Complete 8086 instruction set

Quick reference:

	CMPSB				MOV		
AAA	CMPSW	JAE	JNBE	JPO	MOVSB	RCR	SCASB
AAD	CWD	JB	JNC	JS	MOVSW	REP	SCASW
AAM	DAA	JBE	JNE	JZ	MUL	REPE	SHL
AAS	DAS	JC	JNG	LAHF	NEG	REPNE	SHR
ADC	DEC	JCXZ	JNGE	LDS	NOP	REPNZ	STC
ADD	DIV	JE	JNL	LEA	NOT	REPZ	STD
AND	HLT	JG	JNLE	LES	OR	RET	STI
CALL	IDIV	JGE	JNO	LODSB	OUT	RETF	STOSB
CBW	IMUL	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.

<table>
<tr><td></td><td></td><td>
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

<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>
Back
</td></tr>
<tr><td>AAD</td><td>No operands</td><td>
ASCII Adjust before Division.
Prepares two BCD values for division.

Algorithm:
AL = (AH * 10) + AL
AH = 0
Example:

MOV AX, 0105h ; AH = 01, AL = 05
AAD ; AH = 00, AL = 0Fh (15)
RET

<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>
Back
</td></tr>
<tr><td>AAM</td><td>No operands</td><td>
ASCII Adjust after Multiplication.
Corrects the result of multiplication of two BCD values.

Algorithm:
</td></tr>
</table>

		• AH = AL / 10 • AL = remainder Example: MOV AL, 15 ; AL = 0Fh AAM ; AH = 01, AL = 05 RET C Z S O P A ? r r ? r ? Back

AAS	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
		
		Back

ADC	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Add with Carry. Algorithm: operand1 = operand1 + operand2 + CF Example:

<table>
<tr>
<td></td>
<td></td>
<td>STC ; set CF = 1
MOV AL, 5 ; AL = 5
ADC AL, 1 ; AL = 7
RET

<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>

Back</td>
</tr>
<tr>
<td>ADD</td>
<td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td>
<td>Add.

Algorithm:

operand1 = operand1 + operand2

Example:

MOV AL, 5 ; AL = 5
ADD AL, -3 ; AL = 2
RET

<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>

Back</td>
</tr>
<tr>
<td>AND</td>
<td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td>
<td>Logical AND between all bits of two operands. Result is stored in operand1.

These rules apply:

1 AND 1 = 1
1 AND 0 = 0
0 AND 1 = 0
0 AND 0 = 0

Example:

MOV AL, 'a' ; AL = 01100001b
AND AL, 11011111b ; AL = 01000001b ('A')
RET

<table><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>

Back</td>
</tr>
<tr>
<td>CALL</td>
<td>procedure name
label
4-byte address</td>
<td>Transfers control to procedure. Return address (IP) is pushed to stack. 4-byte address may be entered in this form: 1234h:5678h, first value is a segment second value is an offset. If it's a far call,</td>
</tr>
</table>

		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
		
		C Z S O P A
		unchanged
		
		Back

CBW	No operands	Convert byte into word.
		
		Algorithm:
		
		if high bit of AL = 1 then:
		
		• AH = 255 (0FFh)
		
		else
		
		• AH = 0
		
		Example:
		
		MOV AX, 0 ; AH = 0, AL = 0
		MOV AL, -5 ; AX = 000FBh (251)
		CBW ; AX = 0FFFBh (-5)
		RET
		
		C Z S O P A
		unchanged
		
		Back

CLC	No operands	Clear Carry flag. Algorithm: CF = 0

		C 0 Back
CLD	No operands	Clear Direction flag. SI and DI will be incremented by chain instructions: CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW. Algorithm: DF = 0 D 0 Back
CLI	No operands	Clear Interrupt enable flag. This disables hardware interrupts. Algorithm: IF = 0 I 0 Back
CMC	No operands	Complement Carry flag. Inverts value of CF. Algorithm: if CF = 1 then CF = 0 if CF = 0 then CF = 1 C r Back
CMP	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Compare. Algorithm: operand1 - operand2 result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.

<table>
<tr>
<td></td>
<td></td>
<td>
Example:

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

<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>
Back
</td>
</tr>
<tr>
<td>CMPSB</td>
<td>No operands</td>
<td>
Compare bytes: ES:[DI] from DS:[SI].

Algorithm:

DS:[SI] - ES:[DI]
set flags according to result:
OF, SF, ZF, AF, PF, CF
if DF = 0 then
SI = SI + 1
DI = DI + 1
else
SI = SI - 1
DI = DI - 1

Example:
open cmpsb.asm from c:\emu8086\examples

<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>
Back
</td>
</tr>
<tr>
<td>CMPSW</td>
<td>No operands</td>
<td>
Compare words: ES:[DI] from DS:[SI].

Algorithm:

DS:[SI] - ES:[DI]
set flags according to result:
OF, SF, ZF, AF, PF, CF
if DF = 0 then
SI = SI + 2
DI = DI + 2
else
SI = SI - 2
DI = DI - 2

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

<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr></table>
</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td>
<table><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table>
Back
</td></tr>
<tr><td>CWD</td><td>No operands</td><td>
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 = 00000h:0FFFBh
CWD ; DX AX = 0FFFFh:0FFFBh
RET
<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>
Back
</td></tr>
<tr><td>DAA</td><td>No operands</td><td>
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:

MOV AL, 0Fh ; AL = 0Fh (15)
DAA ; AL = 15h
RET
<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>
</td></tr>
</table>

<table>
<tr><td></td><td></td><td>Back</td></tr>
<tr><td>DAS</td><td>No operands</td><td>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
<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>Back</td></tr>
<tr><td>DEC</td><td>REG
memory</td><td>Decrement.

Algorithm:

operand = operand - 1

Example:

MOV AL, 255 ; AL = 0FFh (255 or -1)
DEC AL ; AL = 0FEh (254 or -2)
RET
<table><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>CF - unchanged!
Back</td></tr>
<tr><td>DIV</td><td>REG
memory</td><td>Unsigned divide.

Algorithm:</td></tr>
</table>

		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
		
		C Z S O P A
		? ? ? ? ? ?
		
		Back

HLT	No operands	Halt the System. Example: MOV AX, 5 HLT C Z S O P A unchanged Back

IDIV	REG
	memory
		Signed divide.
		
		Algorithm:
		
		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 = 0FF35h
		MOV BL, 4
		IDIV BL ; AL = -50 (0CEh), AH = -3 (0FDh)
		RET
		
		C Z S O P A
		? ? ? ? ? ?
		
		Back

<table>
<tr>
<td>IMUL</td>
<td>REG
memory</td>
<td>
Signed multiply.

Algorithm:

when operand is a byte:
AX = AL * operand.

when operand is a word:
(DX AX) = AX * operand.

Example:

MOV AL, -2
MOV BL, -4
IMUL BL ; AX = 8
RET
<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>
CF=OF=0 when result fits into operand of IMUL.
Back
</td>
</tr>
<tr>
<td>IN</td>
<td>AL, im.byte
AL, DX
AX, im.byte
AX, DX</td>
<td>
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.
Example:

IN AX, 4 ; get status of traffic lights.
IN AL, 7 ; get status of stepper-motor.
<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>
Back
</td>
</tr>
<tr>
<td>INC</td>
<td>REG
memory</td>
<td>
Increment.

Algorithm:

operand = operand + 1

Example:

MOV AL, 4
INC AL ; AL = 5
RET
<table>
<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>
CF - unchanged!
Back
</td>
</tr>
</table>

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
		
		C Z S O P A I
		unchanged 0
		
		Back

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
		
		Back

IRET	No operands	Interrupt Return. Algorithm: Pop from stack: ◦ IP ◦ CS ◦ flags register C Z S O P A popped

<table>
<tr><td></td><td></td><td>Back</td></tr>
<tr><td>JA</td><td>label</td><td>Short Jump if first operand is Above 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, 250
CMP AL, 5
JA label1
PRINT 'AL is not above 5'
JMP exit
label1:
PRINT 'AL is above 5'
exit:
RET

<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>
Back</td></tr>
<tr><td>JAE</td><td>label</td><td>Short Jump if first operand is Above or Equal to second operand (as set by CMP instruction). Unsigned.

Algorithm:

if CF = 0 then jump

Example:

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

<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></td></tr>
</table>

		Back

JB	label	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
		
		C Z S O P A
		unchanged
		
		Back

JBE	label	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
		
		C Z S O P A
		unchanged

		Back

JC	label	Short Jump if Carry flag is set to 1.
		
		Algorithm:
		
		if CF = 1 then jump
		
		Example:
		
		include 'emu8086.inc'
		ORG 100h
		MOV AL, 255
		ADD AL, 1
		JC label1
		PRINT 'no carry.'
		JMP exit
		label1:
		PRINT 'has carry.'
		exit:
		RET
		
		C Z S O P A
		unchanged
		
		Back

JCXZ	label	Short Jump if CX register is 0. Algorithm: if CX = 0 then jump Example: include 'emu8086.inc' ORG 100h MOV CX, 0 JCXZ label1 PRINT 'CX is not zero.' JMP exit label1: PRINT 'CX is zero.' exit: RET C Z S O P A unchanged Back
JE	label	Short Jump if first operand is Equal to second operand (as set by CMP instruction).

<table>
<tr>
<td></td>
<td></td>
<td>Signed/Unsigned.

Algorithm:

if ZF = 1 then jump

Example:
<pre>
   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
</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>
Back</td>
</tr>
<tr>
<td>JG</td>
<td>label</td>
<td>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:
<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>
<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>
Back</td>
</tr>
<tr>
<td>JGE</td>
<td>label</td>
<td>Short Jump if first operand is Greater or Equal to second operand (as set by CMP instruction). Signed.</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td>
Algorithm:

if SF = OF then jump

Example:
<pre>
  include 'emu8086.inc'
  ORG 100h
  MOV AL, 2
  CMP AL, -5
  JGE label1
  PRINT 'AL < -5'
  JMP exit
label1:
  PRINT 'AL >= -5'
exit:
  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>
Back
</td>
</tr>
<tr>
<td>JL</td>
<td>label</td>
<td>
Short Jump if first operand is Less then second operand (as set by CMP instruction). Signed.

Algorithm:

if SF <> OF then jump

Example:
<pre>
  include 'emu8086.inc'
  ORG 100h
  MOV AL, -2
  CMP AL, 5
  JL label1
  PRINT 'AL >= 5.'
  JMP exit
label1:
  PRINT 'AL < 5.'
exit:
  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>
Back
</td>
</tr>
<tr>
<td>JLE</td>
<td>label</td>
<td>
Short Jump if first operand is Less or Equal to second operand (as set by CMP instruction). Signed.
</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td>
Algorithm:

if SF <> OF or ZF = 1 then jump

Example:
<pre>
   include 'emu8086.inc'
   ORG 100h
   MOV AL, -2
   CMP AL, 5
   JLE label1
   PRINT 'AL > 5.'
   JMP exit
label1:
   PRINT 'AL <= 5.'
exit:
   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>
Back
</td>
</tr>
<tr>
<td>JMP</td>
<td>label
4-byte address</td>
<td>
Unconditional Jump. Transfers control to another part of the program. 4-byte address may be entered in this form: 1234h:5678h, first value is a segment second value is an offset.

Algorithm:

always jump

Example:
<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>
<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>
Back
</td>
</tr>
<tr>
<td>JNA</td>
<td>label</td>
<td>Short Jump if first operand is Not Above second operand (as set by CMP instruction). Unsigned.</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td>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

<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>
Back</td></tr>
<tr><td>JNAE</td><td>label</td><td>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

<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>
Back</td></tr>
<tr><td>JNB</td><td>label</td><td>Short Jump if first operand is Not Below second operand (as set by CMP instruction). Unsigned.</td></tr>
</table>

		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
		
		C Z S O P A
		unchanged
		
		Back

JNBE	label	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
		
		C Z S O P A
		unchanged
		
		Back

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`
		
		C Z S O P A
		unchanged
		
		Back

JNE	label	Short Jump if first operand is Not Equal to second operand (as set by CMP instruction). Signed/Unsigned.
		
		Algorithm:
		
		if ZF = 0 then jump
		
		Example:
		`include 'emu8086.inc'`
		
		`ORG 100h`
		`MOV AL, 2`
		`CMP AL, 3`
		`JNE  label1`
		`PRINT 'AL = 3.'`
		`JMP exit`
		`label1:`
		`PRINT 'Al <> 3.'`
		`exit:`
		`RET`
		
		C Z S O P A
		unchanged
		
		Back

JNG	label	Short Jump if first operand is Not Greater then

		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
		
		C Z S O P A
		unchanged
		
		Back

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
		
		C Z S O P A
		unchanged
		
		Back

JNL	label	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
		
		Back

JNLE	label	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

		Back

JNO	label	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
		unchanged
		
		Back

JNP	label	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

<table>
<tr><td></td><td></td><td>C Z S O P A
unchanged

Back</td></tr>
<tr><td>JNS</td><td>label</td><td>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

Back</td></tr>
<tr><td>JNZ</td><td>label</td><td>Short Jump if Not Zero (not equal). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.

Algorithm:

if ZF = 0 then jump

Example:

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</td></tr>
</table>

		C Z S O P A unchanged Back

JO	label	Short Jump if Overflow.
		
		Algorithm:
		
		if OF = 1 then jump
		
		Example:
		
		; -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
		
		C Z S O P A
		unchanged
		Back

JP	label	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.
		  JP label1
		  PRINT 'parity odd.'
		  JMP exit
		label1:

<table>
<tr>
<td></td>
<td></td>
<td>
  PRINT 'parity even.'
exit:
  RET

<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>

Back
</td>
</tr>
<tr>
<td>JPE</td>
<td>label</td>
<td>
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

<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>

Back
</td>
</tr>
<tr>
<td>JPO</td>
<td>label</td>
<td>
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.'
</td>
</tr>
</table>

		JMP exit label1: PRINT 'parity odd.' exit: RET C Z S O P A unchanged Back
JS	label	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 Back

JZ	label	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 C Z S O P A unchanged Back
LAHF	No operands	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. C Z S O P A unchanged Back
LDS	REG, memory	Load memory double word into word register and DS. Algorithm: • REG = first word • DS = second word Example: ORG 100h LDS AX, m RET m DW 1234h DW 5678h END

		AX is set to 1234h, DS is set to 5678h. C Z S O P A unchanged Back

LEA	REG, memory	Load Effective Address.
		
		Algorithm:
		
		• REG = address of memory (offset)
		
		Example:
		
		MOV BX, 35h
		MOV DI, 12h
		LEA SI, [BX+DI] ; SI = 35h + 12h = 47h
		
		Note: The integrated 8086 assembler automatically replaces **LEA** with a more efficient **MOV** where possible. For example:
		
		org 100h
		LEA AX, m ; AX = offset of m
		RET
		m dw 1234h
		END
		
		C Z S O P A
		unchanged
		
		Back

LES	REG, memory	Load memory double word into word register and ES. Algorithm: • REG = first word • ES = second word Example: ORG 100h

		LES AX, m RET m DW 1234h DW 5678h END AX is set to 1234h, ES is set to 5678h. C Z S O P A unchanged Back

LODSB	No operands	Load byte at DS:[SI] into AL. Update SI.
		
		Algorithm:
		
		• AL = DS:[SI]
		• if DF = 0 then
		    ◦ SI = SI + 1
		  else
		    ◦ SI = SI - 1
		
		Example:
		
		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'
		
		C Z S O P A
		unchanged
		
		Back

LODSW	No operands	Load word at DS:[SI] into AX. Update SI. Algorithm:

<table>
<tr>
<td></td>
<td></td>
<td>

- 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
```

<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>

Back

</td>
</tr>
<tr>
<td>LOOP</td>
<td>label</td>
<td>

Decrease CX, jump to label if CX not zero.

Algorithm:

- CX = CX - 1
- if CX <> 0 then
    - jump

  else
    - no jump, continue

Example:

```
   include 'emu8086.inc'

   ORG 100h
   MOV CX, 5
label1:
   PRINTN 'loop!'
   LOOP label1
   RET
```

<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>

Back

</td>
</tr>
<tr>
<td>LOOPE</td>
<td>label</td>
<td>

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

</td>
</tr>
</table>

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
```

C	Z	S	O	P	A
unchanged					

Back

LOOPNE	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
   MOV SI, 0
   MOV CX, 5
label1:
   PUTC '*'
```

<table>
<tr><td></td><td></td><td>
  MOV AL, v1[SI]
  INC SI   ; next byte (SI=SI+1).
  CMP AL, 7
  LOOPNE label1
  RET
  v1 db 9, 8, 7, 6, 5
<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>
Back
</td></tr>
<tr><td>LOOPNZ</td><td>label</td><td>
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
<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>
Back
</td></tr>
<tr><td>LOOPZ</td><td>label</td><td>
Decrease CX, jump to label if CX not zero and ZF = 1.

Algorithm:
CX = CX - 1
if (CX <> 0) and (ZF = 1) then
</td></tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td>

- 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
```

<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>

Back

</td>
</tr>
<tr>
<td>MOV</td>
<td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate

SREG, memory
memory, SREG
REG, SREG
SREG, REG</td>
<td>

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, 01011111b ; CL = color attribute.
MOV BX, 15Eh      ; BX = position on screen.
MOV [BX], CX      ; w.[0B800h:015Eh] = CX.
RET               ; returns to operating system.
```

</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td>
<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>
Back
</td></tr>
<tr><td>MOVSB</td><td>No operands</td><td>
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:
<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>
<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>
Back
</td></tr>
<tr><td>MOVSW</td><td>No operands</td><td>
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:
</td></tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td>
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>
<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>
Back
</td>
</tr>
<tr>
<td>MUL</td>
<td>REG
memory</td>
<td>
Unsigned multiply.

Algorithm:

when operand is a byte:
AX = AL * operand.

when operand is a word:
(DX AX) = AX * operand.

Example:

MOV AL, 200 ; AL = 0C8h
MOV BL, 4
MUL BL ; AX = 0320h (800)
RET
<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>
CF=OF=0 when high section of the result is zero.
Back
</td>
</tr>
<tr>
<td>NEG</td>
<td>REG
memory</td>
<td>
Negate. Makes operand negative (two's complement).

Algorithm:

Invert all bits of the operand
Add 1 to inverted operand

Example:

MOV AL, 5 ; AL = 05h
NEG AL ; AL = 0FBh (-5)
</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td>NEG AL ; AL = 05h (5)
RET
<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>
Back</td></tr>
<tr><td>NOP</td><td>No operands</td><td>No Operation.

Algorithm:
Do nothing
Example:

; do nothing, 3 times:
NOP
NOP
NOP
RET
<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>
Back</td></tr>
<tr><td>NOT</td><td>REG
memory</td><td>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
<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>
Back</td></tr>
<tr><td>OR</td><td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td><td>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</td></tr>
</table>

		0 OR 0 = 0 Example: MOV AL, 'A' ; AL = 01000001b OR AL, 00100000b ; AL = 01100001b ('a') RET C Z S O P A 0 r r 0 r ? Back

OUT	im.byte, AL
	im.byte, AX
	DX, AL
	DX, AX
		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.
		
		Example:
		
		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.
		
		C Z S O P A
		unchanged
		
		Back

POP	REG SREG memory	Get 16 bit value from the stack. Algorithm: • operand = SS:[SP] (top of the stack) • SP = SP + 2 Example: MOV AX, 1234h PUSH AX POP DX ; DX = 1234h RET C Z S O P A unchanged Back
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 C Z S O P A unchanged Back
POPF	No operands	Get flags register from the stack. Algorithm: • flags = SS:[SP] (top of the stack) • SP = SP + 2 C Z S O P A popped Back
PUSH	REG SREG memory immediate	Store 16 bit value in the stack. Note: **PUSH immediate** works only on 80186 CPU and later! Algorithm: • SP = SP - 2 • SS:[SP] (top of the stack) = operand Example: MOV AX, 1234h PUSH AX POP DX ; DX = 1234h RET

		C | Z | S | O | P | A unchanged Back
PUSHA	No operands	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. Note: this instruction works only on **80186** CPU and later! Algorithm: • PUSH AX • PUSH CX • PUSH DX • PUSH BX • PUSH SP • PUSH BP • PUSH SI • PUSH DI C | Z | S | O | P | A unchanged Back
PUSHF	No operands	Store flags register in the stack. Algorithm: • SP = SP - 2 • SS:[SP] (top of the stack) = flags C | Z | S | O | P | A unchanged Back
RCL	memory, immediate REG, immediate memory, CL REG, CL	Rotate operand1 left through Carry Flag. The number of rotates is set by operand2. When **immediate** is greater then 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). Algorithm:

		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 C O r r OF=0 if first operand keeps original sign. Back
RCR	memory, immediate REG, immediate memory, CL REG, CL	Rotate operand1 right through Carry Flag. The number of rotates is set by operand2. Algorithm: 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. Example: STC ; set carry (CF=1). MOV AL, 1Ch ; AL = 00011100b RCR AL, 1 ; AL = 10001110b, CF=0. RET C O r r OF=0 if first operand keeps original sign. Back

REP	chain instruction	Repeat following MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW instructions CX times.
		
		Algorithm:
		
		check_cx:
		
		if CX <> 0 then
		
		• do following chain instruction
		• CX = CX - 1
		• go back to check_cx
		
		else
		
		• exit from REP cycle

		Z r Back

REPE	chain instruction	Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 1 (result is Equal), maximum CX times.
		
		Algorithm:
		
		check_cx:
		
		if CX <> 0 then
		
		• do following chain instruction
		• CX = CX - 1
		• if ZF = 1 then:
		  ◦ go back to check_cx
		  else
		  ◦ exit from REPE cycle
		
		else
		
		• exit from REPE cycle
		
		example:
		open **cmpsb.asm** from c:\emu8086\examples
		
		Z
		r
		Back

REPNE	chain instruction	Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 0 (result is Not Equal), maximum CX times.
		
		Algorithm:
		
		check_cx:
		
		if CX <> 0 then
		
		• do following chain instruction
		• CX = CX - 1
		• if ZF = 0 then:
		  ◦ go back to check_cx
		  else
		  ◦ exit from REPNE cycle
		
		else
		
		• exit from REPNE cycle

		Z r Back

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

REPZ	chain instruction	Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 1 (result is Zero), maximum CX times.
		
		Algorithm:
		
		check_cx:
		
		if CX <> 0 then
		
		• do following chain instruction
		• CX = CX - 1
		• if ZF = 1 then:
		    ◦ go back to check_cx
		  else
		    ◦ exit from REPZ cycle
		
		else
		
		• exit from REPZ cycle
		
		Z
		r

		Back

RET	No operands or even immediate	Return from near procedure.
		
		Algorithm:
		
		• Pop from stack:
		    ○ IP
		• if immediate operand is present: SP = SP + operand
		
		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
		
		C Z S O P A
		unchanged
		
		Back

RETF	No operands or even immediate	Return from Far procedure.
		
		Algorithm:
		
		• Pop from stack:
		    ○ IP
		    ○ CS
		• if immediate operand is present: SP = SP + operand
		
		C Z S O P A
		unchanged
		
		Back

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 C O r r OF=0 if first operand keeps original sign. Back
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 C O r r OF=0 if first operand keeps original sign. Back
SAHF	No operands	Store AH register into low 8 bits of Flags register. Algorithm: flags register = AH AH bit: 7 6 5 4 3 2 1 0 [SF] [ZF] [0] [AF] [0] [PF] [1] [CF] bits 1, 3, 5 are reserved. C Z S O P A r r r r r r Back
SAL	memory, immediate REG, immediate	Shift Arithmetic operand1 Left. The number of shifts is set by operand2.

	memory, CL REG, CL	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, 0E0h ; AL = 11100000b SAL AL, 1 ; AL = 11000000b, CF=1. RET C O r r OF=0 if first operand keeps original sign. Back

SAR	memory, immediate
	REG, immediate
	
	memory, CL
	REG, CL
		Shift Arithmetic operand1 Right. The number of shifts is set by operand2.
		
		Algorithm:
		
		• 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.
		
		Example:
		
		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
		
		C O
		r r
		
		OF=0 if first operand keeps original sign.
		
		Back

SBB	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Subtract with Borrow. Algorithm: operand1 = operand1 - operand2 - CF Example: STC MOV AL, 5 SBB AL, 3 ; AL = 5 - 3 - 1 = 1 RET

		C Z S O P A r r r r r r Back
SCASB	No operands	Compare bytes: AL from ES:[DI]. Algorithm: • AL - ES:[DI] • set flags according to result: OF, SF, ZF, AF, PF, CF • if DF = 0 then ◦ DI = DI + 1 else ◦ DI = DI - 1 C Z S O P A r r r r r r Back
SCASW	No operands	Compare words: AX from ES:[DI]. Algorithm: • AX - ES:[DI] • set flags according to result: OF, SF, ZF, AF, PF, CF • if DF = 0 then ◦ DI = DI + 2 else ◦ DI = DI - 2 C Z S O P A r r r r r r Back
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 C O r r OF=0 if first operand keeps original sign. Back
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 C O r r OF=0 if first operand keeps original sign. Back
STC	No operands	Set Carry flag. Algorithm: CF = 1 C 1 Back
STD	No operands	Set Direction flag. SI and DI will be decremented by chain instructions: CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW. Algorithm: DF = 1 D 1 Back

<table>
<tr>
<td>STI</td>
<td>No operands</td>
<td>Set Interrupt enable flag. This enables hardware interrupts.

Algorithm:

IF = 1

<table><tr><td>I</td></tr><tr><td>1</td></tr></table>
Back</td>
</tr>
<tr>
<td>STOSB</td>
<td>No operands</td>
<td>Store byte in AL into ES:[DI]. Update DI.

Algorithm:
ES:[DI] = AL
if DF = 0 then
DI = DI + 1
else
DI = DI - 1
Example:

ORG 100h

LEA DI, a1
MOV AL, 12h
MOV CX, 5

REP STOSB

RET

a1 DB 5 dup(0)

<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>
Back</td>
</tr>
<tr>
<td>STOSW</td>
<td>No operands</td>
<td>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:</td>
</tr>
</table>

		ORG 100h
		
		LEA DI, a1
		MOV AX, 1234h
		MOV CX, 5
		
		REP STOSW
		
		RET
		
		a1 DW 5 dup(0)
		
		C Z S O P A
		unchanged
		
		Back

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

TEST	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Logical AND between all bits of two operands for flags only. These flags are effected: **ZF, SF, PF.**Result is not stored anywhere. These rules apply: 1 AND 1 = 1 1 AND 0 = 0 0 AND 1 = 0 0 AND 0 = 0 Example: MOV AL, 00000101b TEST AL, 1 ; ZF = 0. TEST AL, 10b ; ZF = 1. RET

		C Z S O P 0 r r 0 r Back

XCHG	REG, memory
	memory, REG
	REG, REG
		Exchange values of two operands.
		
		Algorithm:
		
		operand1 < - > operand2
		
		Example:
		
		MOV AL, 5
		MOV AH, 2
		XCHG AL, AH ; AL = 2, AH = 5
		XCHG AL, AH ; AL = 5, AH = 2
		RET
		
		C Z S O P A
		unchanged
		
		Back

XLATB	No operands	Translate byte from table. Copy value of memory byte at DS:[BX + unsigned AL] to AL register. Algorithm: AL = DS:[BX + unsigned AL] Example: ORG 100h LEA BX, dat MOV AL, 2 XLATB ; AL = 33h RET dat DB 11h, 22h, 33h, 44h, 55h C Z S O P A unchanged Back
XOR	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Logical XOR (Exclusive OR) between all bits of two operands. Result is stored in first operand. These rules apply:

		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 <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>