MOV - MOV Destination, source.

The MOV instruction copies a word or byte of data from a specified source to a specified destination. The destination can be a register or 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 instruction does not affect any flag.

```
MOV CX, 037AH
MOV BL, [437AH].
MOV AX, BX
MOV DL, [BX].
MOV DS, BX
MOV RESULT [BP], AX
MOV ES: RESULTS [BP], AX.
```

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.

=> LEA BP, SS:STACK_TOP

=> LEA CX, [BX][DI].

SUB - SUB Destnation, source.
SBB - SBB Destnation, source.

These instructions subtract the number in some source from the number in some destnation and put the result in the destnation. The SBB instruction also subtracts the content of carry flag from the destnation. The source may be an immediate number, a register or memory location. The destnation can also be a register or a memory location. However, the source and the destnation can not both be memory location. The source and the destnation 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. Flags affected: AF, CF, OF, PF, SF, ZF.

```
SUB CX, BX .
SBB CH, AL .

SUB AX, 3427H .
SBB BX, [3427H] .

SUB PRICES [BX], 04H .
SBB CX, TABLE [BX]
SBB TABLE [BX], CX
```

4 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 AX, the result is put 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 be 0's. AF, PF, SF and ZF are undefined after a MUL instruction

```
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 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 quotient, and 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, and the least 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 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. Likewise, if you want to divide a word by another word, then put the dividend word in AX and fill DX 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 FFH or a 16-bit destination containing FFFFH is incremented, the result will be all 0's with no carry.

```
INC BL
INC CX
INC BYTE PTR [BX]
INC WORD PTR [BX].
INC TEMP
INC PRICES [BX].
```

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 or a 16-bit destination containing 0000H is decremented, the result will be FFH or FFFFH with no carry.

```
DEC CL
DEC BP
DEC BYTE PTR [BX]
DEC WORD PTR [BP]
DEC COUNT
```

DAA (DD)

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 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 result in the upper nibble of AL in now greater than 9 or if the carry flag was set by the addition or correction, then DAA instruction will add 60H to AL.

```
AL = 59 BCD, BL = 35 BCD.
ADD AL, BL
DAA
AL = 88 BCD, BL = 49 BCD
ADD AL, BL
DAA
```

AAA :

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.

AL = 0011 0101 BL = 0011 1001

ADD AL, BL AL = 0110 1110

AAA AL = 0000 0100

CF = 1 indicates answer is 14 decimal.

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.

```
OR AH, CL
OR BP, SI
OR BL, 80H
OR CX, TABLE [SI]
```

XOR - XOR Destination, Source

This instruction Exclusive-ORs each bit in a source 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 and destination can not both be memory locations. CF and OF are both 0 after XOR, PF, SF, ZF are updated. PF has meaning only for an 8-bit operand. AF is undefined.

```
XOR CL, BH

XOR BP, DI

XOR WORD PTR [BX], 00FFH
```

#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 and the destination can not both be memory locations. The comparison is actually done by subtracting the source byte or word from the destination byte or word. The source and the destination are not changed, but the flags are set to indicate the results of the comparison. 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:

CX = BX
CX > BX.
CX < BX.

```
CMP AL, 01H.
CMP BH, CL.
CMP CX, TEMP
CMP PRICES[BX], 49H.
```

TEST - TEST Destination, source

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 test instruction is often used to set flags before a conditional jump instruction.

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

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

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
JBE NEXT
CMP AX, 4371H
JNA NEXT
```

JG / JNLE:

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 same as the overflow flag.

=>CMP BL, 39H
JA NEXT

=>CMP BL, 39H
JNLE NEXT.

JL / JGNE :

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

JE / JZ ~~&~~

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
```

JNE / JNZ :

This instruction is 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
CMP AL, 72
JNE NEXT

ADD AX, 0002H
DEC BX
JNZ NEXT
```

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. The source of the word can be general purpose register, segment register, or memory. The stack segment register and 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

destroyed by a procedure
This instruction does not affect
any flag.

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

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 data in the 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.

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

ENDS (END SEGMENT)

This directive is used with the name of a segment to indicate the end of the 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 would 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 initalized with the value 437A A when the program is loaded into memory to be run.

```
WORDS DW 1234H, 3456H.
STORAGE DW 100 DUP(0)
STORAGE DW 100 DUP(?)
```

#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).

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