

# Complete 8086 instruction set

---

Quick reference:

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

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
		<p>ASCII Adjust after Addition. Corrects result in AH and AL after addition when working with BCD values.</p> <p>It works according to the following Algorithm:</p> <p>if low nibble of AL &gt; 9 or AF = 1 then:</p>

AAA	No operands	<ul style="list-style-type: none"><li>• <math>AL = AL + 6</math></li><li>• <math>AH = AH + 1</math></li><li>• <math>AF = 1</math></li><li>• <math>CF = 1</math></li></ul> <p>else</p> <ul style="list-style-type: none"><li>• <math>AF = 0</math></li><li>• <math>CF = 0</math></li></ul> <p>in both cases: clear the high nibble of AL.</p> <p>Example:</p> <pre>MOV AX, 15    ; AH = 00, AL = 0Fh AAA           ; AH = 01, AL = 05 RET</pre> <table><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									
AAD	No operands	<p>ASCII Adjust before Division. Prepares two BCD values for division.</p> <p>Algorithm:</p> <ul style="list-style-type: none"><li>• <math>AL = (AH * 10) + AL</math></li><li>• <math>AH = 0</math></li></ul> <p>Example:</p> <pre>MOV AX, 0105h ; AH = 01, AL = 05 AAD           ; AH = 00, AL = 0Fh (15) RET</pre> <table><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	?									
		<p>ASCII Adjust after Multiplication. Corrects the result of multiplication of two BCD values.</p> <p>Algorithm:</p> <ul style="list-style-type: none"><li>• <math>AH = AL / 10</math></li><li>• <math>AL = \text{remainder}</math></li></ul>												

## Example:

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

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



AAM

No operands

## ASCII Adjust after Subtraction.

Corrects result in AH and AL after subtraction when working with BCD values.

## 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, 02FFh ; AH = 02, AL = 0FFh
AAS           ; AH = 01, AL = 09
RET
```

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



AAS

No operands

## Add with Carry.




## Algorithm:

operand1 = operand1 + operand2 + CF





## Example:

ADC

REG, memory  
memory, REG  
REG, REG

	memory, immediate REG, immediate	<pre>STC      ; set CF = 1 MOV AL, 5 ; AL = 5 ADC AL, 1 ; AL = 7 RET</pre> <div> <div>CZSOPA</div> <div>rrrrrrr</div> </div> 
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> <div> <div>CZSOPA</div> <div>rrrrrrr</div> </div> 
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> <pre>1 AND 1 = 1 1 AND 0 = 0 0 AND 1 = 0 0 AND 0 = 0</pre> <p>Example:</p> <pre>MOV AL, 'a' ; AL = 01100001b AND AL, 11011111b ; AL = 01000001b ('A') RET</pre> <div> <div>CZSOPA</div> <div>0rr0r</div> </div> 
		<p>Transfers control to procedure, return address is (IP) is pushed to stack. <i>4-byte address</i> may be entered in this form: 1234h:5678h, first value is a</p>

CALL	procedure name label 4-byte address	<p>segment second value is an offset (this is a far call, so CS is also pushed to stack).</p> <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><div>CZSOPA</div><div>unchanged</div></div> <div></div>
CBW	No operands	<p>Convert byte into word.</p> <p>Algorithm:</p> <p>if high bit of AL = 1 then:</p> <ul style="list-style-type: none"><li>AH = 255 (0FFh)</li></ul> <p>else</p> <ul style="list-style-type: none"><li>AH = 0</li></ul> <p>Example:</p> <pre>MOV AX, 0 ; AH = 0, AL = 0 MOV AL, -5 ; AX = 000FBh (251) CBW ; AX = 0FFFBh (-5) RET</pre> <div><div>CZSOPA</div><div>unchanged</div></div> <div></div>
		<p>Clear Carry flag.</p> <p>Algorithm:</p> <p>CF = 0</p>

CLC	No operands	<div> <div>C</div> <div>0</div> </div> 
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> $DF = 0$ <div> <div>D</div> <div>0</div> </div> 
CLI	No operands	<p>Clear Interrupt enable flag. This disables hardware interrupts.</p> <p>Algorithm:</p> $IF = 0$ <div> <div>I</div> <div>0</div> </div> 
CMC	No operands	<p>Complement Carry flag. Inverts value of CF.</p> <p>Algorithm:</p> <pre>if CF = 1 then CF = 0 if CF = 0 then CF = 1</pre> <div> <div>C</div> <div>r</div> </div> 
		<p>Compare.</p> <p>Algorithm:</p> $\text{operand1} - \text{operand2}$

CMP	REG, memory memory, REG REG, REG memory, immediate REG, immediate	<p>result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.</p> <p>Example:</p> <pre>MOV AL, 5 MOV BL, 5 CMP AL, BL ; AL = 5, ZF = 1 (so equal!) RET</pre> <div><table><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></div> <div></div>	C	Z	S	O	P	A	r	r	r	r	r	r
C	Z	S	O	P	A									
r	r	r	r	r	r									
CMPSB	No operands	<p>Compare bytes: ES:[DI] from DS:[SI].</p> <p>Algorithm:</p> <ul style="list-style-type: none"><li>• DS:[SI] - ES:[DI]</li><li>• set flags according to result: OF, SF, ZF, AF, PF, CF</li><li>• if DF = 0 then<ul style="list-style-type: none"><li>◦ SI = SI + 1</li><li>◦ DI = DI + 1</li></ul></li><li>else<ul style="list-style-type: none"><li>◦ SI = SI - 1</li><li>◦ DI = DI - 1</li></ul></li></ul> <p>Example: open <b>cmpsb.asm</b> from c:\emu8086\examples</p> <div><table><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></div> <div></div>	C	Z	S	O	P	A	r	r	r	r	r	r
C	Z	S	O	P	A									
r	r	r	r	r	r									
CMPSW	No operands	<p>Compare words: ES:[DI] from DS:[SI].</p> <p>Algorithm:</p> <ul style="list-style-type: none"><li>• DS:[SI] - ES:[DI]</li><li>• set flags according to result: OF, SF, ZF, AF, PF, CF</li><li>• if DF = 0 then<ul style="list-style-type: none"><li>◦ SI = SI + 2</li><li>◦ DI = DI + 2</li></ul></li><li>else<ul style="list-style-type: none"><li>◦ SI = SI - 2</li><li>◦ DI = DI - 2</li></ul></li></ul>												



example:

open **cmpsw.asm** from c:\emu8086\examples

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



Convert Word to Double word.

Algorithm:

if high bit of AX = 1 then:

- DX = 65535 (0FFFFh)

else

- DX = 0

Example:

```
MOV DX, 0    ; DX = 0
MOV AX, 0    ; AX = 0
MOV AX, -5   ; DX AX = 0000h:0FFFBh
CWD          ; DX AX = 0FFFFh:0FFFBh
RET
```

C	Z	S	O	P	A
unchanged					



Decimal adjust After Addition.

Corrects the result of addition of two packed BCD values.

Algorithm:

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

- AL = AL + 6
- AF = 1

if AL > 9Fh or CF = 1 then:

- AL = AL + 60h
- CF = 1

Example:

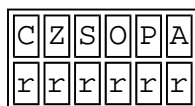
CWD

No operands

DAA

No operands

```
MOV AL, 0Fh ; AL = 0Fh (15)
DAA          ; AL = 15h
RET
```



DAS

No operands

Decimal adjust After Subtraction.  
Corrects the result of subtraction of two packed BCD values.

Algorithm:

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

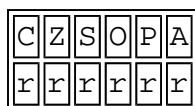
- AL = AL - 6
- AF = 1

if AL > 9Fh or CF = 1 then:

- AL = AL - 60h
- CF = 1

Example:

```
MOV AL, 0FFh ; AL = 0FFh (-1)
DAS          ; AL = 99h, CF = 1
RET
```



DEC

REG  
memory

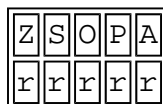
Decrement.

Algorithm:

operand = operand - 1

Example:




```
MOV AL, 255 ; AL = 0FFh (255 or -1)
DEC AL      ; AL = 0FEh (254 or -2)
RET
```



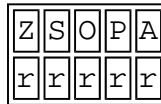
CF - unchanged!



DIV	REG memory	<div>Unsigned divide.</div> <div>Algorithm:</div> <div><div>when operand is a <b>byte</b>:</div><div>AL = AX / operand</div><div>AH = remainder (modulus)</div><div>when operand is a <b>word</b>:</div><div>AX = (DX AX) / operand</div><div>DX = remainder (modulus)</div></div> <div>Example:</div> <div>MOV AX, 203 ; AX = 00CBh</div> <div>MOV BL, 4</div> <div>DIV BL ; AL = 50 (32h), AH = 3</div> <div>RET</div> <div><table><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></div> <div></div>	C	Z	S	O	P	A	?	?	?	?	?	?
C	Z	S	O	P	A									
?	?	?	?	?	?									
HLT	No operands	<div>Halt the System.</div> <div>Example:</div> <div>MOV AX, 5</div> <div>HLT</div> <div><table><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></div>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
IDIV	REG memory	<div>Signed divide.</div> <div>Algorithm:</div> <div><div>when operand is a <b>byte</b>:</div><div>AL = AX / operand</div><div>AH = remainder (modulus)</div><div>when operand is a <b>word</b>:</div><div>AX = (DX AX) / operand</div><div>DX = remainder (modulus)</div></div> <div>Example:</div> <div>MOV AX, -203 ; AX = 0FF35h</div> <div>MOV BL, 4</div> <div>IDIV BL ; AL = -50 (0CEh), AH = -3 (0FDh)</div> <div>RET</div>												

		<div> <div> CZSOPA  ?? ?? ?? </div> <div>  </div> </div>
IMUL	REG memory	<p>Signed multiply.</p> <p>Algorithm:</p> <p>when operand is a <b>byte</b>:  <math>AX = AL * \text{operand}</math>.</p> <p>when operand is a <b>word</b>:  <math>(DX\ AX) = AX * \text{operand}</math>.</p> <p>Example:</p> <pre>MOV AL, -2 MOV BL, -4 IMUL BL      ; AX = 8 RET</pre> <div> <div> CZSOPA  r??r?? </div> <p>CF=OF=0 when result fits into operand of IMUL.</p> <div>  </div> </div>
IN	AL, im.byte AL, DX AX, im.byte AX, DX	<p>Input from port into <b>AL</b> or <b>AX</b>.  Second operand is a port number. If required to access port number over 255 - <b>DX</b> 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> <div> <div> CZSOPA  unchanged </div> <div>  </div> </div>
INC	REG memory	<p>Increment.</p> <p>Algorithm:</p> $\text{operand} = \text{operand} + 1$ <p>Example:</p> <pre>MOV AL, 4 INC AL      ; AL = 5</pre>

RET



CF - unchanged!



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
```



Interrupt Return.

IRET	No operands	<p>Algorithm:</p> <p>Pop from stack:</p> <ul style="list-style-type: none"><li>o IP</li><li>o CS</li><li>o flags register</li></ul> <div><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">popped</td></tr></table></div> <div></div>	C	Z	S	O	P	A	popped					
C	Z	S	O	P	A									
popped														
JA	label	<p>Short Jump if first operand is Above second operand (as set by CMP instruction). Unsigned.</p> <p>Algorithm:</p> <p>if (CF = 0) and (ZF = 0) then jump</p> <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> <div><table><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></div>	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> <p>if CF = 0 then jump</p> <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:</pre>												

```

    PRINT 'AL is above or equal to 5'
exit:
    RET

```



Short Jump if first operand is Below second operand (as set by CMP instruction). Unsigned.

Algorithm:

if CF = 1 then jump

Example:

```

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

```



Short Jump if first operand is Below or Equal to 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, 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

```






JB

label

JBE

label

		<div> <div>CZSOPA</div> <div>unchanged</div> </div> 
JC	label	<p>Short Jump if Carry flag is set to 1.</p> <p>Algorithm:</p> <p>if CF = 1 then jump</p> <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> <div>CZSOPA</div> <div>unchanged</div> </div> 
JCXZ	label	<p>Short Jump if CX register is 0.</p> <p>Algorithm:</p> <p>if CX = 0 then jump</p> <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> <div>CZSOPA</div> <div>unchanged</div> </div> 
		<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
```



JE

label

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

Algorithm:

if (ZF = 0) and (SF = OF) then jump

Example:

```
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
```







JG

label

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

Algorithm:

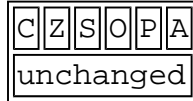
JGE	label	<div>if SF = OF then jump</div> <div>Example:</div> <div><pre>include 'emu8086.inc' ORG 100h MOV AL, 2 CMP AL, -5 JGE label1 PRINT 'AL &lt; -5' JMP exit label1:     PRINT 'AL &gt;= -5' exit:     RET</pre></div> <div><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">unchanged</td></tr></table></div> <div></div>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
JL	label	<div>Short Jump if first operand is Less then second operand (as set by CMP instruction). Signed.</div> <div>Algorithm:</div> <div>if SF &lt;&gt; OF then jump</div> <div>Example:</div> <div><pre>include 'emu8086.inc' ORG 100h MOV AL, -2 CMP AL, 5 JL label1 PRINT 'AL &gt;= 5.' JMP exit label1:     PRINT 'AL &lt; 5.' exit:     RET</pre></div> <div><table><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></div>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
		<div>Short Jump if first operand is Less or Equal to second operand (as set by CMP instruction). Signed.</div> <div>Algorithm:</div> <div>if SF &lt;&gt; OF or ZF = 1 then jump</div> <div>Example:</div>												

JLE	label	<pre>include 'emu8086.inc' ORG 100h MOV AL, -2 CMP AL, 5 JLE label1 PRINT 'AL &gt; 5.' JMP exit label1: PRINT 'AL &lt;= 5.' exit: RET</pre> <div> <div>CZSOPA</div> <div>unchanged</div> </div> 
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>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> <div>CZSOPA</div> <div>unchanged</div> </div> 
JNA	label	<p>Short Jump if first operand is Not Above second operand (as set by CMP instruction). Unsigned.</p> <p>Algorithm:</p> <p>if CF = 1 or ZF = 1 then jump</p> <p>Example:</p> <pre>include 'emu8086.inc'  ORG 100h MOV AL, 2</pre>

```

    CMP AL, 5
    JNA label1
    PRINT 'AL is above 5.'
    JMP exit
label1:
    PRINT 'AL is not above 5.'
exit:
    RET

```



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

```



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.'

```

JNAE

label

JNB

label

```

    JMP exit
label1:
    PRINT 'AL >= 5.'
exit:
    RET

```



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

Algorithm:

if (CF = 0) and (ZF = 0) then jump

Example:

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

```

ORG 100h
MOV AL, 7
CMP AL, 5
JNBE label1
PRINT 'AL <= 5.'
JMP exit

```

```

label1:
    PRINT 'AL > 5.'
exit:
    RET

```



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:



```

JNBE

label

JNC

label

		<p>RET</p> <div> <div>CZSOPA</div> <div>unchanged</div> </div> 
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> <p>if ZF = 0 then jump</p> <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 &lt;&gt; 3.' exit: RET</pre> <div> <div>CZSOPA</div> <div>unchanged</div> </div> 
JNG	label	<p>Short Jump if first operand is Not Greater then second operand (as set by CMP instruction). Signed.</p> <p>Algorithm:</p> <p>if (ZF = 1) and (SF &lt;&gt; OF) then jump</p> <p>Example:</p> <pre>include 'emu8086.inc'  ORG 100h MOV AL, 2 CMP AL, 3 JNG label1 PRINT 'AL &gt; 3.' JMP exit label1: PRINT 'Al &lt;= 3.' exit: RET</pre>

C	Z	S	O	P	A
unchanged					



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
```

C	Z	S	O	P	A
unchanged					



Short Jump if first operand is Not Less then 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
JNL label1
PRINT 'AL < -3.'
JMP exit
label1:
PRINT 'Al >= -3.'
exit:
RET
```

C	Z	S	O	P	A
---	---	---	---	---	---

unchanged



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

Algorithm:

if (SF = OF) and (ZF = 0) then jump

Example:

```
include 'emu8086.inc'

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

C	Z	S	O	P	A
unchanged					



Short Jump if Not Overflow.

Algorithm:

if OF = 0 then jump

Example:

```
; -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
```

C	Z	S	O	P	A
---	---	---	---	---	---

JNLE

label

JNO

label



unchanged



Short Jump if No 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.
JNP label1
PRINT 'parity even.'
JMP exit
label1:
PRINT 'parity odd.'
exit:
RET
```

C	Z	S	O	P	A
unchanged					



Short Jump if Not Signed (if positive). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.

Algorithm:

if SF = 0 then jump

Example:

```
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
```




C	Z	S	O	P	A
unchanged					



JNP

label

JNS

label

		
JNZ	label	<p>Short Jump if Not Zero (not equal). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.</p> <p>Algorithm:</p> <p>if ZF = 0 then jump</p> <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> <div>CZSOPA</div> <div>unchanged</div> </div> 
JO	label	<p>Short Jump if Overflow.</p> <p>Algorithm:</p> <p>if OF = 1 then jump</p> <p>Example:</p> <pre>; -5 - 127 = -132 (not in -128..127) ; the result of SUB is wrong (124), ; so OF = 1 is set:  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> <div>CZSOPA</div> <div>unchanged</div> </div> 

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: PRINT 'parity even.' exit: RET</pre> <div><table><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></div>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
JPE	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. JPE label1 PRINT 'parity odd.' JMP exit label1: PRINT 'parity even.' exit: RET</pre> <div><table><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></div>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
		<p>Short Jump if Parity Odd. Only 8 low bits of result</p>												

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
```

C	Z	S	O	P	A
unchanged					



Short Jump if Signed (if negative). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.

Algorithm:

if SF = 1 then jump

Example:

```
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
```

C	Z	S	O	P	A
unchanged					



Short Jump if Zero (equal). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.

**Algorithm:**

```
if ZF = 1 then jump
```

**Example:**

```
include 'emu8086.inc'

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



JZ

label

Load AH from 8 low bits of Flags register.

**Algorithm:**

```
AH = flags register
```

```
AH bit:  7    6    5    4    3    2    1    0
         [SF] [ZF] [0] [AF] [0] [PF] [1] [CF]
```

bits 1, 3, 5 are reserved.



LAHF

No operands



Load memory double word into word register and DS.


**Algorithm:**

- REG = first word
- DS = second word

**Example:**

```
ORG 100h
```

LDS	REG, memory	<pre>LDS AX, m  RET  m  DW  1234h     DW  5678h  END</pre> <p>AX is set to 1234h, DS is set to 5678h.</p> <div> <div>CZSOPA</div> <div>unchanged</div> </div> 
LEA	REG, memory	<p>Load Effective Address.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> <li>REG = address of memory (offset)</li> </ul> <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 <b>LEA</b> with a more efficient <b>MOV</b> where possible. For example:</p> <pre>org 100h LEA AX, m          ; AX = offset of m RET m  dw  1234h END</pre> <div> <div>CZSOPA</div> <div>unchanged</div> </div> 
		<p>Load memory double word into word register and ES.</p> <p>Algorithm:</p>

LES	REG, memory	<ul style="list-style-type: none"><li>• REG = first word</li><li>• ES = second word</li></ul> <p>Example:</p> <pre>ORG 100h LES AX, m RET m DW 1234h    DW 5678h END</pre> <p>AX is set to 1234h, ES is set to 5678h.</p> <div><div>CZSOPA</div><div>unchanged</div></div> <div></div>
LODSB	No operands	<p>Load byte at DS: [SI] into AL. Update SI.</p> <p>Algorithm:</p> <ul style="list-style-type: none"><li>• AL = DS:[SI]</li><li>• if DF = 0 then<ul style="list-style-type: none"><li>◦ SI = SI + 1</li></ul></li><li>else<ul style="list-style-type: none"><li>◦ SI = SI - 1</li></ul></li></ul> <p>Example:</p> <pre>ORG 100h LEA SI, a1 MOV CX, 5 MOV AH, 0Eh  m: LODSB INT 10h LOOP m  RET  a1 DB 'H', 'e', 'l', 'l', 'o'</pre> <div><div>CZSOPA</div><div>unchanged</div></div>



Load word at DS:[SI] into AX. Update SI.

Algorithm:

- $AX = DS:[SI]$
- 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
```

C	Z	S	O	P	A
unchanged					



Decrease CX, jump to label if CX not zero.

Algorithm:

- $CX = CX - 1$
- if  $CX \neq 0$  then
  - jump
- else
  - no jump, continue

Example:

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

```
ORG 100h
```

```
MOV CX, 5
```

```
label1:
```

```
PRINTN 'loop!'
```

```
LOOP label1
```

```
RET
```

C	Z	S	O	P	A
unchanged					

LODSW

No operands

LOOP

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
```



LOOPE

label

Decrease CX, jump to label if CX not zero and Not Equal (ZF = 0).

Algorithm:

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

Example:

```
; Loop until '7' is found,
; or 5 times.
```

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

```
ORG 100h
```

LOOPNE

label

```

MOV SI, 0
MOV CX, 5
label1:
  PUTC '*'
  MOV AL, v1[SI]
  INC SI          ; next byte (SI=SI+1).
  CMP AL, 7
  LOOPNE label1
  RET
v1 db 9, 8, 7, 6, 5

```

C	Z	S	O	P	A
unchanged					



Decrease CX, jump to label if CX not zero and ZF = 0.

Algorithm:

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

Example:

```

; Loop until '7' is found,
; or 5 times.

```

```
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

```

C	Z	S	O	P	A
unchanged					





Decrease CX, jump to label if CX not zero and ZF = 1.

Algorithm:

- CX = CX - 1

LOOPNZ

label

LOOPZ	label	<div><ul style="list-style-type: none"><li>• if (CX &lt;&gt; 0) and (ZF = 1) then<ul style="list-style-type: none"><li>◦ jump</li></ul></li><li>else<ul style="list-style-type: none"><li>◦ no jump, continue</li></ul></li></ul></div> <div>Example:</div> <div><pre>; 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</pre></div> <div><div>CZSOPA</div><div>unchanged</div></div> <div></div>
MOV	<div>REG, memory</div> <div>memory, REG</div> <div>REG, REG</div> <div>memory, immediate</div> <div>REG, immediate</div> <div>SREG, memory</div> <div>memory, SREG</div> <div>REG, SREG</div> <div>SREG, REG</div>	<div>Copy operand2 to operand1.</div> <div>The MOV instruction <u>cannot</u>:</div> <div><ul style="list-style-type: none"><li>• set the value of the CS and IP registers.</li><li>• copy value of one segment register to another segment register (should copy to general register first).</li><li>• copy immediate value to segment register (should copy to general register first).</li></ul></div> <div>Algorithm:</div> <div>operand1 = operand2</div> <div>Example:</div> <div><pre>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, 01011111b   ; CL = color attribute. MOV BX, 15Eh        ; BX = position on screen. MOV [BX], CX         ; w.[0B800h:015Eh] = CX. RET                 ; returns to operating system.</pre></div> <div></div>

C	Z	S	O	P	A
unchanged					



MOVSB

No operands

Copy byte at DS:[SI] to ES:[DI]. Update SI and DI.

Algorithm:

- ES:[DI] = DS:[SI]
- if DF = 0 then
  - SI = SI + 1
  - DI = DI + 1
- else
  - SI = SI - 1
  - DI = DI - 1

Example:

```
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)
```

C	Z	S	O	P	A
unchanged					





Copy **word** at DS:[SI] to ES:[DI]. Update SI and DI.

Algorithm:

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

Example:

MOVSW	No operands	<div>ORG 100h</div> <div>CLD</div> <div>LEA SI, a1</div> <div>LEA DI, a2</div> <div>MOV CX, 5</div> <div>REP MOVSW</div> <div>RET</div> <div>a1 DW 1,2,3,4,5</div> <div>a2 DW 5 DUP(0)</div> <div><table><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></div>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
MUL	REG memory	<div>Unsigned multiply.</div> <div>Algorithm:</div> <div>when operand is a <b>byte</b>:</div> <div>AX = AL * operand.</div> <div>when operand is a <b>word</b>:</div> <div>(DX AX) = AX * operand.</div> <div>Example:</div> <div>MOV AL, 200 ; AL = 0C8h</div> <div>MOV BL, 4</div> <div>MUL BL ; AX = 0320h (800)</div> <div>RET</div> <div><table><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></div> <div>CF=OF=0 when high section of the result is zero.</div> <div></div>	C	Z	S	O	P	A	r	?	?	r	?	?
C	Z	S	O	P	A									
r	?	?	r	?	?									
NEG	REG memory	<div>Negate. Makes operand negative (two's complement).</div> <div>Algorithm:</div> <div><ul style="list-style-type: none"><li>• Invert all bits of the operand</li><li>• Add 1 to inverted operand</li></ul></div> <div>Example:</div> <div>MOV AL, 5 ; AL = 05h</div> <div>NEG AL ; AL = 0FBh (-5)</div> <div>NEG AL ; AL = 05h (5)</div>												

RET

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



NOP

No operands

No Operation.

Algorithm:

- Do nothing

Example:

```
; do nothing, 3 times:
NOP
NOP
NOP
RET
```

C	Z	S	O	P	A
unchanged					



NOT

REG  
memory

Invert each bit of the operand.

Algorithm:

- if bit is 1 turn it to 0.
- if bit is 0 turn it to 1.

Example:

```
MOV AL, 00011011b
NOT AL   ; AL = 11100100b
RET
```

C	Z	S	O	P	A
unchanged					








Logical OR between all bits of two operands.  
Result is stored in first operand.

These rules apply:




```
1 OR 1 = 1
1 OR 0 = 1
0 OR 1 = 1
0 OR 0 = 0
```

REG, memory



OR	memory, REG REG, REG memory, immediate REG, immediate	<p>Example:</p> <pre>MOV AL, 'A'           ; AL = 01000001b OR AL, 00100000b      ; AL = 01100001b ('a') RET</pre> <table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>0</td><td>r</td><td>r</td><td>0</td><td>r</td><td>?</td></tr></table> 	C	Z	S	O	P	A	0	r	r	0	r	?
C	Z	S	O	P	A									
0	r	r	0	r	?									
OUT	im.byte, AL im.byte, AX DX, AL DX, AX	<p>Output from <b>AL</b> or <b>AX</b> to port. First operand is a port number. If required to access port number over 255 - <b>DX</b> 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><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"><li>operand = SS:[SP] (top of the stack)</li><li>SP = SP + 2</li></ul> <p>Example:</p> <pre>MOV AX, 1234h PUSH AX POP  DX      ; DX = 1234h RET</pre> <table><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 all general purpose registers DI, SI, BP, SP, BX, DX, CX, AX from the stack.												



POPA	No operands	<p>SP value is ignored, it is Popped but not set to SP register).</p> <p>Note: this instruction works only on <b>80186</b> CPU and later!</p> <p>Algorithm:</p> <ul style="list-style-type: none"><li>• POP DI</li><li>• POP SI</li><li>• POP BP</li><li>• POP xx (SP value ignored)</li><li>• POP BX</li><li>• POP DX</li><li>• POP CX</li><li>• POP AX</li></ul> <div><table><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></div>	C	Z	S	O	P	A	unchanged					
C	Z	S	O	P	A									
unchanged														
POPF	No operands	<p>Get flags register from the stack.</p> <p>Algorithm:</p> <ul style="list-style-type: none"><li>• flags = SS:[SP] (top of the stack)</li><li>• SP = SP + 2</li></ul> <div><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">popped</td></tr></table></div> <div></div>	C	Z	S	O	P	A	popped					
C	Z	S	O	P	A									
popped														
PUSH	REG SREG memory immediate	<p>Store 16 bit value in the stack.</p> <p>Note: <b>PUSH immediate</b> works only on 80186 CPU and later!</p> <p>Algorithm:</p> <ul style="list-style-type: none"><li>• SP = SP - 2</li><li>• SS:[SP] (top of the stack) = operand</li></ul> <p>Example:</p> <pre>MOV AX, 1234h PUSH AX POP DX      ; DX = 1234h RET</pre> <div></div>												



		<div> <div>CZSOPA</div> <div>unchanged</div> </div> 
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 <b>80186</b> CPU and later!</p> <p>Algorithm:</p> <ul style="list-style-type: none"> <li>• PUSH AX</li> <li>• PUSH CX</li> <li>• PUSH DX</li> <li>• PUSH BX</li> <li>• PUSH SP</li> <li>• PUSH BP</li> <li>• PUSH SI</li> <li>• PUSH DI</li> </ul> <div> <div>CZSOPA</div> <div>unchanged</div> </div> 
PUSHF	No operands	<p>Store flags register in the stack.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> <li>• <math>SP = SP - 2</math></li> <li>• <math>SS:[SP]</math> (top of the stack) = flags</li> </ul> <div> <div>CZSOPA</div> <div>unchanged</div> </div> 
		<p>Rotate operand1 left through Carry Flag. The number of rotates is set by operand2. When <b>immediate</b> is greater than 1, assembler generates several <b>RCL xx, 1</b> instructions because 8086 has machine code only for this instruction (the same principle works for all other shift/rotate instructions).</p> <p>Algorithm:</p>

RCL	memory, immediate REG, immediate  memory, CL REG, CL	<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.</p> <p>Example:</p> <pre>STC                ; set carry (CF=1). MOV AL, 1Ch        ; AL = 00011100b RCL AL, 1           ; AL = 00111001b,  CF=0. RET</pre> <div><div>C</div><div>O</div><div>r</div><div>r</div></div> <p>OF=0 if first operand keeps original sign.</p> <div></div>
RCR	memory, immediate REG, immediate  memory, CL REG, CL	<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> <div><div>C</div><div>O</div><div>r</div><div>r</div></div> <p>OF=0 if first operand keeps original sign.</p> <div></div>
		<p>Repeat following MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW instructions CX times.</p> <p>Algorithm:</p> <p>check_cx:</p> <p>if CX &lt;&gt; 0 then</p> <ul style="list-style-type: none"><li>do following <u>chain instruction</u></li></ul>

REP	chain instruction	<ul style="list-style-type: none"> <li>• <math>CX = CX - 1</math></li> <li>• go back to <code>check_cx</code></li> </ul> <p>else</p> <ul style="list-style-type: none"> <li>• exit from REP cycle</li> </ul> <div data-bbox="691 371 730 472" style="border: 1px solid black; padding: 2px; display: inline-block;"> <div style="border: 1px solid black; padding: 2px; display: inline-block;">Z</div>  <div style="border: 1px solid black; padding: 2px; display: inline-block;">r</div> </div> <div data-bbox="1406 472 1485 539" style="text-align: right;">  </div>
REPE	chain instruction	<p>Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 1 (result is Equal), maximum CX times.</p> <p>Algorithm:</p> <p><code>check_cx:</code></p> <p>if <math>CX \neq 0</math> then</p> <ul style="list-style-type: none"> <li>• do following <u>chain instruction</u></li> <li>• <math>CX = CX - 1</math></li> <li>• if ZF = 1 then: <ul style="list-style-type: none"> <li>◦ go back to <code>check_cx</code></li> </ul> </li> <li>else <ul style="list-style-type: none"> <li>◦ exit from REPE cycle</li> </ul> </li> </ul> <p>else</p> <ul style="list-style-type: none"> <li>• exit from REPE cycle</li> </ul> <p>example: open <b>cmplib.asm</b> from <code>c:\emu8086\examples</code></p> <div data-bbox="691 1503 730 1603" style="border: 1px solid black; padding: 2px; display: inline-block;"> <div style="border: 1px solid black; padding: 2px; display: inline-block;">Z</div>  <div style="border: 1px solid black; padding: 2px; display: inline-block;">r</div> </div> <div data-bbox="1406 1603 1485 1671" style="text-align: right;">  </div>
		<p>Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 0 (result is Not Equal), maximum CX times.</p> <p>Algorithm:</p> <p><code>check_cx:</code></p> <p>if <math>CX \neq 0</math> then</p> <ul style="list-style-type: none"> <li>• do following <u>chain instruction</u></li> </ul>

REPNE	chain instruction	<ul style="list-style-type: none"> <li>• <math>CX = CX - 1</math></li> <li>• if <math>ZF = 0</math> then:             <ul style="list-style-type: none"> <li>◦ go back to check_cx</li> </ul> </li> <li>else             <ul style="list-style-type: none"> <li>◦ exit from REPNE cycle</li> </ul> </li> </ul> <div data-bbox="691 488 730 589" style="border: 1px solid black; padding: 2px; display: inline-block;"> <div style="border: 1px solid black; padding: 2px; display: inline-block;">Z</div>  <div style="border: 1px solid black; padding: 2px; display: inline-block;">r</div> </div> <div data-bbox="1406 589 1485 658" style="text-align: right;">  </div>
REPNZ	chain instruction	<p>Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while <math>ZF = 0</math> (result is Not Zero), maximum CX times.</p> <p>Algorithm:</p> <p>check_cx:</p> <p>if <math>CX \neq 0</math> then</p> <ul style="list-style-type: none"> <li>• do following <u>chain instruction</u></li> <li>• <math>CX = CX - 1</math></li> <li>• if <math>ZF = 0</math> then:             <ul style="list-style-type: none"> <li>◦ go back to check_cx</li> </ul> </li> <li>else             <ul style="list-style-type: none"> <li>◦ exit from REPNZ cycle</li> </ul> </li> </ul> <p>else</p> <ul style="list-style-type: none"> <li>• exit from REPNZ cycle</li> </ul> <div data-bbox="691 1507 730 1608" style="border: 1px solid black; padding: 2px; display: inline-block;"> <div style="border: 1px solid black; padding: 2px; display: inline-block;">Z</div>  <div style="border: 1px solid black; padding: 2px; display: inline-block;">r</div> </div> <div data-bbox="1406 1608 1485 1677" style="text-align: right;">  </div>
		<p>Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while <math>ZF = 1</math> (result is Zero), maximum CX times.</p> <p>Algorithm:</p> <p>check_cx:</p> <p>if <math>CX \neq 0</math> then</p>

REPZ	chain instruction	<ul style="list-style-type: none"> <li>do following <u>chain instruction</u></li> <li><math>CX = CX - 1</math></li> <li>if <math>ZF = 1</math> then:             <ul style="list-style-type: none"> <li>go back to check_cx</li> </ul> </li> <li>else             <ul style="list-style-type: none"> <li>exit from REPZ cycle</li> </ul> </li> </ul> <p>else</p> <ul style="list-style-type: none"> <li>exit from REPZ cycle</li> </ul> <div data-bbox="691 528 729 629"> <div>Z</div> <div>r</div> </div> <div data-bbox="1406 629 1482 701"></div>
RET	No operands or even immediate	<p>Return from near procedure.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> <li>Pop from stack:             <ul style="list-style-type: none"> <li>IP</li> </ul> </li> <li>if <u>immediate</u> operand is present:             <ul style="list-style-type: none"> <li><math>SP = SP + \text{operand}</math></li> </ul> </li> </ul> <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 data-bbox="691 1592 880 1693"> <div>CZSOPA</div> <div>unchanged</div> </div> <div data-bbox="1406 1693 1482 1765"></div>
RETF	No operands or even immediate	<p>Return from Far procedure.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> <li>Pop from stack:             <ul style="list-style-type: none"> <li>IP</li> <li>CS</li> </ul> </li> <li>if <u>immediate</u> operand is present:</li> </ul>

$$SP = SP + \text{operand}$$


ROL

memory, immediate  
REG, immediate

memory, CL  
REG, CL

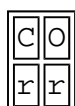
Rotate operand1 left. The number of rotates is set by operand2.

Algorithm:

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.



ROR

memory, immediate  
REG, immediate

memory, CL  
REG, CL

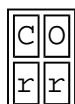
Rotate operand1 right. The number of rotates is set by operand2.

Algorithm:

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

Example:

```
MOV AL, 1Ch      ; AL = 00011100b
ROR AL, 1        ; AL = 00001110b, CF=0.
RET
```





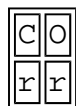
OF=0 if first operand keeps original sign.



Store AH register into low 8 bits of Flags register.

Algorithm:

SAHF	No operands	<div>flags register = AH</div> <div>AH bit:    7    6    5    4    3    2    1    0           [SF] [ZF] [0] [AF] [0] [PF] [1] [CF]</div> <div>bits 1, 3, 5 are reserved.</div> <div><table><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></div> <div></div>	C	Z	S	O	P	A	r	r	r	r	r	r
C	Z	S	O	P	A									
r	r	r	r	r	r									
SAL	<div>memory, immediate REG, immediate</div> <div>memory, CL REG, CL</div>	<div>Shift Arithmetic operand1 Left. The number of shifts is set by operand2.</div> <div>Algorithm:</div> <div><ul style="list-style-type: none"><li>Shift all bits left, the bit that goes off is set to CF.</li><li>Zero bit is inserted to the right-most position.</li></ul></div> <div>Example:</div> <div><div>MOV AL, 0E0h            ; AL = 11100000b</div><div>SAL AL, 1             ; AL = 11000000b,    CF=1.</div><div>RET</div></div> <div><table><tr><td>C</td><td>O</td></tr><tr><td>r</td><td>r</td></tr></table></div> <div>OF=0 if first operand keeps original sign.</div> <div></div>	C	O	r	r								
C	O													
r	r													
SAR	<div>memory, immediate REG, immediate</div> <div>memory, CL REG, CL</div>	<div>Shift Arithmetic operand1 Right. The number of shifts is set by operand2.</div> <div>Algorithm:</div> <div><ul style="list-style-type: none"><li>Shift all bits right, the bit that goes off is set to CF.</li><li>The sign bit that is inserted to the left-most position has the same value as before shift.</li></ul></div> <div>Example:</div> <div><div>MOV AL, 0E0h            ; AL = 11100000b</div><div>SAR AL, 1             ; AL = 11110000b,    CF=0.</div><div>MOV BL, 4Ch            ; BL = 01001100b</div><div>SAR BL, 1             ; BL = 00100110b,    CF=0.</div><div>RET</div></div>												



OF=0 if first operand keeps original sign.



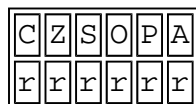
Subtract with Borrow.

Algorithm:

$\text{operand1} = \text{operand1} - \text{operand2} - \text{CF}$

Example:

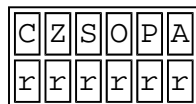
```
STC
MOV AL, 5
SBB AL, 3      ; AL = 5 - 3 - 1 = 1
RET
```



Compare bytes: AL from ES:[DI].

Algorithm:

- $\text{AL} - \text{ES}:[\text{DI}]$
- set flags according to result:  
OF, SF, ZF, AF, PF, CF
- if  $\text{DF} = 0$  then
  - $\text{DI} = \text{DI} + 1$
- else
  - $\text{DI} = \text{DI} - 1$



Compare words: AX from ES:[DI].

Algorithm:

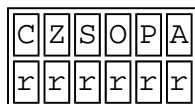
- $\text{AX} - \text{ES}:[\text{DI}]$
- set flags according to result:  
OF, SF, ZF, AF, PF, CF
- if  $\text{DF} = 0$  then
  - $\text{DI} = \text{DI} + 2$



```

else
    o DI = DI - 2

```



SHL

```

memory, immediate
REG, immediate

```

```

memory, CL
REG, CL

```

Shift operand1 Left. The number of shifts is set by operand2.

Algorithm:

- Shift all bits left, the bit that goes off is set to CF.
- Zero bit is inserted to the right-most position.

Example:

```

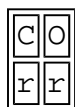
MOV AL, 11100000b
SHL AL, 1          ; AL = 11000000b, CF=1.

```

```

RET

```



OF=0 if first operand keeps original sign.



SHR

```

memory, immediate
REG, immediate

```

```

memory, CL
REG, CL

```

Shift operand1 Right. The number of shifts is set by operand2.

Algorithm:

- Shift all bits right, the bit that goes off is set to CF.
- Zero bit is inserted to the left-most position.

Example:

```

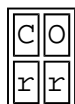
MOV AL, 00000111b
SHR AL, 1          ; AL = 00000011b, CF=1.

```

```




RET

```



OF=0 if first operand keeps original sign.



STC	No operands	<p>Set Carry flag.</p> <p>Algorithm:</p> $CF = 1$ <div> <div>C</div> <div>1</div> </div> 
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> $DF = 1$ <div> <div>D</div> <div>1</div> </div> 
STI	No operands	<p>Set Interrupt enable flag. This enables hardware interrupts.</p> <p>Algorithm:</p> $IF = 1$ <div> <div>I</div> <div>1</div> </div> 
STOSB	No operands	<p>Store byte in AL into ES:[DI]. Update DI.</p> <p>Algorithm:</p> <ul style="list-style-type: none"> <li>• <math>ES:[DI] = AL</math></li> <li>• if <math>DF = 0</math> then <ul style="list-style-type: none"> <li>◦ <math>DI = DI + 1</math></li> </ul> </li> <li>else <ul style="list-style-type: none"> <li>◦ <math>DI = DI - 1</math></li> </ul> </li> </ul> <p>Example:</p> <pre>ORG 100h LEA DI, a1</pre>

```
MOV AL, 12h
MOV CX, 5

REP STOSB

RET

a1 DB 5 dup(0)
```



STOSW

No operands

Store word in AX into ES:[DI]. Update DI.

Algorithm:

- $ES:[DI] = AX$
- if  $DF = 0$  then
  - $DI = DI + 2$
- else
  - $DI = DI - 2$

Example:

```
ORG 100h

LEA DI, a1
MOV AX, 1234h
MOV CX, 5

REP STOSW

RET

a1 DW 5 dup(0)
```



SUB

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

Subtract.




Algorithm:

$operand1 = operand1 - operand2$

Example:

```
MOV AL, 5
SUB AL, 1           ; AL = 4

RET
```

		<div> <div> <div>C</div><div>Z</div><div>S</div><div>O</div><div>P</div><div>A</div> <div>r</div><div>r</div><div>r</div><div>r</div><div>r</div><div>r</div> </div> <div>  </div> </div>
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: <b>ZF, SF, PF</b>. Result is not stored anywhere.</p> <p>These rules apply:</p> <pre> 1 AND 1 = 1 1 AND 0 = 0 0 AND 1 = 0 0 AND 0 = 0 </pre> <p>Example:</p> <pre> MOV AL, 00000101b TEST AL, 1           ; ZF = 0. TEST AL, 10b         ; ZF = 1. RET </pre> <div> <div> <div>C</div><div>Z</div><div>S</div><div>O</div><div>P</div> <div>0</div><div>r</div><div>r</div><div>0</div><div>r</div> </div> <div>  </div> </div>
XCHG	REG, memory memory, REG REG, REG	<p>Exchange values of two operands.</p> <p>Algorithm:</p> <pre>operand1 &lt; - &gt; operand2</pre> <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> <div> <div> <div>C</div><div>Z</div><div>S</div><div>O</div><div>P</div><div>A</div> <div>unchanged</div> </div> <div>  </div> </div>
		<p>Translate byte from table. Copy value of memory byte at DS:[BX + unsigned AL] to AL register.</p> <p>Algorithm:</p>

XLATB	No operands	<p>AL = DS:[BX + unsigned AL]</p> <p>Example:</p> <pre>ORG 100h LEA BX, dat MOV AL, 2 XLATB      ; AL = 33h  RET  dat DB 11h, 22h, 33h, 44h, 55h</pre> <table><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>	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> <pre>1 XOR 1 = 0 1 XOR 0 = 1 0 XOR 1 = 1 0 XOR 0 = 0</pre> <p>Example:</p> <pre>MOV AL, 00000111b XOR AL, 00000010b      ; AL = 00000101b RET</pre> <table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>0</td><td>r</td><td>r</td><td>0</td><td>r</td><td>?</td></tr></table> <div></div>	C	Z	S	O	P	A	0	r	r	0	r	?
C	Z	S	O	P	A									
0	r	r	0	r	?									