90H
```

TEST

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

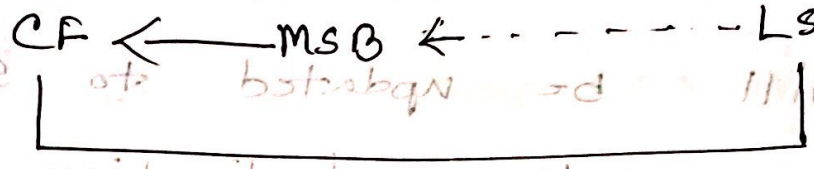
is changed. The destination can be a register or a memory location. Effect of and both 0's after TEST. PF, SF and ZF will be updated to show the result of the destination. AF is be undefined.

```
TEST  AL, BH
TEST  CX, 0001H
TEST  BP, [BX][DI]
```

RCL

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 is rotated into the carry flag and the bit in the

carry flag is rotated around into LSB of the operand.



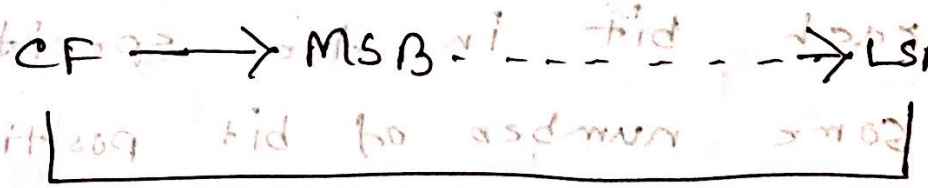
For multi-bit rotates, CF will contain the bit most recently rotated out of the MSB. If you want to rotate operand by one bit position, you can specify this by putting a 1 in the count position of the instruction. To rotate by a number other than one bit position.

RCL/D 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 multi-bit rotate.

```
RCL DX,1
MOV CL,4
RCL SUM [BX],CL
```

RCR

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 and the bit in the carry flag is rotated around into MSB of the operand.



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

To rotate more than one bit position, load the desired number into the CL register and put "CL" in the count position of the instruction. RCR affects only CF and OF

RCR BX, 1
MOV CL, 9
RCR BYTE PTR [BX], 9

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 of the MSB operation, a 0 is put in the LSB position. The MSB will be shifted into CF. Bits shifted into CF previously will be lost.

The destination operand can be a byte or word. It can be a register or in a memory location. The flags are affected as follows: CF contains the bit most recently shifted out from MSB. OF will be 1 if CF and the current MSB are not the same; for multiple-bit shifts, SF and ZF will be updated to reflect the the condition of the destination. PF will have meaning only for an operand in AL.

```
SAL BX, 1
MOV CL, 02h
SAL BP, CL
SAL BYTE PTR [BX], 1
```

SAR

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, the sign bit is copied into the MSB. The LSB will be shifted into CF. Bits shifted into CF previously will be lost.

MSB → MSB → ... → LSB → CF

The flags are affected as follow:

CF - contains the bits 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. PF will have meaning only for an 8-bit destination. AF will be undefined after SAR.

```
SAR DX, 1
MOV CL, 02H
SAR WORD PTR [BP], CL
```

SHR

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 its place. The bit shifted out of the LSB position goes to CF. Bits shifted into CF previously will be lost.

The flags are affected by SHR as follows: 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. SF and ZF will be updated to show the condition of the destination. PF will have meaning only for an 8-bit destination.

```
SHR BP, 1
MOV CL, 03H
SHR BYTE PTR [BX]
```

JMP

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

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, 9371H
JBE NEXT
```

```
CMP AX, 9371H
JNA NEXT
```

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
JG NEXT

CMP BL, 39H
JNLE NEXT
```

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
JL AGAIN
```

```
CMP BL, 39H
JNGE AGAIN
```

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

This instruction is usually used after a Compare instruction. This 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
JLE NEXT
CMP BL, 39H
JNG NEXT
```

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
JE DONE
IN AL, 30H
SUB AL, 30H
JZ START
```

PUSH

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. This instruction does not affect any flag.

```
PUSH BX
PUSH DS
PUSH BL
PUSH TABLE [BX]
```

POP

The POP instruction copies a word from the stack location pointed to by the stack pointer to a destination specified in the

instruction. The destination can be a general-purpose register. After the word is copied to the specified destination, the stack pointer is automatically incremented by 2 to point to the next word on the stack. The POP instruction does not affect any flag.

```
POP   DX
POP   DS
POP   TABLE[DX]
```

IN - IN Accumulator, Port

IN AL, 0C8H

IN AX, 39H

OUT - OUT Port, Accumulator

The OUT instruction copies a byte from AL or a word from AX to the specified port. For the fixed port form, the 8-bit port address is specified directly in the instruction.

OUT 3BH, AL

OUT 2CH, AX

ENDS (END SEGMENT)

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

CODE @ SEGMENT

CODE ENDS

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 only one END.

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.

```
WORDS DW 1234H, 3456H
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 PROC directive, the term near or the term far is used to specify the type of the procedure. The statement DIVIDE PROC 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
SQUARE_ROOT ENDP
```

INCLUDE (INCLUDE SOURCE CODE FROM FILE)

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