Complete 8086 instruction set

Quick reference:

<table>
<thead>
<tr>
<th>8086 Instruction</th>
<th>8086 Instruction</th>
<th>8086 Instruction</th>
<th>8086 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
<td>AAA</td>
</tr>
<tr>
<td>AAD</td>
<td>AAD</td>
<td>AAD</td>
<td>AAD</td>
</tr>
<tr>
<td>AAM</td>
<td>AAM</td>
<td>AAM</td>
<td>AAM</td>
</tr>
<tr>
<td>AAS</td>
<td>AAS</td>
<td>AAS</td>
<td>AAS</td>
</tr>
<tr>
<td>ADC</td>
<td>ADC</td>
<td>ADC</td>
<td>ADC</td>
</tr>
<tr>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
</tr>
<tr>
<td>CALL</td>
<td>CALL</td>
<td>CALL</td>
<td>CALL</td>
</tr>
<tr>
<td>CBW</td>
<td>CBW</td>
<td>CBW</td>
<td>CBW</td>
</tr>
<tr>
<td>CLC</td>
<td>CLC</td>
<td>CLC</td>
<td>CLC</td>
</tr>
<tr>
<td>CLI</td>
<td>CLI</td>
<td>CLI</td>
<td>CLI</td>
</tr>
<tr>
<td>CMC</td>
<td>CMC</td>
<td>CMC</td>
<td>CMC</td>
</tr>
<tr>
<td>CMP</td>
<td>CMP</td>
<td>CMP</td>
<td>CMP</td>
</tr>
<tr>
<td>CMPSB</td>
<td>CMPSW</td>
<td>CMPSB</td>
<td>CMPSW</td>
</tr>
<tr>
<td>JAE</td>
<td>JNF</td>
<td>JNC</td>
<td>JNS</td>
</tr>
<tr>
<td>JBE</td>
<td>JB</td>
<td>JBE</td>
<td>JB</td>
</tr>
<tr>
<td>JE</td>
<td>JG</td>
<td>JE</td>
<td>JE</td>
</tr>
<tr>
<td>JEB</td>
<td>JC</td>
<td>JEB</td>
<td>JC</td>
</tr>
<tr>
<td>JEC</td>
<td>JEC</td>
<td>JEC</td>
<td>JEC</td>
</tr>
<tr>
<td>JED</td>
<td>JED</td>
<td>JED</td>
<td>JED</td>
</tr>
<tr>
<td>JEE</td>
<td>JEE</td>
<td>JEE</td>
<td>JEE</td>
</tr>
<tr>
<td>JEL</td>
<td>JEL</td>
<td>JEL</td>
<td>JEL</td>
</tr>
<tr>
<td>JEN</td>
<td>JEN</td>
<td>JEN</td>
<td>JEN</td>
</tr>
<tr>
<td>JEO</td>
<td>JEO</td>
<td>JEO</td>
<td>JEO</td>
</tr>
<tr>
<td>JEP</td>
<td>JEP</td>
<td>JEP</td>
<td>JEP</td>
</tr>
<tr>
<td>LADE</td>
<td>LEA</td>
<td>LADE</td>
<td>LEA</td>
</tr>
<tr>
<td>LDS</td>
<td>LES</td>
<td>LDS</td>
<td>LES</td>
</tr>
<tr>
<td>LDSB</td>
<td>LDS</td>
<td>LDS</td>
<td>LDS</td>
</tr>
<tr>
<td>LODB</td>
<td>LDB</td>
<td>LODB</td>
<td>LDB</td>
</tr>
<tr>
<td>LODB</td>
<td>LOD</td>
<td>LODB</td>
<td>LOD</td>
</tr>
<tr>
<td>LODW</td>
<td>LOD</td>
<td>LODW</td>
<td>LOD</td>
</tr>
<tr>
<td>LOOP</td>
<td>LOOP</td>
<td>LOOP</td>
<td>LOOP</td>
</tr>
<tr>
<td>LOOPN</td>
<td>LOOPN</td>
<td>LOOPN</td>
<td>LOOPN</td>
</tr>
<tr>
<td>LOOPZ</td>
<td>LOOPZ</td>
<td>LOOPZ</td>
<td>LOOPZ</td>
</tr>
<tr>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>MOVSB</td>
<td>MOVW</td>
<td>MOVSB</td>
<td>MOVW</td>
</tr>
<tr>
<td>MUL</td>
<td>MUL</td>
<td>MUL</td>
<td>MUL</td>
</tr>
<tr>
<td>NEG</td>
<td>NEG</td>
<td>NEG</td>
<td>NEG</td>
</tr>
<tr>
<td>NOP</td>
<td>NOP</td>
<td>NOP</td>
<td>NOP</td>
</tr>
<tr>
<td>OR</td>
<td>OR</td>
<td>OR</td>
<td>OR</td>
</tr>
<tr>
<td>OUT</td>
<td>OUT</td>
<td>OUT</td>
<td>OUT</td>
</tr>
<tr>
<td>POP</td>
<td>POP</td>
<td>POP</td>
<td>POP</td>
</tr>
<tr>
<td>POPA</td>
<td>POPA</td>
<td>POPA</td>
<td>POPA</td>
</tr>
<tr>
<td>POPF</td>
<td>POPF</td>
<td>POPF</td>
<td>POPF</td>
</tr>
<tr>
<td>PUSH</td>
<td>PUSH</td>
<td>PUSH</td>
<td>PUSH</td>
</tr>
<tr>
<td>PUSH</td>
<td>PUSH</td>
<td>PUSH</td>
<td>PUSH</td>
</tr>
<tr>
<td>PUSH</td>
<td>PUSH</td>
<td>PUSH</td>
<td>PUSH</td>
</tr>
<tr>
<td>SAL</td>
<td>SAL</td>
<td>SAL</td>
<td>SAL</td>
</tr>
<tr>
<td>SAHF</td>
<td>SAHF</td>
<td>SAHF</td>
<td>SAHF</td>
</tr>
<tr>
<td>STOSB</td>
<td>STOSB</td>
<td>STOSB</td>
<td>STOSB</td>
</tr>
<tr>
<td>STOSW</td>
<td>STOSW</td>
<td>STOSW</td>
<td>STOSW</td>
</tr>
<tr>
<td>SUB</td>
<td>SUB</td>
<td>SUB</td>
<td>SUB</td>
</tr>
<tr>
<td>STI</td>
<td>STI</td>
<td>STI</td>
<td>STI</td>
</tr>
<tr>
<td>STOS</td>
<td>STOS</td>
<td>STOS</td>
<td>STOS</td>
</tr>
<tr>
<td>XCHG</td>
<td>XCHG</td>
<td>XCHG</td>
<td>XCHG</td>
</tr>
<tr>
<td>XLATB</td>
<td>XLATB</td>
<td>XLATB</td>
<td>XLATB</td>
</tr>
<tr>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
</tr>
</tbody>
</table>

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
• 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:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII Adjust after Addition. Corrects result in AH and AL after addition when working with BCD values. It works according to the following Algorithm: if low nibble of AL &gt; 9 or AF = 1 then:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
AAA

No operands

- 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

AAD

No operands

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

ASCII Adjust after Multiplication.
Corrects the result of multiplication of two BCD values.

Algorithm:

- AH = AL / 10
- AL = remainder
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| AAM         | No operands | MOV AL, 15 ; AL = 0Fh
AAM          ; AH = 01, AL = 05
RET          |             |        |
| AAS         | 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          |        |
| ADC         | Add with Carry. |
| REG, memory | Algorithm: |
| memory, REG | operand1 = operand1 + operand2 + CF |
| REG, REG    | Example:  |
memory, immediate
REG, immediate

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

Add.

Algorithm:
operand1 = operand1 + operand2

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

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

Transfers control to procedure, return address is (IP) is pushed to stack. 4-byte address may be entered in this form: 1234h:5678h, first value is a
CALL

procedure name
label
4-byte address

segment second value is an offset (this is a far
call, so CS is also pushed to stack).

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

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

Clear Carry flag.

Algorithm:

CF = 0
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operands</th>
<th>Description</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLC</td>
<td>No operands</td>
<td>Clear Direction flag. SI and DI will be incremented by chain instructions: CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW.</td>
<td></td>
</tr>
<tr>
<td>CLD</td>
<td>No operands</td>
<td>Clear Direction flag. SI and DI will be incremented by chain instructions: CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW.</td>
<td></td>
</tr>
<tr>
<td>CLI</td>
<td>No operands</td>
<td>Clear Interrupt enable flag. This disables hardware interrupts.</td>
<td></td>
</tr>
<tr>
<td>CMC</td>
<td>No operands</td>
<td>Complement Carry flag. Inverts value of CF.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Algorithm:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>if CF = 1 then CF = 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>if CF = 0 then CF = 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compare.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Algorithm:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>operand1 - operand2</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>Description</td>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>CMP</td>
<td>REG, memory memory, REG REG, REG memory, immediate REG, immediate</td>
<td>result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.</td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

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

<table>
<thead>
<tr>
<th>CMPSB</th>
<th>No operands</th>
<th>Compare bytes: ES:[DI] from DS:[SI].</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm:</td>
<td>DS:[SI] - ES:[DI]</td>
<td>set flags according to result: OF, SF, ZF, AF, PF, CF</td>
</tr>
<tr>
<td>if DF = 0 then</td>
<td>SI = SI + 1</td>
<td>DI = DI + 1</td>
</tr>
<tr>
<td>else</td>
<td>SI = SI - 1</td>
<td>DI = DI - 1</td>
</tr>
<tr>
<td>Example:</td>
<td>open <code>cmpsb.asm</code> from c:\emu8086\examples</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMPSW</th>
<th>No operands</th>
<th>Compare words: ES:[DI] from DS:[SI].</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm:</td>
<td>DS:[SI] - ES:[DI]</td>
<td>set flags according to result: OF, SF, ZF, AF, PF, CF</td>
</tr>
<tr>
<td>if DF = 0 then</td>
<td>SI = SI + 2</td>
<td>DI = DI + 2</td>
</tr>
<tr>
<td>else</td>
<td>SI = SI - 2</td>
<td>DI = DI - 2</td>
</tr>
</tbody>
</table>
example:
open cmpsw.asm from c:\emu8086\examples

CWD

No operands

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 ; AX AX = 0000h:0FFFBh
CWD ; DX AX = 0FFFFh:0FFFBh
RET

DAA

No operands

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

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
DIV

REG memory

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

C Z S O P A

1111111

HLT

No operands

Halt the System.

Example:

MOV AX, 5
HLT

C Z S O P A
unchanged

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
Signed multiply.

Algorithm:

- when operand is a byte:
  \[ AX = AL \times \text{operand} \]

- when operand is a word:
  \[ (DX \ AX) = AX \times \text{operand} \]

Example:

```
MOV AL, -2
MOV BL, -4
IMUL BL      ; AX = 8
RET
```

CF=OF=0 when result fits into operand of IMUL.

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.
```

Increment.

Algorithm:

\[ \text{operand} = \text{operand} + 1 \]

Example:

```
MOV AL, 4
INC AL       ; AL = 5
```
Interrupt number 4 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

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
<table>
<thead>
<tr>
<th>Instruction</th>
<th>No operands</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRET</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pop from stack:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o IP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o flags register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Algorithm:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if (CF = 0) and (ZF = 0) then jump</td>
</tr>
<tr>
<td></td>
<td></td>
<td></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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Algorithm:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if CF = 0 then jump</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

```assembly
c#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
```

```assembly
c#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:
```
### JB

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

**Algorithm:**

```plaintext
if CF = 1 then jump
```

**Example:**

```plaintext
#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
```

### JBE

**Description:**
Short Jump if first operand is Below or Equal to second operand (as set by CMP instruction). Unsigned.

**Algorithm:**

```plaintext
if CF = 1 or ZF = 1 then jump
```

**Example:**

```plaintext
#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
```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>JC</td>
<td>label</td>
<td>Short Jump if Carry flag is set to 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Algorithm:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if CF = 1 then jump</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>include 'emu8086.inc'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ORG 100h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MOV AL, 255</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ADD AL, 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JC label</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PRINT 'no carry.'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JMP exit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>label1:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PRINT 'has carry.'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exit:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RET</td>
</tr>
<tr>
<td>J CXZ</td>
<td>label</td>
<td>Short Jump if CX register is 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Algorithm:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if CX = 0 then jump</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>include 'emu8086.inc'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ORG 100h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MOV CX, 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J CXZ label1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PRINT 'CX is not zero.'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JMP exit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>label1:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PRINT 'CX is zero.'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exit:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RET</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Short Jump if first operand is Equal to second operand (as set by CMP instruction).</td>
</tr>
</tbody>
</table>
Signed/Unsigned.

Algorithm:

\[
\text{if } ZF = 1 \text{ then jump}
\]

Example:

```plaintext
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
```

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

Algorithm:

\[
\text{if } (ZF = 0) \text{ and } (SF = OF) \text{ then jump}
\]

Example:

```plaintext
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
```

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

Algorithm:
JGE

if SF = OF then jump

Example:

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

CZSOPA
unchanged

JL

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

Algorithm:

    if SF <> OF then jump

Example:

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

CZSOPA
unchanged

JL

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

Algorithm:

    if SF <> OF or ZF = 1 then jump

Example:
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JLE</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.</td>
</tr>
<tr>
<td>JMP</td>
<td>Short Jump if first operand is Not Above second operand (as set by CMP instruction). Unsigned.</td>
</tr>
<tr>
<td>JNA</td>
<td></td>
</tr>
</tbody>
</table>

### Algorithm:

#### Unconditional Jump:

- always jump

#### Example:

```assembly
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
```

### Algorithm:

#### Short Jump if first operand is Not Above second operand:

- if CF = 1 or ZF = 1 then jump

#### Example:

```assembly
include 'emu8086.inc'
ORG 100h
MOV AL, 2
```
CMP AL, 5
JNA label1
PRINT 'AL is above 5.'
JMP exit
label1:
PRINT 'AL is not above 5.'
exit:
RET

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

Algorithm:

    if CF = 1 then jump

Example:

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

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

Algorithm:

    if CF = 0 then jump

Example:

include 'emu8086.inc'
ORG 100h
MOV AL, 7
CMP AL, 5
JNB label1
PRINT 'AL < 5.'
JMP exit
label1:
    PRINT 'AL >= 5.'
exit:
    RET

C Z S O P
unchanged

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
unchanged

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:
Short Jump if first operand is Not Equal to second operand (as set by CMP instruction). Signed/Unsigned.

Algorithm:

\[
\text{if } ZF = 0 \text{ then jump}
\]

Example:

```assembly
include 'emu8086.inc'
ORG 100h
MOV AL, 2
CMP AL, 3
JNE label1
PRINT 'AL = 3.'
JMP exit
label1:
PRINT 'AL <> 3.'
exit:
RET
```

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

Algorithm:

\[
\text{if } (ZF = 1) \text{ and } (SF <> OF) \text{ then jump}
\]

Example:

```assembly
include 'emu8086.inc'
ORG 100h
MOV AL, 2
CMP AL, 3
JNG label1
PRINT 'AL > 3.'
JMP exit
label1:
PRINT 'AL <= 3.'
exit:
RET
```
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
```

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

Algorithm:

\[
\text{if (SF = OF) and (ZF = 0) then jump}
\]

Example:

```c
#include 'emu8086.inc'

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

Short Jump if Not Overflow.

Algorithm:

\[
\text{if OF = 0 then jump}
\]

Example:

```c
; -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
```
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:

\[
\text{if PF} = 0 \text{ then jump}
\]

Example:

```assembly
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
```

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

Algorithm:

\[
\text{if SF} = 0 \text{ then jump}
\]

Example:

```assembly
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
```
Short Jump if Not Zero (not equal). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.

Algorithm:

    if ZF = 0 then jump

Example:

```assembly
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
```

Short Jump if Overflow.

Algorithm:

    if OF = 1 then jump

Example:

```assembly
; -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
```
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:
PRINT 'parity even.'
exit:
RET
```

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

Algorithm:

if PF = 1 then jump

Example:

```
include 'emu8086.inc'

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

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 = 1 then jump

Example:

```
include 'emu8086.inc'

ORG 100h
MOV AL, 00000101b ; AL = 5
OR AL, 0 ; just set flags.
JPO label1
PRINT 'parity odd.'
JMP exit
label1:
PRINT 'parity even.'
exit:
RET
```
are checked. Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.

Algorithm:

\[
\text{if PF} = 0 \text{ then jump}
\]

Example:

```assembly
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
```

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

Algorithm:

\[
\text{if SF} = 1 \text{ then jump}
\]

Example:

```assembly
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
```

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

Algorithm:

\[
\text{if ZF} = 0 \text{ then jump}
\]

Example:

```assembly
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
```
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JZ</td>
<td>if ZF = 1 then jump</td>
</tr>
<tr>
<td>label</td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>include 'emu8086.inc'</td>
</tr>
<tr>
<td></td>
<td>ORG 100h</td>
</tr>
<tr>
<td></td>
<td>MOV AL, 5</td>
</tr>
<tr>
<td></td>
<td>CMP AL, 5</td>
</tr>
<tr>
<td></td>
<td>JZ label1</td>
</tr>
<tr>
<td></td>
<td>PRINT 'AL is not equal to 5.'</td>
</tr>
<tr>
<td></td>
<td>JMP exit</td>
</tr>
<tr>
<td></td>
<td>label1:</td>
</tr>
<tr>
<td></td>
<td>PRINT 'AL is equal to 5.'</td>
</tr>
<tr>
<td></td>
<td>exit:</td>
</tr>
<tr>
<td></td>
<td>RET</td>
</tr>
</tbody>
</table>

- **LAHF**
  - No operands
  - Load AH from 8 low bits of Flags register.
  - Algorithm:
    - \[ AH = \text{flags register} \]
  - 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
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Assembly Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDS REG, memory</td>
<td>Load DS from memory location.</td>
<td>LDS AX, m&lt;br&gt;RET&lt;br&gt;m DW 1234h&lt;br&gt;DW 5678h&lt;br&gt;END</td>
</tr>
</tbody>
</table>

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

---

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Algorithm</th>
</tr>
</thead>
</table>
| 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

---

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Algorithm</th>
</tr>
</thead>
</table>
|            | Load memory double word into word register and ES. | **Algorithm:**  
  - |
LES
REG, memory

- REG = first word
- ES = second word

Example:

```assembly
ORG 100h
LES AX, m
RET
m DW 1234h
   DW 5678h
END
```

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

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:

```assembly
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'
```
**LODSW**

**No operands**

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
```

**LOOP**

**label**

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
labell: PRINTN 'loop!'
LOOP labell
RET
```

```
C Z S O P A
unchanged
```
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Algorithm</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOPNE label</td>
<td>Decrease CX, jump to label if CX not zero and Not Equal (ZF = 0).</td>
<td>- CX = CX - 1</td>
<td>include 'emu8086.inc'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- if (CX &lt;&gt; 0) and (ZF = 0) then</td>
<td>ORG 100h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- jump</td>
<td>MOV AX, 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- else</td>
<td>MOV CX, 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- no jump, continue</td>
<td>label1:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PUTC '*'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ADD AX, 100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CMP AH, 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LOOPE label1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RET</td>
</tr>
</tbody>
</table>

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
    LOOPNE label1
    RET
v1 db 9, 8, 7, 6, 5

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

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

Algorithm:

- CX = CX - 1
• if (CX <> 0) and (ZF = 1) then
  o jump
else
  o 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

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.
| 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
```

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

```
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)

---

MUL

REG memory

Unsigned multiply.

Algorithm:

- **when operand is a byte:**
  \[ AX = AL \times \text{operand} \]

- **when operand is a word:**
  \[ (DX AX) = AX \times \text{operand} \]

Example:

```
MOV AL, 200   ; AL = 0C8h
MOV BL, 4
MUL BL        ; AX = 0320h (800)
RET
```

CF=OF=0 when high section of the result is zero.

---

NEG

REG memory

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)
NEG AL      ; AL = 05h (5)
```
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Algorithm</th>
<th>Example</th>
</tr>
</thead>
</table>
| **NOP**     | No Operation. | Do nothing | ; do nothing, 3 times: 
NOP 
NOP 
NOP 
RET |
| **NOT**     | Invert each bit of the operand. | if bit is 1 turn it to 0. 
if bit is 0 turn it to 1. | MOV AL, 00011011b 
NOT AL ; AL = 11100100b 
RET |
| **OR**      | Logical OR between all bits of two operands. 
Result is stored in first operand. | | REG, memory |
| OR | memory, REG  
REG, REG  
memory, immediate  
REG, immediate |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Example:</td>
<td></td>
</tr>
</tbody>
</table>
| MOV AL, 'A' ; AL = 01000001b  
OR AL, 00100000b ; AL = 01100001b ('a')  
RET |
|  |

| 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. |
|  |

| POP | REG  
SREG  
memory |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Get 16 bit value from the stack.</td>
<td></td>
</tr>
<tr>
<td>Algorithm:</td>
<td></td>
</tr>
</tbody>
</table>
| - operand = SS:[SP] (top of the stack)  
- SP = SP + 2 |
| Example: |
| MOV AX, 1234h  
PUSH AX  
POP DX ; DX = 1234h  
RET |
|  |

<p>|  | Pop all general purpose registers DI, SI, BP, SP, BX, DX, CX, AX from the stack. |</p>
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>POPA</td>
<td>No operands</td>
<td>SP value is ignored, it is Popped but not set to SP register. Note: this instruction works only on 80186 CPU and later!</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- POP DI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- POP SI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- POP BP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- POP xx  (SP value ignored)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- POP BX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- POP DX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- POP CX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- POP AX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>POPF</td>
<td>No operands</td>
<td>Get flags register from the stack.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- flags = SS:[SP] (top of the stack)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SP = SP + 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH</td>
<td>REG SREG memory immediate</td>
<td>Store 16 bit value in the stack. Note: PUSH immediate works only on 80186 CPU and later!</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SP = SP - 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SS:[SP] (top of the stack) = operand</td>
</tr>
</tbody>
</table>

**Example:**

```
MOV AX, 1234h
PUSH AX
POP DX     ; DX = 1234h
RET
```
### 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

### PUSHF

**No operands**

Store flags register in the stack.

Algorithm:

- SP = SP - 2
- SS:[SP] (top of the stack) = flags

### 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
```

OF=0 if first operand keeps original sign.

---

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
```

OF=0 if first operand keeps original sign.

---

Repeat following MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW instructions CX times.

Algorithm:

```
check_cx:
if CX <> 0 then
  do following chain instruction
```
REP chain instruction

- \( CX = CX - 1 \)
- go back to check\_cx

else
- exit from REP cycle

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

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
REPNE chain instruction

- CX = CX - 1
- if ZF = 0 then:
  - go back to check_cx
  - exit from REPNE cycle
- exit from REPNE cycle

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
    - exit from REPNE cycle
  - exit from REPNE cycle

Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 1 (result is Zero), maximum CX times.

Algorithm:

check_cx:

if CX <> 0 then
REPZ

chain instruction

- 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

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

RETF

No operands or even immediate

Return from Far procedure.

Algorithm:

- Pop from stack:
  - IP
  - CS
- if immediate operand is present:
SP = SP + operand

unchanged

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.

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

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.

SAL

memory, immediate
REG, immediate

memory, CL
REG, CL

Shift Arithmetic 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, 0E0h ; AL = 11100000b
SAL AL, 1 ; AL = 11000000b, CF=1.
RET

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=1.
MOV BL, 4Ch ; BL = 01001100b
SAR BL, 1 ; BL = 00100110b, CF=0.
RET
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Algorithm</th>
</tr>
</thead>
</table>
| SBB         | Subtract with Borrow. | \[
\text{operand1} = \text{operand1} - \text{operand2} - \text{CF}
\]
|             | Example: | STC
MOV AL, 5
SBB AL, 3 ; AL = 5 - 3 - 1 = 1
RET |
| SCASB       | Compare bytes: AL from ES:[DI]. | AL - ES:[DI]  
set flags according to result: OF, SF, ZF, AF, PF, CF  
if DF = 0 then  
\(\text{DI} = \text{DI} + 1\)  
else  
\(\text{DI} = \text{DI} - 1\) |
| SCASW       | Compare words: AX from ES:[DI]. | AX - ES:[DI]  
set flags according to result: OF, SF, ZF, AF, PF, CF  
if DF = 0 then  
\(\text{DI} = \text{DI} + 2\) |
<table>
<thead>
<tr>
<th>SHL</th>
<th>Memory, immediate</th>
<th>REG, immediate</th>
<th>Memory, CL</th>
<th>REG, CL</th>
</tr>
</thead>
</table>

### 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:**

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

OF=0 if first operand keeps original sign.

<table>
<thead>
<tr>
<th>SHR</th>
<th>Memory, immediate</th>
<th>REG, immediate</th>
<th>Memory, CL</th>
<th>REG, CL</th>
</tr>
</thead>
</table>

### 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:**

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

OF=0 if first operand keeps original sign.
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operands</th>
<th>Description</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STC</strong></td>
<td>No operands</td>
<td>Set Carry flag.</td>
<td>CF = 1</td>
</tr>
<tr>
<td><strong>STD</strong></td>
<td>No operands</td>
<td>Set Direction flag. SI and DI will be decremented by chain instructions: CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW.</td>
<td>DF = 1</td>
</tr>
<tr>
<td><strong>STI</strong></td>
<td>No operands</td>
<td>Set Interrupt enable flag. This enables hardware interrupts.</td>
<td>IF = 1</td>
</tr>
<tr>
<td><strong>STOSB</strong></td>
<td>No operands</td>
<td>Store byte in AL into ES:[DI]. Update DI.</td>
<td></td>
</tr>
</tbody>
</table>

Algorithm:

- $\text{ES:}[\text{DI}] = \text{AL}$
- if $DF = 0$ then
  - $\text{DI} = \text{DI} + 1$
- else
  - $\text{DI} = \text{DI} - 1$

**Example:**

```
ORG 100h
LEA DI, al
```
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
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:**

```assembly
MOV AL, 00000101b
TEST AL, 1         ; ZF = 0.
TEST AL, 10b       ; ZF = 1.
RET
```

**Exchange values of two operands.**

**Algorithm:**

`operand1 < - > operand2`

**Example:**

```assembly
MOV AL, 5
MOV AH, 2
XCHG AL, AH     ; AL = 2, AH = 5
XCHG AL, AH     ; AL = 5, AH = 2
RET
```

**Translate byte from table.**

Copy value of memory byte at DS:[BX + unsigned AL] to AL register.

**Algorithm:**
XLATB

No operands

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

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