

Complete 8086 instruction set

Quick reference:

AAA	CMPSB	JAE	JNBE	JPO	MOV	RCR	SCASB
AAD	CMPSW	JB	JNC	JS	MOVS	REP	SCASW
AAM	CWD	IBE	JNE	JZ	MUL	REPE	SHL
AAS	DAA	JC	JNG	LAHF	NEG	REPNE	SHR
ADC	DAS	JCXZ	JNGE	LDS	NOP	REPNZ	STC
ADD	DEC	JE	JNL	LEA	NOT	REPZ	STD
AND	DIV	JG	JNLE	LES	OR	RET	STI
CALL	IMUL	JGE	JNO	LODSB	OUT	RETF	STOSB
CBW	IDIV	JL	JNP	LODSW	POP	ROL	STOSW
CLC	IN	JLE	JNS	LOOP	POPA	ROR	SUB
CLD	INC	JMP	JNZ	LOOPE	POPF	SAHF	TEST
CLI	INT	JNA	JO	LOOPNE	PUSH	SAL	XCHG
CMC	INTO	JNAE	JP	LOOPNZ	PUSHA	SAR	XLATB
CMP	IRET	JNB	JPE	LOOPZ	PUSHF	SBB	XOR
		JA			RCL		

Operand types:

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

SREG: DS, ES, SS, and only as second operand: CS.

memory: [BX], [BX+SI+7], variable, etc... (see [Memory Access](#)).

immediate: 5, -24, 3Fh, 10001101b, etc...

Notes:

- When two operands are required for an instruction they are separated by comma. For example:

REG, memory

- When there are two operands, both operands must have the same size (except shift and rotate instructions). For example:

AL, DL

DX, AX

m1 DB ?

AL, m1

m2 DW ?

AX, m2

- Some instructions allow several operand combinations. For example:

memory, immediate
REG, immediate

memory, REG
REG, SREG

- Some examples contain macros, so it is advisable to use **Shift + F8** hot key to *Step Over* (to make macro code execute at maximum speed set **step delay** to zero), otherwise emulator will step through each instruction of a macro. Here is an example that uses PRINTN macro:

```
include 'emu8086.inc'
ORG 100h
MOV AL, 1
MOV BL, 2
PRINTN 'Hello World!' ; macro.
MOV CL, 3
PRINTN 'Welcome!' ; macro.
RET
```

These marks are used to show the state of the flags:

- 1** - instruction sets this flag to **1**.
0 - instruction sets this flag to **0**.
r - flag value depends on result of the instruction.
? - flag value is undefined (maybe **1** or **0**).
-

Some instructions generate exactly the same machine code, so disassembler may have a problem decoding to your original code. This is especially important for Conditional Jump instructions (see "[Program Flow Control](#)" in Tutorials for more information).

Instructions in alphabetical order:

Instruction	Operands	Description
AAA	No operands	ASCII Adjust after Addition. Corrects result in AH and AL after addition when working with BCD values.

It works according to the following Algorithm:

if low nibble of AL > 9 or AF = 1 then:

- AL = AL + 6
- AH = AH + 1
- AF = 1
- CF = 1

else

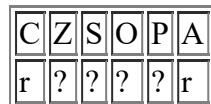
- AF = 0
- CF = 0

in both cases:

clear the high nibble of AL.

Example:

```
MOV AX, 15 ; AH = 00, AL = 0Fh
AAA        ; AH = 01, AL = 05
RET
```



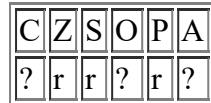
		ASCII Adjust before Division. Prepares two BCD values for division.												
		Algorithm:												
AAD	No operands	<ul style="list-style-type: none"> • AL = (AH * 10) + AL • AH = 0 												
		Example:												
		<pre>MOV AX, 0105h ; AH = 01, AL = 05 AAD ; AH = 00, AL = 0Fh (15) RET</pre>												
		<table border="1"> <tr> <td>C</td> <td>Z</td> <td>S</td> <td>O</td> <td>P</td> <td>A</td> </tr> <tr> <td>?</td> <td>r</td> <td>r</td> <td>?</td> <td>r</td> <td>?</td> </tr> </table>	C	Z	S	O	P	A	?	r	r	?	r	?
C	Z	S	O	P	A									
?	r	r	?	r	?									

		ASCII Adjust after Multiplication. Corrects the result of multiplication of two BCD values.
		Algorithm:
AAM	No operands	

- AH = AL / 10
- AL = remainder

Example:

```
MOV AL, 15 ; AL = 0Fh
AAM      ; AH = 01, AL = 05
RET
```



		<p>ASCII Adjust after Subtraction. Corrects result in AH and AL after subtraction when working with BCD values.</p> <p>Algorithm:</p> <p>if low nibble of AL > 9 or AF = 1 then:</p> <ul style="list-style-type: none"> • AL = AL - 6 • AH = AH - 1 • AF = 1 • CF = 1 <p>else</p> <ul style="list-style-type: none"> • AF = 0 • CF = 0 <p>in both cases: clear the high nibble of AL.</p> <p>Example:</p> <pre>MOV AX, 02FFh ; AH = 02, AL = 0FFh AAS ; AH = 01, AL = 09 RET</pre> <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td>r</td><td>?</td><td>?</td><td>?</td><td>?</td><td>r</td></tr> </table>	C	Z	S	O	P	A	r	?	?	?	?	r
C	Z	S	O	P	A									
r	?	?	?	?	r									
AAS	No operands													

ADC	REG, memory memory, REG REG, REG memory, immediate REG, immediate	<p>Add with Carry.</p> <p>Algorithm:</p> <p>operand1 = operand1 + operand2 + CF</p> <p>Example:</p>
-----	---	---

```
STC      ; set CF = 1
MOV AL, 5 ; AL = 5
ADC AL, 1 ; AL = 7
RET
```



ADD	REG, memory memory, REG REG, REG memory, immediate REG, immediate	<p>Add.</p> <p>Algorithm:</p> $\text{operand1} = \text{operand1} + \text{operand2}$ <p>Example:</p> <pre>MOV AL, 5 ; AL = 5 ADD AL, -3 ; AL = 2 RET</pre> <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr> </table>	C	Z	S	O	P	A	r	r	r	r	r	r
C	Z	S	O	P	A									
r	r	r	r	r	r									
AND	REG, memory memory, REG REG, REG memory, immediate REG, immediate	<p>Logical AND between all bits of two operands. Result is stored in operand1.</p> <p>These rules apply:</p> <p>1 AND 1 = 1 1 AND 0 = 0 0 AND 1 = 0 0 AND 0 = 0</p> <p>Example:</p> <pre>MOV AL, 'a' ; AL = 01100001b AND AL, 11011111b ; AL = 01000001b ('A') RET</pre> <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td></tr> <tr><td>0</td><td>r</td><td>r</td><td>0</td><td>r</td></tr> </table>	C	Z	S	O	P	0	r	r	0	r		
C	Z	S	O	P										
0	r	r	0	r										
CALL	procedure name label 4-byte address	Transfers control to procedure. Return address (IP) is pushed to stack. <i>4-byte address</i> may be entered in this form: 1234h:5678h, first value is a segment second value is an offset. If it's a far call,												

then code segment is pushed to stack as well.

Example:

ORG 100h ; for COM file.

CALL p1

ADD AX, 1

RET ; return to OS.

p1 PROC ; procedure declaration.

MOV AX, 1234h

RET ; return to caller.

p1 ENDP



		Convert byte into word.												
		<p>Algorithm:</p> <p>if high bit of AL = 1 then:</p> <ul style="list-style-type: none"> • AH = 255 (0FFh) <p>else</p> <ul style="list-style-type: none"> • AH = 0 												
CBW	No operands	<p>Example:</p> <pre>MOV AX, 0 ; AH = 0, AL = 0 MOV AL, -5 ; AX = 000FBh (251) CBW ; AX = 0FFF8h (-5) RET</pre> <table border="1"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td> </tr> <tr> <td colspan="6">unchanged</td> </tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
CLC	No operands	<p>Clear Carry flag.</p> <p>Algorithm:</p> <p>CF = 0</p>												


[Back](#)

CLD	No operands	<p>Clear Direction flag. SI and DI will be incremented by chain instructions: CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW.</p> <p>Algorithm:</p> <p>$DF = 0$</p> <p>A small diagram of the flag register. It consists of two vertical columns of three squares each. The top row has 'D' in the left column and '0' in the right column. The middle row has an empty square in the left column and 'I' in the right column. The bottom row has an empty square in the left column and '0' in the right column.</p> Back
CLI	No operands	<p>Clear Interrupt enable flag. This disables hardware interrupts.</p> <p>Algorithm:</p> <p>$IF = 0$</p> <p>A small diagram of the flag register. It consists of two vertical columns of three squares each. The top row has an empty square in the left column and 'I' in the right column. The middle row has an empty square in the left column and '0' in the right column.</p> Back
CMC	No operands	<p>Complement Carry flag. Inverts value of CF.</p> <p>Algorithm:</p> <p>if $CF = 1$ then $CF = 0$ if $CF = 0$ then $CF = 1$</p> <p>A small diagram of the flag register. It consists of two vertical columns of three squares each. The top row has an empty square in the left column and 'C' in the right column. The middle row has an empty square in the left column and 'r' in the right column.</p> Back
CMP	REG, memory memory, REG REG, REG memory, immediate REG, immediate	<p>Compare.</p> <p>Algorithm:</p> <p>$operand1 - operand2$</p> <p>result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.</p>

Example:

```
MOV AL, 5
MOV BL, 5
CMP AL, BL ; AL = 5, ZF = 1 (so equal!)
RET
```



		Compare bytes: ES:[DI] from DS:[SI].
		Algorithm:
CMPSB	No operands	<ul style="list-style-type: none"> • DS:[SI] - ES:[DI] • set flags according to result: OF, SF, ZF, AF, PF, CF • if DF = 0 then <ul style="list-style-type: none"> ◦ SI = SI + 1 ◦ DI = DI + 1 else <ul style="list-style-type: none"> ◦ SI = SI - 1 ◦ DI = DI - 1 <p>Example: open cmpsb.asm from c:\emu8086\examples</p>



		Compare words: ES:[DI] from DS:[SI].
		Algorithm:
CMPSW	No operands	<ul style="list-style-type: none"> • DS:[SI] - ES:[DI] • set flags according to result: OF, SF, ZF, AF, PF, CF • if DF = 0 then <ul style="list-style-type: none"> ◦ SI = SI + 2 ◦ DI = DI + 2 else <ul style="list-style-type: none"> ◦ SI = SI - 2 ◦ DI = DI - 2 <p>example: open cmpsw.asm from c:\emu8086\examples</p>





		<p>Convert Word to Double word.</p> <p>Algorithm:</p> <p>if high bit of AX = 1 then:</p> <ul style="list-style-type: none"> • DX = 65535 (0FFFFh) <p>else</p> <ul style="list-style-type: none"> • DX = 0 <p>CWD No operands</p> <p>Example:</p> <pre>MOV DX, 0 ; DX = 0 MOV AX, 0 ; AX = 0 MOV AX, -5 ; DX AX = 00000h:0FFFh CWD ; DX AX = 0FFFFh:0FFFh RET</pre> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr> <td>unchanged</td><td></td><td></td><td></td><td></td><td></td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
DAA	No operands	<p>Decimal adjust After Addition. Corrects the result of addition of two packed BCD values.</p> <p>Algorithm:</p> <p>if low nibble of AL > 9 or AF = 1 then:</p> <ul style="list-style-type: none"> • AL = AL + 6 • AF = 1 <p>if AL > 9Fh or CF = 1 then:</p> <ul style="list-style-type: none"> • AL = AL + 60h • CF = 1 <p>Example:</p> <pre>MOV AL, 0Fh ; AL = 0Fh (15) DAA ; AL = 15h RET</pre> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr> <td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr> </table>	C	Z	S	O	P	A	r	r	r	r	r	r
C	Z	S	O	P	A									
r	r	r	r	r	r									



		<p>Decimal adjust After Subtraction. Corrects the result of subtraction of two packed BCD values.</p> <p>Algorithm:</p> <p>if low nibble of AL > 9 or AF = 1 then:</p> <ul style="list-style-type: none"> • AL = AL - 6 • AF = 1 <p>if AL > 9Fh or CF = 1 then:</p> <ul style="list-style-type: none"> • AL = AL - 60h • CF = 1 <p>Example:</p> <pre>MOV AL, 0FFh ; AL = 0FFh (-1) DAS ; AL = 99h, CF = 1 RET</pre> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr> <td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr> </table>	C	Z	S	O	P	A	r	r	r	r	r	r
C	Z	S	O	P	A									
r	r	r	r	r	r									



		<p>Decrement.</p> <p>Algorithm:</p> <p>operand = operand - 1</p> <p>Example:</p> <pre>MOV AL, 255 ; AL = 0FFh (255 or -1) DEC AL ; AL = 0FEh (254 or -2) RET</pre> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr> <td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr> </table> <p>CF - unchanged!</p>	Z	S	O	P	A	r	r	r	r	r
Z	S	O	P	A								
r	r	r	r	r								



DIV	REG memory	<p>Unsigned divide.</p> <p>Algorithm:</p>
-----	---------------	--

when operand is a byte:

AL = AX / operand

AH = remainder (modulus)

when operand is a word:

AX = (DX AX) / operand

DX = remainder (modulus)

Example:

```
MOV AX, 203 ; AX = 00CBh
MOV BL, 4
DIV BL      ; AL = 50 (32h), AH = 3
RET
```



HLT	No operands	Halt the System.										
		Example: <pre>MOV AX, 5 HLT</pre> <table border="1"> <tr> <td>C</td> <td>Z</td> <td>S</td> <td>O</td> <td>P</td> <td>A</td> </tr> <tr> <td>unchanged</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </table>	C	Z	S	O	P	A	unchanged			
C	Z	S	O	P	A							
unchanged												



IDIV	REG memory	<p>Signed divide.</p> <p>Algorithm:</p> <p>when operand is a byte:</p> <p>AL = AX / operand</p> <p>AH = remainder (modulus)</p> <p>when operand is a word:</p> <p>AX = (DX AX) / operand</p> <p>DX = remainder (modulus)</p> <p>Example:</p> <pre>MOV AX, -203 ; AX = 0FF35h MOV BL, 4 IDIV BL ; AL = -50 (0CEh), AH = -3 (0FDh) RET</pre> <table border="1"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr> <td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td></tr> </table>	C	Z	S	O	P	A	?	?	?	?	?	?
C	Z	S	O	P	A									
?	?	?	?	?	?									



		<p>Signed multiply.</p> <p>Algorithm:</p> <p>when operand is a byte: $AX = AL * \text{operand}$.</p> <p>when operand is a word: $(DX\ AX) = AX * \text{operand}$.</p> <p>Example:</p> <pre>MOV AL, -2 MOV BL, -4 IMUL BL ; AX = 8 RET</pre> <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td>r</td><td>?</td><td>?</td><td>r</td><td>?</td><td>?</td></tr> </table> <p>CF=OF=0 when result fits into operand of IMUL.</p>	C	Z	S	O	P	A	r	?	?	r	?	?
C	Z	S	O	P	A									
r	?	?	r	?	?									
IN	AL, im.byte AL, DX AX, im.byte AX, DX	<p>Input from port into AL or AX. Second operand is a port number. If required to access port number over 255 - DX register should be used.</p> <p>Example:</p> <pre>IN AX, 4 ; get status of traffic lights. IN AL, 7 ; get status of stepper-motor.</pre> <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td colspan="6">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
INC	REG memory	<p>Increment.</p> <p>Algorithm:</p> <p>$\text{operand} = \text{operand} + 1$</p> <p>Example:</p> <pre>MOV AL, 4 INC AL ; AL = 5 RET</pre> <table border="1"> <tr><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr> </table> <p>CF - unchanged!</p>	Z	S	O	P	A	r	r	r	r	r		
Z	S	O	P	A										
r	r	r	r	r										

INT

immediate byte

Interrupt numbered by immediate byte (0..255).**Algorithm:**

Push to stack:

- flags register
- CS
- IP
- IF = 0
- Transfer control to interrupt procedure

Example:

```
MOV AH, 0Eh ; teletype.
MOV AL, 'A'
INT 10h    ; BIOS interrupt.
RET
```



INTO

No operands

Interrupt 4 if Overflow flag is 1.

Algorithm:

if OF = 1 then INT 4

Example:

```
; -5 - 127 = -132 (not in -128..127)
; the result of SUB is wrong (124),
; so OF = 1 is set:
MOV AL, -5
SUB AL, 127 ; AL = 7Ch (124)
INTO      ; process error.
RET
```



IRET

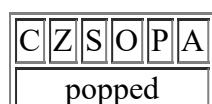
No operands

Interrupt Return.

Algorithm:

Pop from stack:

- IP
- CS
- flags register





		<p>Short Jump if first operand is Above second operand (as set by CMP instruction). Unsigned.</p> <p>Algorithm:</p> <pre>if (CF = 0) and (ZF = 0) then jump</pre> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 250 CMP AL, 5 JA label1 PRINT 'AL is not above 5' JMP exit label1: PRINT 'AL is above 5' exit: RET</pre> <table border="1"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr> <td colspan="6">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
JA	label	<p>Short Jump if first operand is Above or Equal to second operand (as set by CMP instruction). Unsigned.</p> <p>Algorithm:</p> <pre>if CF = 0 then jump</pre> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 5 CMP AL, 5 JAE label1 PRINT 'AL is not above or equal to 5' JMP exit label1: PRINT 'AL is above or equal to 5' exit: RET</pre> <table border="1"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr> <td colspan="6">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
JAE	label	<p>Short Jump if first operand is Above or Equal to second operand (as set by CMP instruction). Unsigned.</p> <p>Algorithm:</p> <pre>if CF = 0 then jump</pre> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 5 CMP AL, 5 JAE label1 PRINT 'AL is not above or equal to 5' JMP exit label1: PRINT 'AL is above or equal to 5' exit: RET</pre> <table border="1"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr> <td colspan="6">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														



		<p>Short Jump if first operand is Below second operand (as set by CMP instruction). Unsigned.</p> <p>Algorithm:</p> <p style="padding-left: 40px;">if CF = 1 then jump</p> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 1 CMP AL, 5 JB label1 PRINT 'AL is not below 5' JMP exit label1: PRINT 'AL is below 5' exit: RET</pre> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr> <td colspan="6" style="text-align: center;">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
JB	label	<p>Short Jump if first operand is Below or Equal to second operand (as set by CMP instruction). Unsigned.</p> <p>Algorithm:</p> <p style="padding-left: 40px;">if CF = 1 or ZF = 1 then jump</p> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 5 CMP AL, 5 JBE label1 PRINT 'AL is not below or equal to 5' JMP exit label1: PRINT 'AL is below or equal to 5' exit: RET</pre> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr> <td colspan="6" style="text-align: center;">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
JBE	label	<p>Short Jump if first operand is Below or Equal to second operand (as set by CMP instruction). Unsigned.</p> <p>Algorithm:</p> <p style="padding-left: 40px;">if CF = 1 or ZF = 1 then jump</p> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 5 CMP AL, 5 JBE label1 PRINT 'AL is not below or equal to 5' JMP exit label1: PRINT 'AL is below or equal to 5' exit: RET</pre> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr> <td colspan="6" style="text-align: center;">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														



JC	label	<p>Short Jump if Carry flag is set to 1.</p> <p>Algorithm:</p> <pre>if CF = 1 then jump</pre> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 255 ADD AL, 1 JC label1 PRINT 'no carry.' JMP exit label1: PRINT 'has carry.' exit: RET</pre> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">C</td> <td style="padding: 2px;">Z</td> <td style="padding: 2px;">S</td> <td style="padding: 2px;">O</td> <td style="padding: 2px;">P</td> <td style="padding: 2px;">A</td> </tr> <tr> <td colspan="6" style="text-align: center; padding: 2px;">unchanged</td> </tr> </table> </div>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
JCXZ	label	<p>Short Jump if CX register is 0.</p> <p>Algorithm:</p> <pre>if CX = 0 then jump</pre> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV CX, 0 JCXZ label1 PRINT 'CX is not zero.' JMP exit label1: PRINT 'CX is zero.' exit: RET</pre> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">C</td> <td style="padding: 2px;">Z</td> <td style="padding: 2px;">S</td> <td style="padding: 2px;">O</td> <td style="padding: 2px;">P</td> <td style="padding: 2px;">A</td> </tr> <tr> <td colspan="6" style="text-align: center; padding: 2px;">unchanged</td> </tr> </table> </div>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
JE	label	<p>Short Jump if first operand is Equal to second operand (as set by CMP instruction).</p>												

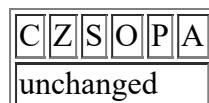
Signed/Unsigned.

Algorithm:

if ZF = 1 then jump

Example:

```
include 'emu8086.inc'
ORG 100h
MOV AL, 5
CMP AL, 5
JE label1
PRINT 'AL is not equal to 5.'
JMP exit
label1:
PRINT 'AL is equal to 5.'
exit:
RET
```



JG	label	<p>Short Jump if first operand is Greater than second operand (as set by CMP instruction). Signed.</p> <p>Algorithm:</p> <p style="text-align: center;">if (ZF = 0) and (SF = OF) then jump</p> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 5 CMP AL, -5 JG label1 PRINT 'AL is not greater -5.' JMP exit label1: PRINT 'AL is greater -5.' exit: RET</pre> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <table border="1" style="border-collapse: collapse; text-align: center;"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr> <td colspan="6">unchanged</td></tr> </table> </div>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
JGE	label	<p>Short Jump if first operand is Greater or Equal to second operand (as set by CMP instruction). Signed.</p>												

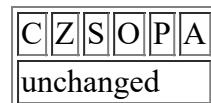


Algorithm:

if SF = OF then jump

Example:

```
include 'emu8086.inc'
ORG 100h
MOV AL, 2
CMP AL, -5
JGE label1
PRINT 'AL < -5'
JMP exit
label1:
PRINT 'AL >= -5'
exit:
RET
```



JL

label

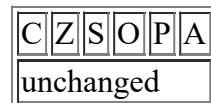
Short Jump if first operand is Less than second operand (as set by CMP instruction). Signed.

Algorithm:

if SF \neq OF then jump

Example:

```
include 'emu8086.inc'
ORG 100h
MOV AL, -2
CMP AL, 5
JL label1
PRINT 'AL >= 5.'
JMP exit
label1:
PRINT 'AL < 5.'
exit:
RET
```



JLE

label

Short Jump if first operand is Less or Equal to second operand (as set by CMP instruction). Signed.

Algorithm:

if SF <> OF or ZF = 1 then jump

Example:

```
include 'emu8086.inc'
ORG 100h
MOV AL, -2
CMP AL, 5
JLE label1
PRINT 'AL > 5.'
JMP exit
label1:
PRINT 'AL <= 5.'
exit:
RET
```



JMP	label 4-byte address	<p>Unconditional Jump. Transfers control to another part of the program. <i>4-byte address</i> may be entered in this form: 1234h:5678h, first value is a segment second value is an offset.</p> <p>Algorithm:</p> <p style="text-align: center;">always jump</p> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 5 JMP label1 ; jump over 2 lines! PRINT 'Not Jumped!' MOV AL, 0 label1: PRINT 'Got Here!' RET</pre> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td colspan="6">unchanged</td></tr> </table> </div>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														

JNA	label	Short Jump if first operand is Not Above second operand (as set by CMP instruction). Unsigned.
-----	-------	--

Algorithm:

if CF = 1 or ZF = 1 then jump

Example:

```
include 'emu8086.inc'

ORG 100h
MOV AL, 2
CMP AL, 5
JNA label1
PRINT 'AL is above 5.'
JMP exit
label1:
PRINT 'AL is not above 5.'
exit:
RET
```



JNAE label Short Jump if first operand is Not Above and Not Equal to second operand (as set by CMP instruction). Unsigned.

Algorithm:

if CF = 1 then jump

Example:

```
include 'emu8086.inc'

ORG 100h
MOV AL, 2
CMP AL, 5
JNAE label1
PRINT 'AL >= 5.'
JMP exit
label1:
PRINT 'AL < 5.'
exit:
RET
```



JNB label Short Jump if first operand is Not Below second operand (as set by CMP instruction). Unsigned.

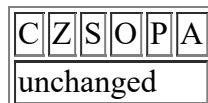
Algorithm:

```
if CF = 0 then jump
```

Example:

```
include 'emu8086.inc'
```

```
ORG 100h
MOV AL, 7
CMP AL, 5
JNB label1
PRINT 'AL < 5.'
JMP exit
label1:
PRINT 'AL >= 5.'
exit:
RET
```



JNBE	label	<p>Short Jump if first operand is Not Below and Not Equal to second operand (as set by CMP instruction). Unsigned.</p> <p>Algorithm:</p> <pre>if (CF = 0) and (ZF = 0) then jump</pre> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 7 CMP AL, 5 JNBE label1 PRINT 'AL <= 5.' JMP exit label1: PRINT 'AL > 5.' exit: RET</pre> <p></p>
JNC	label	Short Jump if Carry flag is set to 0.

Algorithm:

```
if CF = 0 then jump
```

Example:

```
include 'emu8086.inc'
```

```
ORG 100h
MOV AL, 2
ADD AL, 3
JNC label1
PRINT 'has carry.'
JMP exit
label1:
PRINT 'no carry.'
exit:
RET
```



JNE	label	<p>Short Jump if first operand is Not Equal to second operand (as set by CMP instruction). Signed/Unsigned.</p> <p>Algorithm:</p> <pre>if ZF = 0 then jump</pre> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 2 CMP AL, 3 JNE label1 PRINT 'AL = 3.' JMP exit label1: PRINT 'AL <> 3.' exit: RET</pre> <div data-bbox="642 1886 850 1987" data-label="Diagram"> <p>A diagram of the 8086 flag status register. It consists of six boxes arranged horizontally: C (Carry), Z (Zero), S (Sign), O (Overflow), P (Parity), and A (Auxiliary). Below the boxes, the word "unchanged" is written.</p> </div> <div data-bbox="1380 1989 1480 2064" data-label="Image"> </div>
JNG	label	Short Jump if first operand is Not Greater than

second operand (as set by CMP instruction). Signed.

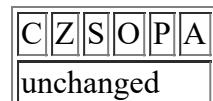
Algorithm:

if (ZF = 1) and (SF <> OF) then jump

Example:

```
include 'emu8086.inc'
```

```
ORG 100h
MOV AL, 2
CMP AL, 3
JNG label1
PRINT 'AL > 3.'
JMP exit
label1:
PRINT 'AL <= 3.'
exit:
RET
```



JNGE

label

Short Jump if first operand is Not Greater and Not Equal to second operand (as set by CMP instruction). Signed.

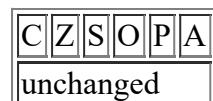
Algorithm:

if SF <> OF then jump

Example:

```
include 'emu8086.inc'
```

```
ORG 100h
MOV AL, 2
CMP AL, 3
JNGE label1
PRINT 'AL >= 3.'
JMP exit
label1:
PRINT 'AL < 3.'
exit:
RET
```



JNL	label	<p>Short Jump if first operand is Not Less than second operand (as set by CMP instruction). Signed.</p> <p>Algorithm:</p> <p style="text-align: center;">if SF = OF then jump</p> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 2 CMP AL, -3 JNL label1 PRINT 'AL < -3.' JMP exit label1: PRINT 'AL >= -3.' exit: RET</pre> <table border="1" data-bbox="647 954 853 1051"> <tr> <td>C</td> <td>Z</td> <td>S</td> <td>O</td> <td>P</td> <td>A</td> </tr> <tr> <td colspan="6" style="text-align: center;">unchanged</td> </tr> </table> 	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
JNLE	label	<p>Short Jump if first operand is Not Less and Not Equal to second operand (as set by CMP instruction). Signed.</p> <p>Algorithm:</p> <p style="text-align: center;">if (SF = OF) and (ZF = 0) then jump</p> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 2 CMP AL, -3 JNLE label1 PRINT 'AL <= -3.' JMP exit label1: PRINT 'AL > -3.' exit: RET</pre> <table border="1" data-bbox="647 2021 853 2117"> <tr> <td>C</td> <td>Z</td> <td>S</td> <td>O</td> <td>P</td> <td>A</td> </tr> <tr> <td colspan="6" style="text-align: center;">unchanged</td> </tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														



		Short Jump if Not Overflow.						
JNO	label	<p>Algorithm:</p> <p style="text-align: center;">if OF = 0 then jump</p> <p>Example:</p> <pre>; -5 - 2 = -7 (inside -128..127) ; the result of SUB is correct, ; so OF = 0: include 'emu8086.inc' ORG 100h MOV AL, -5 SUB AL, 2 ; AL = 0F9h (-7) JNO label1 PRINT 'overflow!' JMP exit label1: PRINT 'no overflow.' exit: RET</pre> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <table border="1" style="margin-bottom: 5px;"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td> </tr> </table> <p>unchanged</p> </div>	C	Z	S	O	P	A
C	Z	S	O	P	A			
JNP	label	<p>Short Jump if No Parity (odd). Only 8 low bits of result are checked. Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.</p> <p>Algorithm:</p> <p style="text-align: center;">if PF = 0 then jump</p> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 00000111b ; AL = 7 OR AL, 0 ; just set flags. JNP label1 PRINT 'parity even.' JMP exit label1: PRINT 'parity odd.' exit: RET</pre>						



[Back](#)

		<p>Short Jump if Not Signed (if positive). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.</p> <p>Algorithm:</p> <pre>if SF = 0 then jump</pre> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 00000111b ; AL = 7 OR AL, 0 ; just set flags. JNS label1 PRINT 'signed.' JMP exit label1: PRINT 'not signed.' exit: RET</pre> <div data-bbox="642 1123 850 1212" style="margin-top: 20px;"> <p>A diagram of the 8086 flag status register. It consists of six boxes arranged in two rows of three. The top row contains 'C', 'Z', 'S'. The bottom row contains 'O', 'P', 'A'. Below the register is the word 'unchanged'.</p> </div> <div data-bbox="1380 1212 1475 1286" data-label="Text"> <p>Back</p> </div>
--	--	---

		<p>Short Jump if Not Zero (not equal). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.</p> <p>Algorithm:</p> <pre>if ZF = 0 then jump</pre> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 00000111b ; AL = 7 OR AL, 0 ; just set flags. JNZ label1 PRINT 'zero.' JMP exit label1: PRINT 'not zero.' exit: RET</pre> <div data-bbox="642 2144 850 2167" style="margin-top: 20px;"> <p>A diagram of the 8086 flag status register. It consists of six boxes arranged in two rows of three. The top row contains 'C', 'Z', 'S'. The bottom row contains 'O', 'P', 'A'. Below the register is the word 'unchanged'.</p> </div>
--	--	--

C	Z	S	O	P	A
unchanged					



		<p>Short Jump if Overflow.</p> <p>Algorithm:</p> <pre>if OF = 1 then jump</pre> <p>Example:</p> <pre>; -5 - 127 = -132 (not in -128..127) ; the result of SUB is wrong (124), ; so OF = 1 is set:</pre> <pre>include 'emu8086.inc' org 100h MOV AL, -5 SUB AL, 127 ; AL = 7Ch (124) JO label1 PRINT 'no overflow.' JMP exit label1: PRINT 'overflow!' exit: RET</pre> <div style="text-align: center;"> <table border="1" style="margin: auto;"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td colspan="6">unchanged</td></tr> </table> </div> <div data-bbox="1380 1313 1479 1388" data-label="Image"> </div>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
JP	label	<p>Short Jump if Parity (even). Only 8 low bits of result are checked. Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.</p> <p>Algorithm:</p> <pre>if PF = 1 then jump</pre> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 00000101b ; AL = 5 OR AL, 0 ; just set flags. JP label1 PRINT 'parity odd.' JMP exit label1:</pre>												

```
PRINT 'parity even.'
exit:
RET
```



Short Jump if Parity Even. Only 8 low bits of result are checked. Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.

Algorithm:

if PF = 1 then jump

Example:

```
include 'emu8086.inc'

ORG 100h
MOV AL, 00000101b ; AL = 5
OR AL, 0           ; just set flags.
JPE label1
PRINT 'parity odd.'
JMP exit
label1:
PRINT 'parity even.'
exit:
RET
```



JPE

label

Short Jump if Parity Odd. Only 8 low bits of result are checked. Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.

Algorithm:

if PF = 0 then jump

Example:

```
include 'emu8086.inc'

ORG 100h
MOV AL, 00000111b ; AL = 7
OR AL, 0           ; just set flags.
JPO label1
PRINT 'parity even.'
```

```
JMP exit
label1:
PRINT 'parity odd.'
exit:
RET
```



		<p>JS</p> <p>Short Jump if Signed (if negative). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.</p> <p>Algorithm:</p> <pre>if SF = 1 then jump</pre> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 10000000b ; AL = -128 OR AL, 0 ; just set flags. JS label1 PRINT 'not signed.' JMP exit label1: PRINT 'signed.' exit: RET</pre> <div data-bbox="642 1325 852 1421" data-label="Diagram"> <p>A diagram showing the six 8086 flags in a row: Carry (C), Zero (Z), Sign (S), Overflow (O), Parity (P), and Auxiliary (A). Below the row of boxes is the word "unchanged".</p> </div> <div data-bbox="1380 1421 1479 1495" data-label="Image"> <p>Back</p> </div>
--	--	--

	<p>JZ</p> <p>label</p>	<p>Short Jump if Zero (equal). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.</p> <p>Algorithm:</p> <pre>if ZF = 1 then jump</pre> <p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV AL, 5 CMP AL, 5 JZ label1 PRINT 'AL is not equal to 5.'</pre>
--	-------------------------------	---

```
JMP exit
label1:
PRINT 'AL is equal to 5.'
exit:
RET
```

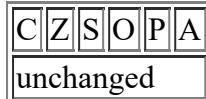


		<p>Load AH from 8 low bits of Flags register.</p> <p>Algorithm:</p> $AH = \text{flags register}$ <p>AH bit: 7 6 5 4 3 2 1 0 [SF] [ZF] [0] [AF] [0] [PF] [1] [CF]</p> <p>bits 1, 3, 5 are reserved.</p>
LAHF	No operands	<p>The diagram shows a flag register with six bits labeled C, Z, S, O, P, and A from left to right. Below the register, the word "unchanged" is written.</p>



LDS	REG, memory	<p>Load memory double word into word register and DS.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • REG = first word • DS = second word <p>Example:</p> <pre>ORG 100h LDS AX, m RET m DW 1234h DW 5678h END</pre>
-----	-------------	--

AX is set to 1234h, DS is set to 5678h.



LEA	REG, memory	<p>Load Effective Address.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • REG = address of memory (offset) <p>Example:</p> <pre>MOV BX, 35h MOV DI, 12h LEA SI, [BX+DI] ; SI = 35h + 12h = 47h</pre> <p>Note: The integrated 8086 assembler automatically replaces LEA with a more efficient MOV where possible. For example:</p> <pre>org 100h LEA AX, m ; AX = offset of m RET m dw 1234h END</pre> <table border="1"> <tr> <td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr> <td colspan="6">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
LES	REG, memory	<p>Load memory double word into word register and ES.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • REG = first word • ES = second word <p>Example:</p> <pre>ORG 100h</pre>												

		<p>LES AX, m</p> <p>RET</p> <p>m DW 1234h DW 5678h</p> <p>END</p> <p>AX is set to 1234h, ES is set to 5678h.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td colspan="6">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
	No operands	<p>Load byte at DS:[SI] into AL. Update SI.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • AL = DS:[SI] • if DF = 0 then <ul style="list-style-type: none"> ◦ SI = SI + 1 else <ul style="list-style-type: none"> ◦ SI = SI - 1 <p>Example:</p> <p>ORG 100h</p> <p>LEA SI, a1 MOV CX, 5 MOV AH, 0Eh</p> <p>m: LODSB INT 10h LOOP m</p> <p>RET</p> <p>a1 DB 'H', 'e', 'l', 'l', 'o'</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td colspan="6">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
LODSB	No operands	<p>Load word at DS:[SI] into AX. Update SI.</p> <p>Algorithm:</p>												

- AX = DS:[SI]
- if DF = 0 then
 - SI = SI + 2
- else
 - SI = SI - 2

Example:

```
ORG 100h
```

```
LEA SI, a1
MOV CX, 5
```

REP LODSW ; finally there will be 555h in AX.

RET

a1 dw 111h, 222h, 333h, 444h, 555h



		Decrease CX, jump to label if CX not zero.												
		<p>Algorithm:</p> <ul style="list-style-type: none"> • CX = CX - 1 • if CX <> 0 then <ul style="list-style-type: none"> ◦ jump else <ul style="list-style-type: none"> ◦ no jump, continue 												
LOOP	label	<p>Example:</p> <pre>include 'emu8086.inc' ORG 100h MOV CX, 5 label1: PRINTN 'loop!' LOOP label1 RET</pre> <table border="1"> <tr> <td>C</td> <td>Z</td> <td>S</td> <td>O</td> <td>P</td> <td>A</td> </tr> <tr> <td colspan="6">unchanged</td> </tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														



LOOPE	label	Decrease CX, jump to label if CX not zero and Equal (ZF = 1).
-------	-------	---

Algorithm:

- CX = CX - 1
- if (CX <> 0) and (ZF = 1) then
 - jump
 - else
 - no jump, continue

Example:

```
; Loop until result fits into AL alone,
; or 5 times. The result will be over 255
; on third loop (100+100+100),
; so loop will exit.
```

```
include 'emu8086.inc'
```

```
ORG 100h
MOV AX, 0
MOV CX, 5
label1:
PUTC '*'
ADD AX, 100
CMP AH, 0
LOOPE label1
RET
```

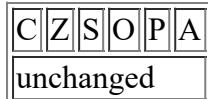


		<p>Decrease CX, jump to label if CX not zero and Not Equal (ZF = 0).</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • CX = CX - 1 • if (CX <> 0) and (ZF = 0) then <ul style="list-style-type: none"> ◦ jump else <ul style="list-style-type: none"> ◦ no jump, continue <p>Example:</p> <p>; Loop until '7' is found, ; or 5 times.</p> <pre>include 'emu8086.inc' ORG 100h MOV SI, 0 MOV CX, 5 label1: PUTC '*'</pre>
--	--	---

```

MOV AL, v1[SI]
INC SI      ; next byte (SI=SI+1).
CMP AL, 7
LOOPNE label1
RET
v1 db 9, 8, 7, 6, 5

```



		<p>Decrease CX, jump to label if CX not zero and ZF = 0.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • CX = CX - 1 • if (CX \neq 0) and (ZF = 0) then <ul style="list-style-type: none"> ◦ jump else ◦ no jump, continue <p>Example:</p> <pre>; Loop until '7' is found, ; or 5 times.</pre> <pre> include 'emu8086.inc' ORG 100h MOV SI, 0 MOV CX, 5 label1: PUTC '*' MOV AL, v1[SI] INC SI ; next byte (SI=SI+1). CMP AL, 7 LOOPNZ label1 RET v1 db 9, 8, 7, 6, 5 </pre> <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td colspan="6">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														



		<p>Decrease CX, jump to label if CX not zero and ZF = 1.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • CX = CX - 1 • if (CX \neq 0) and (ZF = 1) then
--	--	--

- jump
- else
 - no jump, continue

Example:

```
; Loop until result fits into AL alone,  
; or 5 times. The result will be over 255  
; on third loop (100+100+100),  
; so loop will exit.
```

```
include 'emu8086.inc'
```

```
ORG 100h  
MOV AX, 0  
MOV CX, 5  
label1:  
PUTC '*'  
ADD AX, 100  
CMP AH, 0  
LOOPZ label1  
RET
```



MOV

REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate

SREG, memory
memory, SREG
REG, SREG
SREG, REG

Copy operand2 to operand1.

The MOV instruction cannot:

- set the value of the CS and IP registers.
- copy value of one segment register to another segment register (should copy to general register first).
- copy immediate value to segment register (should copy to general register first).

Algorithm:

```
operand1 = operand2
```

Example:

```
ORG 100h  
MOV AX, 0B800h ; set AX = B800h (VGA memory).  
MOV DS, AX ; copy value of AX to DS.  
MOV CL, 'A' ; CL = 41h (ASCII code).  
MOV CH, 0101111b ; CL = color attribute.  
MOV BX, 15Eh ; BX = position on screen.  
MOV [BX], CX ; w.[0B800h:015Eh] = CX.  
RET ; returns to operating system.
```

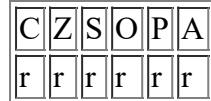




		<p>Copy byte at DS:[SI] to ES:[DI]. Update SI and DI.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • $ES:[DI] = DS:[SI]$ • if $DF = 0$ then <ul style="list-style-type: none"> ◦ $SI = SI + 1$ ◦ $DI = DI + 1$ else <ul style="list-style-type: none"> ◦ $SI = SI - 1$ ◦ $DI = DI - 1$ <p>Example:</p> <pre>ORG 100h CLD LEA SI, a1 LEA DI, a2 MOV CX, 5 REP MOVSB RET a1 DB 1,2,3,4,5 a2 DB 5 DUP(0)</pre>
MOVSB	No operands	<p><code>C Z S O P A</code> unchanged</p>
MOVSW	No operands	<p>Back</p> <p>Copy word at DS:[SI] to ES:[DI]. Update SI and DI.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • $ES:[DI] = DS:[SI]$ • if $DF = 0$ then <ul style="list-style-type: none"> ◦ $SI = SI + 2$ ◦ $DI = DI + 2$ else <ul style="list-style-type: none"> ◦ $SI = SI - 2$ ◦ $DI = DI - 2$ <p>Example:</p>

		<pre>ORG 100h CLD LEA SI, a1 LEA DI, a2 MOV CX, 5 REP MOVSW RET a1 DW 1,2,3,4,5 a2 DW 5 DUP(0) <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td colspan="6">unchanged</td></tr> </table> </pre>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
		<p> Back</p> <p>Unsigned multiply.</p> <p>Algorithm:</p> <p>when operand is a byte: $AX = AL * \text{operand}$.</p> <p>when operand is a word: $(DX\ AX) = AX * \text{operand}$.</p> <p>Example:</p> <pre>MOV AL, 200 ; AL = 0C8h MOV BL, 4 MUL BL ; AX = 0320h (800) RET</pre> <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td>r</td><td>?</td><td>?</td><td>r</td><td>?</td><td>?</td></tr> </table> <p>CF=OF=0 when high section of the result is zero.</p>	C	Z	S	O	P	A	r	?	?	r	?	?
C	Z	S	O	P	A									
r	?	?	r	?	?									
MUL	REG memory	<p> Back</p> <p>Negate. Makes operand negative (two's complement).</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • Invert all bits of the operand • Add 1 to inverted operand <p>Example:</p> <pre>MOV AL, 5 ; AL = 05h NEG AL ; AL = 0FBh (-5)</pre>												

NEG AL ; AL = 05h (5)
RET



NOP	No operands	<p>No Operation.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • Do nothing <p>Example:</p> <p>; do nothing, 3 times: NOP NOP NOP RET</p> <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td colspan="6">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														

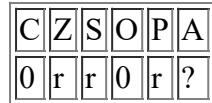
NOT	REG memory	<p>Invert each bit of the operand.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • if bit is 1 turn it to 0. • if bit is 0 turn it to 1. <p>Example:</p> <p>MOV AL, 00011011b NOT AL ; AL = 11100100b RET</p> <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td colspan="6">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														

OR	REG, memory memory, REG REG, REG memory, immediate REG, immediate	<p>Logical OR between all bits of two operands. Result is stored in first operand.</p> <p>These rules apply:</p> <p>1 OR 1 = 1 1 OR 0 = 1 0 OR 1 = 1</p>
----	---	--

0 OR 0 = 0

Example:

```
MOV AL, 'A'      ; AL = 01000001b
OR AL, 00100000b ; AL = 01100001b ('a')
RET
```



OUT	im.byte, AL im.byte, AX DX, AL DX, AX	<p>Output from AL or AX to port. First operand is a port number. If required to access port number over 255 - DX register should be used.</p> <p>Example:</p> <pre>MOV AX, 0FFFh ; Turn on all OUT 4, AX ; traffic lights. MOV AL, 100b ; Turn on the third OUT 7, AL ; magnet of the stepper-motor.</pre> <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td colspan="6">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														

POP	REG SREG memory	<p>Get 16 bit value from the stack.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • operand = SS:[SP] (top of the stack) • SP = SP + 2 <p>Example:</p> <pre>MOV AX, 1234h PUSH AX POP DX ; DX = 1234h RET</pre> <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td colspan="6">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														

POPA	No operands	Pop all general purpose registers DI, SI, BP, SP, BX, DX, CX, AX from the stack.
------	-------------	--

SP value is ignored, it is Popped but not set to SP register).

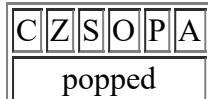
Note: this instruction works only on **80186** CPU and later!

Algorithm:

- POP DI
- POP SI
- POP BP
- POP xx (SP value ignored)
- POP BX
- POP DX
- POP CX
- POP AX



POPF	No operands	Get flags register from the stack.
		<p>Algorithm:</p> <ul style="list-style-type: none"> • flags = SS:[SP] (top of the stack) • SP = SP + 2



PUSH	REG SREG memory immediate	Store 16 bit value in the stack.
		<p>Note: PUSH immediate works only on 80186 CPU and later!</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • SP = SP - 2 • SS:[SP] (top of the stack) = operand <p>Example:</p> <pre>MOV AX, 1234h PUSH AX POP DX ; DX = 1234h RET</pre>





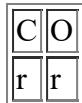
Back

PUSHA	No operands	<p>Push all general purpose registers AX, CX, DX, BX, SP, BP, SI, DI in the stack. Original value of SP register (before PUSHA) is used.</p> <p>Note: this instruction works only on 80186 CPU and later!</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • PUSH AX • PUSH CX • PUSH DX • PUSH BX • PUSH SP • PUSH BP • PUSH SI • PUSH DI <p>A diagram of the flag register (CF, ZF, SF, OF, PF, AF) in a 2x3 grid. The first six columns are labeled C, Z, S, O, P, A respectively. The seventh column contains the word "unchanged".</p>
PUSHF	No operands	<p>Store flags register in the stack.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • SP = SP - 2 • SS:[SP] (top of the stack) = flags <p>A diagram of the flag register (CF, ZF, SF, OF, PF, AF) in a 2x3 grid. The first six columns are labeled C, Z, S, O, P, A respectively. The seventh column contains the word "unchanged".</p>
RCL	memory, immediate REG, immediate memory, CL REG, CL	<p>Rotate operand1 left through Carry Flag. The number of rotates is set by operand2. When immediate is greater than 1, assembler generates several RCL xx, 1 instructions because 8086 has machine code only for this instruction (the same principle works for all other shift/rotate instructions).</p> <p>Algorithm:</p>

shift all bits left, the bit that goes off is set to CF and previous value of CF is inserted to the right-most position.

Example:

```
STC      ; set carry (CF=1).
MOV AL, 1Ch   ; AL = 00011100b
RCL AL, 1    ; AL = 00111001b, CF=0.
RET
```



OF=0 if first operand keeps original sign.



		<p>Rotate operand1 right through Carry Flag. The number of rotates is set by operand2.</p> <p>Algorithm:</p> <p>shift all bits right, the bit that goes off is set to CF and previous value of CF is inserted to the left-most position.</p> <p>Example:</p> <pre>STC ; set carry (CF=1). MOV AL, 1Ch ; AL = 00011100b RCR AL, 1 ; AL = 10001110b, CF=0. RET</pre> <p></p> <p>OF=0 if first operand keeps original sign.</p>
RCR	memory, immediate REG, immediate memory, CL REG, CL	<p>Repeat following MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW instructions CX times.</p> <p>Algorithm:</p> <p>check_cx:</p> <p>if CX \neq 0 then</p> <ul style="list-style-type: none"> • do following <u>chain instruction</u> • CX = CX - 1 • go back to check_cx <p>else</p> <ul style="list-style-type: none"> • exit from REP cycle
REP	chain instruction	



		<p>Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 1 (result is Equal), maximum CX times.</p> <p>Algorithm:</p> <p>check_cx:</p> <pre>if CX <> 0 then • do following <u>chain instruction</u> • CX = CX - 1 • if ZF = 1 then: ○ go back to check_cx else ○ exit from REPE cycle else • exit from REPE cycle</pre> <p>example: open cmpsb.asm from c:\emu8086\examples</p>
REPE	chain instruction	<p>Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 0 (result is Not Equal), maximum CX times.</p> <p>Algorithm:</p> <p>check_cx:</p> <pre>if CX <> 0 then • do following <u>chain instruction</u> • CX = CX - 1 • if ZF = 0 then: ○ go back to check_cx else ○ exit from REPNE cycle else • exit from REPNE cycle</pre>
REPNE	chain instruction	<p>Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 0 (result is Not Equal), maximum CX times.</p> <p>Algorithm:</p> <p>check_cx:</p> <pre>if CX <> 0 then • do following <u>chain instruction</u> • CX = CX - 1 • if ZF = 0 then: ○ go back to check_cx else ○ exit from REPNE cycle else • exit from REPNE cycle</pre>





REPNZ	chain instruction	<p>Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 0 (result is Not Zero), maximum CX times.</p> <p>Algorithm:</p> <p>check_cx:</p> <pre>if CX <> 0 then • do following <u>chain instruction</u> • CX = CX - 1 • if ZF = 0 then: ○ go back to check_cx else ○ exit from REPZ cycle else • exit from REPZ cycle</pre>
REPZ	chain instruction	<p>Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 1 (result is Zero), maximum CX times.</p> <p>Algorithm:</p> <p>check_cx:</p> <pre>if CX <> 0 then • do following <u>chain instruction</u> • CX = CX - 1 • if ZF = 1 then: ○ go back to check_cx else ○ exit from REPZ cycle else • exit from REPZ cycle</pre>



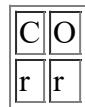


RET	No operands or even immediate	<p>Return from near procedure.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • Pop from stack: <ul style="list-style-type: none"> ◦ IP • if <u>immediate</u> operand is present: $SP = SP + \text{operand}$ <p>Example:</p> <pre>ORG 100h ; for COM file. CALL p1 ADD AX, 1 RET ; return to OS. p1 PROC ; procedure declaration. MOV AX, 1234h RET ; return to caller. p1 ENDP</pre> <div style="border: 1px solid black; padding: 5px; display: inline-block;"> <table border="1" style="margin-bottom: 5px;"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> </table> <p>unchanged</p> </div>	C	Z	S	O	P	A
C	Z	S	O	P	A			
RETF	No operands or even immediate	<p>Return from Far procedure.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • Pop from stack: <ul style="list-style-type: none"> ◦ IP ◦ CS • if <u>immediate</u> operand is present: $SP = SP + \text{operand}$ <div style="border: 1px solid black; padding: 5px; display: inline-block;"> <table border="1" style="margin-bottom: 5px;"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> </table> <p>unchanged</p> </div>	C	Z	S	O	P	A
C	Z	S	O	P	A			
ROL	memory, immediate REG, immediate memory, CL REG, CL	<p>Rotate operand1 left. The number of rotates is set by operand2.</p> <p>Algorithm:</p>						

shift all bits left, the bit that goes off is set to CF and the same bit is inserted to the right-most position.

Example:

```
MOV AL, 1Ch      ; AL = 00011100b
ROL AL, 1        ; AL = 00111000b, CF=0.
RET
```



OF=0 if first operand keeps original sign.



		<p>Rotate operand1 right. The number of rotates is set by operand2.</p> <p>Algorithm:</p> <p>shift all bits right, the bit that goes off is set to CF and the same bit is inserted to the left-most position.</p> <p>Example:</p> <pre>MOV AL, 1Ch ; AL = 00011100b ROR AL, 1 ; AL = 00001110b, CF=0. RET</pre> <p></p> <p>OF=0 if first operand keeps original sign.</p>
ROR	<p>memory, immediate REG, immediate</p> <p>memory, CL REG, CL</p>	

		<p>Store AH register into low 8 bits of Flags register.</p> <p>Algorithm:</p> <p>flags register = AH</p> <p>AH bit: 7 6 5 4 3 2 1 0 [SF] [ZF] [0] [AF] [0] [PF] [1] [CF]</p> <p>bits 1, 3, 5 are reserved.</p> <p></p>
SAHF	No operands	

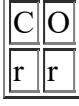
SAL	memory, immediate REG, immediate	Shift Arithmetic operand1 Left. The number of shifts is set by operand2.
-----	-------------------------------------	--

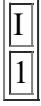
	memory, CL REG, CL	<p>Algorithm:</p> <ul style="list-style-type: none"> Shift all bits left, the bit that goes off is set to CF. Zero bit is inserted to the right-most position. <p>Example:</p> <pre>MOV AL, 0E0h ; AL = 11100000b SAL AL, 1 ; AL = 11000000b, CF=1. RET</pre>  <p>OF=0 if first operand keeps original sign.</p>
SAR	memory, immediate REG, immediate memory, CL REG, CL	<p>Shift Arithmetic operand1 Right. The number of shifts is set by operand2.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> Shift all bits right, the bit that goes off is set to CF. The sign bit that is inserted to the left-most position has the same value as before shift. <p>Example:</p> <pre>MOV AL, 0E0h ; AL = 11100000b SAR AL, 1 ; AL = 11110000b, CF=0. MOV BL, 4Ch ; BL = 01001100b SAR BL, 1 ; BL = 00100110b, CF=0. RET</pre>  <p>OF=0 if first operand keeps original sign.</p>
SBB	REG, memory memory, REG REG, REG memory, immediate REG, immediate	<p>Subtract with Borrow.</p> <p>Algorithm:</p> <p>$\text{operand1} = \text{operand1} - \text{operand2} - \text{CF}$</p> <p>Example:</p> <pre>STC MOV AL, 5 SBB AL, 3 ; AL = 5 - 3 - 1 = 1 RET</pre> 

C	Z	S	O	P	A
r	r	r	r	r	r



SCASB	No operands	<p>Compare bytes: AL from ES:[DI].</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • AL - ES:[DI] • set flags according to result: OF, SF, ZF, AF, PF, CF • if DF = 0 then <ul style="list-style-type: none"> ◦ DI = DI + 1 else <ul style="list-style-type: none"> ◦ DI = DI - 1 <table border="1" data-bbox="641 74 855 864"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr> </table>	C	Z	S	O	P	A	r	r	r	r	r	r
C	Z	S	O	P	A									
r	r	r	r	r	r									
SCASW	No operands	<p>Compare words: AX from ES:[DI].</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • AX - ES:[DI] • set flags according to result: OF, SF, ZF, AF, PF, CF • if DF = 0 then <ul style="list-style-type: none"> ◦ DI = DI + 2 else <ul style="list-style-type: none"> ◦ DI = DI - 2 <table border="1" data-bbox="641 1455 855 1567"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr> </table>	C	Z	S	O	P	A	r	r	r	r	r	r
C	Z	S	O	P	A									
r	r	r	r	r	r									
SHL	<p>memory, immediate REG, immediate</p> <p>memory, CL REG, CL</p>	<p>Shift operand1 Left. The number of shifts is set by operand2.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • Shift all bits left, the bit that goes off is set to CF. • Zero bit is inserted to the right-most position. <p>Example:</p> <pre>MOV AL, 11100000b SHL AL, 1 ; AL = 11000000b, CF=1.</pre>												

		<p>RET</p>  <p>OF=0 if first operand keeps original sign.</p>
		<p>Shift operand1 Right. The number of shifts is set by operand2.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> Shift all bits right, the bit that goes off is set to CF. Zero bit is inserted to the left-most position. <p>Example:</p> <pre>MOV AL, 00000111b SHR AL, 1 ; AL = 00000011b, CF=1.</pre> <p>RET</p>  <p>OF=0 if first operand keeps original sign.</p>
SHR	memory, immediate REG, immediate memory, CL REG, CL	
STC	No operands	<p>Set Carry flag.</p> <p>Algorithm:</p> <p>CF = 1</p> 
STD	No operands	<p>Set Direction flag. SI and DI will be decremented by chain instructions: CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW.</p> <p>Algorithm:</p> <p>DF = 1</p> 

STI	No operands	<p>Set Interrupt enable flag. This enables hardware interrupts.</p> <p>Algorithm:</p> <p>IF = 1</p>  <p> Back</p>
STOSB	No operands	<p>Store byte in AL into ES:[DI]. Update DI.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • ES:[DI] = AL • if DF = 0 then <ul style="list-style-type: none"> ◦ DI = DI + 1 else <ul style="list-style-type: none"> ◦ DI = DI - 1 <p>Example:</p> <pre>ORG 100h LEA DI, a1 MOV AL, 12h MOV CX, 5 REP STOSB RET a1 DB 5 dup(0)</pre>  <p> Back</p>
STOSW	No operands	<p>Store word in AX into ES:[DI]. Update DI.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> • ES:[DI] = AX • if DF = 0 then <ul style="list-style-type: none"> ◦ DI = DI + 2 else <ul style="list-style-type: none"> ◦ DI = DI - 2 <p>Example:</p>

		<p>ORG 100h</p> <p>LEA DI, a1 MOV AX, 1234h MOV CX, 5</p> <p>REP STOSW</p> <p>RET</p> <p>a1 DW 5 dup(0)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td colspan="6" style="text-align: center;">unchanged</td></tr> </table>	C	Z	S	O	P	A	unchanged						 Back
C	Z	S	O	P	A										
unchanged															
SUB	REG, memory memory, REG REG, REG memory, immediate REG, immediate	<p>Subtract.</p> <p>Algorithm:</p> <p>operand1 = operand1 - operand2</p> <p>Example:</p> <p>MOV AL, 5 SUB AL, 1 ; AL = 4</p> <p>RET</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr> <tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr> </table>	C	Z	S	O	P	A	r	r	r	r	r	r	 Back
C	Z	S	O	P	A										
r	r	r	r	r	r										
TEST	REG, memory memory, REG REG, REG memory, immediate REG, immediate	<p>Logical AND between all bits of two operands for flags only. These flags are effected: ZF, SF, PF. Result is not stored anywhere.</p> <p>These rules apply:</p> <p>1 AND 1 = 1 1 AND 0 = 0 0 AND 1 = 0 0 AND 0 = 0</p> <p>Example:</p> <p>MOV AL, 00000101b TEST AL, 1 ; ZF = 0. TEST AL, 10b ; ZF = 1. RET</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td></tr> </table>													



XCHG	REG, memory memory, REG REG, REG	<p>Exchange values of two operands.</p> <p>Algorithm:</p> <p>operand1 < - > operand2</p> <p>Example:</p> <pre>MOV AL, 5 MOV AH, 2 XCHG AL, AH ; AL = 2, AH = 5 XCHG AL, AH ; AL = 5, AH = 2 RET</pre> <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td></tr> <tr><td>A</td><td>unchanged</td><td></td><td></td><td></td></tr> </table>	C	Z	S	O	P	A	unchanged			
C	Z	S	O	P								
A	unchanged											
XLATB	No operands	<p>Translate byte from table. Copy value of memory byte at DS:[BX + unsigned AL] to AL register.</p> <p>Algorithm:</p> <p>AL = DS:[BX + unsigned AL]</p> <p>Example:</p> <pre>ORG 100h LEA BX, dat MOV AL, 2 XLATB ; AL = 33h RET</pre> <p>dat DB 11h, 22h, 33h, 44h, 55h</p> <table border="1"> <tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td></tr> <tr><td>A</td><td>unchanged</td><td></td><td></td><td></td></tr> </table>	C	Z	S	O	P	A	unchanged			
C	Z	S	O	P								
A	unchanged											
XOR	REG, memory memory, REG REG, REG memory, immediate REG, immediate	<p>Logical XOR (Exclusive OR) between all bits of two operands. Result is stored in first operand.</p> <p>These rules apply:</p>										

1	XOR	1	=	0
1	XOR	0	=	1
0	XOR	1	=	1
0	XOR	0	=	0

Example:

```
MOV AL, 00000111b  
XOR AL, 00000010b ; AL = 00000101b  
RET
```

C	Z	S	O	P	A
0	r	r	0	r	?