|  |  |  |
| --- | --- | --- |
| Echipa | Operanzi | Descriere |
| AAA | Fără operanzi | Corectează formatul ASCII după adăugare. Corectează rezultatul din registrele AH și AL după adunare atunci când se lucrează cu valori BCD.  Funcționează în conformitate cu următorul algoritm  Dacă cei patru biți inferiori (din dreapta) din din registrul AL > 9 sau flag-ul AF = 1, atunci:   * AL = AL + 6 * AH = AH + 1 * AF = 1 * CF = 1   altfel   * AF = 0 * CF = 0   în orice caz: ștergeți cei mai mari patru biți din registrul AL.  Exemplu:  MOV AX, 15 ; AH = 00, AL = 0Fh  AAA ; AH = 01, AL = 05  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | ? | ? | ? | ? | r | |
| AAD | Fără operanzi | Corecția formatului ASCII înainte de divizare. Pregătește două valori BCD pentru împărțire.  Algoritm:   * AL = (AH \* 10) + AL * AH = 0   Exemplu:  MOV AX, 0105h ; AH = 01, AL = 05  AAD ; AH = 00, AL = 0Fh (15)  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | ? | r | r | ? | r | ? | |
| AAM | Fără operanzi | Corectează formatul ASCII după înmulțire. Corectează rezultatul înmulțirii a două valori BCD.  Algoritm:   * AH = AL / 10 * AL = restul   Exemplu:  MOV AL, 15 ; AL = 0Fh  AAM ; AH = 01, AL = 05  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | ? | r | r | ? | r | ? | |
| AAS | Fără operanzi | Corectează formatul ASCII după scădere. Corectează rezultatul din registrele AH și AL după scădere atunci când se lucrează cu valori BCD.  Algoritm:  Dacă cei mai mici patru biți ai registrul AL > 9 sau AF = 1, atunci:   * AL = AL - 6 * AH = AH - 1 * AF = 1 * CF = 1   altfel   * AF = 0 * CF = 0   în orice caz: ștergeți cei patru biți superiori ai registrului AL.  Exemplu:  MOV AX, 02FFh ; AH = 02, AL = 0FFh  AAS ; AH = 01, AL = 09  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | ? | ? | ? | ? | r | |
| ADC | REG, memorie memorie, REG REG, REG memorie, imediată  REG, imediată | Adăugare cu transfer.   Algoritm:  operand1 = operand1 + operand2 + CF  Exemplu:  STC ; setați CF = 1  MOV AL, 5 ; AL = 5  ADC AL, 1 ; AL = 7  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | r | r | r | r | r | |
| ADD | REG, memorie memorie, REG REG, REG memorie, imediată  REG, imediată | Adăugare.   Algoritm:  operand1 = operand1 + operand2  Exemplu:  MOV AL, 5 ; AL = 5  ADD AL, -3 ; AL = 2  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | r | r | r | r | r | |
| ȘI | REG, memorie memorie, REG REG, REG memorie, imediată  REG, imediată | ȘI logic între toți biții celor doi operanzi. Rezultatul este scris în primul operand.  Se aplică următoarele reguli:  1 ȘI 1 = 1 1  1 ȘI 0 = 0  ȘI 1 = 0 0  ȘI 0 = 0   Exemplu:  MOV AL, 'a' ; AL = 01100001b  AND AL, 1101111111b ; AL = 01000001b ('A')  RET   |  |  |  |  |  | | --- | --- | --- | --- | --- | | C | Z | S | O | P | | 0 | r | r | 0 | r | |
| CALL | numele procedurii etichetă adresă de  4 octeți | Transmite controlul către procedură, stivuiește adresa următoarei comenzi (de la IP). **Adresa de 4 octeți** poate fi introdusă în următoarea formă: 1234h:5678h, prima valoare este segmentul, a doua valoare este decalajul (în cazul unui apel la distanță, registrul CS este de asemenea pus pe stivă).   Exemplu:  #make\_COM#  ORG 100h ; pentru fișierul COM.  CALL p1  ADD AX, 1  RET ; întoarcerea la sistemul de operare.  p1 PROC ; declarație de procedură.  MOV AX, 1234h  RET ; revenire la program.  p1 ENDP   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| CBW | Fără operanzi | Convertește un octet într-un cuvânt.  Algoritm:  Dacă cel mai înalt bit AL = 1, atunci:   * AH = 255 (0FFh)   altfel   * AH = 0   Exemplu:  MOV AX, 0 ; AH = 0, AL = 0  MOV AL, -5 ; AX = 000FBh (251)  CBW ; AX = 0FFFBh (-5)  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| CLC | Fără operanzi | Șterge indicatorul de transfer (CF).  Algoritm:  CF = 0   |  | | --- | | C | | 0 | |
| CLD | Fără operanzi | Ștergeți indicatorul de direcție (DF). Ștergeți indicatorul de direcție. Valorile registrelor SI și DI vor fi incrementate de comenzile: CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW.  Algoritm:  DF = 0   |  | | --- | | D | | 0 | |
| CLI | Fără operanzi | Șterge indicatorul de întrerupere (IF). Acest lucru dezactivează întreruperile hardware.  Algoritm:  IF = 0   |  | | --- | | I | | 0 | |
| CMC | Fără operanzi | Comută flagul de transport, adică inversează valoarea CF.  Algoritm:  dacă CF = 1 atunci CF = 0 dacă CF = 0 atunci CF = 1   |  | | --- | | C | | r | |
| CMP | REG, memorie memorie, REG REG, REG memorie, imediată  REG, imediată | Comparație.  Algoritm:  operand1 - operand2.  rezultatul nu se scrie nicăieri, se setează stegulețele (OF, SF, ZF, AF, PF, CF) în funcție de rezultat.  Exemplu:  MOV AL, 5  MOV BL, 5  CMP AL, BL ; AL = 5, ZF = 1 (egal!)  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | r | r | r | r | r | |
| CMPSB | Fără operanzi | Compară octeți: ES:[DI] din DS:[SI].  Algoritm:   * DS:[SI] - ES:[DI] * setează stegulețele în funcție de rezultat:  OF, SF, ZF, AF, PF, CF * Dacă DF = 0, atunci   + SI = SI + 1   + DI = DI + 1   altfel   * + SI = SI - 1   + DI = DI - 1   Exemplu: vezi [**cmpsb.asm**](http://www.avprog.narod.ru/progs/Samples/cmpsb.asm) în directorul Samples.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | r | r | r | r | r | |
| CMPSW | Fără operanzi | Compară cuvintele: ES:[DI] cu DS:[SI].  Algoritm:   * DS:[SI] - ES:[DI] * setează stegulețele în funcție de rezultat:  OF, SF, ZF, AF, PF, CF * Dacă DF = 0, atunci   + SI = SI + 2   + DI = DI + 2   altfel   * + SI = SI - 2   + DI = DI - 2   Exemplu: vezi [**cmpsw.asm**](http://www.avprog.narod.ru/progs/Samples/cmpsw.asm) în directorul Samples.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | r | r | r | r | r | |
| CWD | Fără operanzi | Convertește cuvântul într-un cuvânt dublu.  Algoritm:  Dacă bitul înalt AX = 1, atunci:   * DX = 65535 (0FFFFh)   altfel   * DX = 0   Exemplu:  MOV DX, 0 ; DX = 0  MOV AX, 0 ; AX = 0  MOV AX, -5 ; DX AX = 00000h:0FFFBh  CWD ; DX AX = 0FFFFFFFFh:0FFFBh  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| DAA | Fără operanzi | Corecția zecimală după adunare. Corectează rezultatul adunării a două valori BCD ambalate.  Algoritm:  Dacă cei mai mici patru biți AL > 9 sau AF = 1, atunci:   * AL = AL + 6 * AF = 1   dacă AL > 9Fh sau CF = 1, atunci   * AL = AL + 60h * CF = 1   Exemplu:  MOV AL, 0Fh ; AL = 0Fh (15)  DAA ; AL = 15h  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | r | r | r | r | r | |
| DAS | Fără operanzi | Corecția zecimală după scădere. Corectează rezultatul scăderii a două valori BCD ambalate.  Algoritm:  Dacă cei mai mici patru biți AL > 9 sau AF = 1, atunci:   * AL = AL - 6 * AF = 1   dacă AL > 9Fh sau CF = 1, atunci   * AL = AL - 60h * CF = 1   Exemplu:  MOV AL, 0FFh ; AL = 0FFh (-1)  DAS ; AL = 99h, CF = 1  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | r | r | r | r | r | |
| DEC | REG memorie | Scădere.  Algoritm:  operand = operand - 1   Exemplu:  MOV AL, 255 ; AL = 0FFh (255 sau -1)  DEC AL ; AL = 0FEh (254 sau -2)  RET   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Z | S | O | P | A | | r | r | r | r | r |   CF - nicio schimbare! |
| DIV | REG memorie | Diviziune fără semn.  Algoritm:  în cazul în care operandul este un **octet**: AL = AX / operand  AH = restul (modulo)  în cazul în care operandul este **un cuvânt**: AX = (DX AX) / operand  DX = restul (modulo)  Exemplu:  MOV AX, 203 ; AX = 00CBh  MOV BL, 4  DIV BL ; AL = 50 (32h), AH = 3  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | ? | ? | ? | ? | ? | ? | |
| HLT | Fără operanzi | Oprirea sistemului.  Exemplu:  MOV AX, 5  HLT   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | neschimbat | | | | | | |
| IDIV | REG memorie | Divizia de semne.  Algoritm:  în cazul în care operandul este un **octet**: AL = AX / operand  AH = restul (modulo)  în cazul în care operandul este **un cuvânt**: AX = (DX AX) / operand  DX = restul (modulo)  Exemplu:  MOV AX, -203 ; AX = 0FF35h  MOV BL, 4  IDIV BL ; AL = -50 (0CEh), AH = -3 (0FDh)  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | ? | ? | ? | ? | ? | ? | |
| IMUL | REG memorie | Înmulțirea semnelor.  Algoritm:  în cazul în care operandul este un **octet**: AX = AL \* operand.  în cazul în care operandul este **un cuvânt**: (DX AX) = AX \* operand.  Exemplu:  MOV AL, -2  MOV BL, -4  IMUL BL ; AX = 8  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | ? | ? | r | ? | ? |   CF=OF=0 în cazul în care rezultatul se potrivește într-un operand IMUL. |
| ÎN | AL , im.byte  AL, DX  AX, im.byte  AX, DX | Plasează datele din port în **AL** sau **AX**. Al doilea operand este numărul portului. Dacă trebuie accesat un număr de port mai mare de 255, atunci trebuie utilizat registrul **DX**. Exemplu:  IN AX, 4 ; obține starea semaforului.  IN AL, 7 ; obține starea motorului pas cu pas.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| INC | REG memorie | Creștere.  Algoritm:  operand = operand + 1  Exemplu:  MOV AL, 4  INC AL ; AL = 5  RET   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Z | S | O | P | A | | r | r | r | r | r |   CF - nicio schimbare! |
| INT | byte | Execută întreruperea programului și transferă controlul către funcția specificată în octetul imediat (0..255).  Algoritm:  Așezați pe stivă:   * + caz de pavilion   + CS   + IP * IF = 0 * Transmiterea controlului către o procedură de întrerupere   Exemplu:  MOV AH, 0Eh ; teletype.  MOV AL, 'A'  INT 10h ; întrerupere BIOS.  RET   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | I | | rămân neschimbate | | | | | | 0 | |
| INTO | Fără operanzi | Provoacă o întrerupere atunci când se produce o depășire de capacitate (flag OF = 1) și execută comanda IRET 4.  Algoritm:  Dacă OF = 1, atunci INT 4  Exemplu:  ; -5 - 127 = -132 (în afara intervalului -128..127)  ; rezultatul scăderii (SUB) este incorect (124),  ; prin urmare, OF = 1:  MOV AL, -5  SUB AL, 127 ; AL = 7Ch (124)  INTO ; o eroare de proces.  RET |
| IRET | Fără operanzi | Întoarcerea de la gestionarea întreruperilor.  Algoritm:  Descărcați din stivă:   * + IP   + CS   + registrul de steaguri  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | încărcat | | | | | | |
| JA | etichetă | Prescurtare de la "peste" sau "nu mai jos sau egal cu". Se utilizează după verificarea datelor fără semn pentru a transfera controlul la o altă adresă.  Algoritm:  dacă (CF = 0) și (ZF = 0) atunci se face tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 250  CMP AL, 5  JA label1  PRINT 'AL nu mai mult de 5'  JMP exit  label1:  PRINT 'AL este mai mare decât 5'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JAE | etichetă | Salt scurt dacă primul operand este "mai mare sau egal cu" al doilea operand. (Ca rezultat al executării unei instrucțiuni CMP). Fără semn.  Algoritm:  dacă CF = 0, atunci se face tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 5  CMP AL, 5  JAE label1  PRINT 'AL nu mai mult de 5'  JMP exit  label1:  PRINT 'AL este mai mare sau egal cu 5'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JB | etichetă | Un salt scurt dacă primul operand este "mai mic decât" al doilea operand (ca rezultat al instrucțiunii CMP). Fără semn.  Algoritm:  dacă CF = 1, se face tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 1  CMP AL, 5  JB label1  PRINT 'AL cel puțin 5'  JMP exit  label1:  PRINT 'AL mai mic de 5'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JBE | etichetă | Salt scurt dacă primul operand este "mai mic decât" sau "egal cu" al doilea operand (ca rezultat al unei instrucțiuni CMP). Fără semn.  Algoritm:  dacă CF = 1 sau ZF = 1, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 5  CMP AL, 5  JBE label1  PRINT 'AL cel puțin 5'  JMP exit  label1:  PRINT 'AL este mai mic sau egal cu 5'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JC | etichetă | Salt scurt dacă flagul de transport este setat la 1.  Algoritm:  dacă CF = 1, atunci se face tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 255  ADD AL, 1  JC label1  PRINT "niciun transfer.  JMP exit  label1:  PRINT "să aibă un transfer.  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JCXZ | etichetă | Salt scurt dacă registrul CX este 0.  Algoritm:  dacă CX = 0, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV CX, 0  JCXZ label1  PRINT 'CX nu este zero.'  JMP exit  label1:  PRINT 'CX este zero.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JE | etichetă | Salt scurt dacă primul operand este egal cu al doilea operand (ca rezultat al instrucțiunii CMP). Semnat/ne-semnat.  Algoritm:  dacă ZF = 1, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 5  CMP AL, 5  JE label1  PRINT 'AL nu este egal cu 5.'  JMP exit  label1:  PRINT 'AL este egal cu 5.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JG | etichetă | Salt scurt dacă primul operand este mai mare decât al doilea operand (ca rezultat al instrucțiunii CMP). Semnat.  Algoritm:  în cazul în care (ZF = 0) și (SF = OF), atunci se face o tranziție  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 5  CMP AL, -5  JG eticheta1  PRINT 'AL nu mai mult de -5.'  JMP exit  label1:  PRINT 'AL este mai mare decât -5.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JGE | etichetă | Salt scurt dacă primul operand este mai mare sau egal cu al doilea operand (ca rezultat al unei instrucțiuni CMP). Semnat.  Algoritm:  dacă SF = OF, atunci se face o tranziție  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 2  CMP AL, -5  JGE eticheta1  PRINT 'AL < -5'  JMP exit  label1:  PRINT 'AL >= -5'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JL | etichetă | Salt scurt dacă primul operand este mai mic decât al doilea operand (ca rezultat al instrucțiunii CMP). Semnat.  Algoritm:  dacă SF <> OF, atunci se face o tranziție  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, -2  CMP AL, 5  JL label1  PRINT 'AL >= 5.'  JMP exit  label1:  PRINT 'AL < 5.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JLE | etichetă | Salt scurt dacă primul operand este mai mic sau egal cu al doilea operand (ca rezultat al unei instrucțiuni CMP). Semnat.  Algoritm:  În cazul în care SF <> OF sau ZF = 1, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, -2  CMP AL, 5  JLE label1  PRINT 'AL > 5.'  JMP exit  label1:  PRINT 'AL <= 5.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JMP | tag  4 octeți adresă | Transfer necondiționat. Transferă controlul către o altă parte a programului. În această formă se poate introduce **o adresă de 4 octeți**: 1234h:5678h, prima valoare este segmentul, iar a doua valoare este decalajul.   Algoritm:  să facă tranziția în orice caz  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 5  JMP label1 ; "Treceți peste" două linii!  PRINT 'Nu există tranziție!'  MOV AL, 0  label1:  PRINT 'Am ajuns aici!  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JNA | etichetă | Salt scurt dacă primul operand nu este mai mare decât al doilea operand (ca rezultat al unei instrucțiuni CMP). Fără semn.  Algoritm:  dacă CF = 1 sau ZF = 1, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 2  CMP AL, 5  Etichetă JNA1  PRINT 'AL este mai mare decât 5.'  JMP exit  label1:  PRINT 'AL nu mai mult de 5.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JNAE | etichetă | Salt scurt dacă primul operand nu este mai mare sau egal cu al doilea operand (ca rezultat al unei instrucțiuni CMP). Fără semn.  Algoritm:  dacă CF = 1, atunci se face tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 2  CMP AL, 5  JNAE label1  PRINT 'AL >= 5.'  JMP exit  label1:  PRINT 'AL < 5.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JNB | etichetă | Salt scurt dacă primul operand nu este mai mic decât al doilea operand (ca rezultat al unei instrucțiuni CMP). Fără semn.  Algoritm:  dacă CF = 0, atunci se face tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 7  CMP AL, 5  JNB eticheta1  PRINT 'AL < 5.'  JMP exit  label1:  PRINT 'AL >= 5.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JNBE | etichetă | Salt scurt dacă primul operand nu este mai mic sau egal cu al doilea operand (ca rezultat al unei instrucțiuni CMP). Fără semn.  Algoritm:  dacă (CF = 0) și (ZF = 0) atunci se face tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 7  CMP AL, 5  JNBE label1  PRINT 'AL <= 5.'  JMP exit  label1:  PRINT 'AL > 5.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JNC | etichetă | O tranziție scurtă dacă flagul de transport este setat la zero.  Algoritm:  dacă CF = 0, atunci se face tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 2  ADD AL, 3  Eticheta JNC1  PRINT "să aibă un transfer.  JMP exit  label1:  PRINT "niciun transfer.  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JNE | etichetă | Salt scurt dacă primul operand nu este egal cu al doilea operand (ca rezultat al instrucțiunii CMP). Semnat/ne-semnat.  Algoritm:  dacă ZF = 0, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 2  CMP AL, 3  JNE label1  PRINT 'AL = 3.'  JMP exit  label1:  PRINT 'Al <> 3.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JNG | etichetă | Salt scurt dacă primul operand nu este mai mare decât al doilea operand (ca rezultat al unei instrucțiuni CMP). Semnat.  Algoritm:  dacă (ZF = 1) și (SF <> OF) atunci se face tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 2  CMP AL, 3  JNG label1  PRINT 'AL > 3.'  JMP exit  label1:  PRINT 'Al <= 3.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JNGE | etichetă | Salt scurt dacă primul operand nu este mai mare sau egal cu al doilea operand (ca rezultat al unei instrucțiuni CMP). Semnat.  Algoritm:  dacă SF <> OF, atunci se face o tranziție  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 2  CMP AL, 3  JNGE eticheta1  PRINT 'AL >= 3.'  JMP exit  label1:  PRINT 'Al < 3.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JNL | etichetă | Salt scurt dacă primul operand nu este mai mic decât al doilea operand (ca rezultat al unei instrucțiuni CMP). Semnat.  Algoritm:  dacă SF = OF, atunci se face o tranziție  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 2  CMP AL, -3  JNL eticheta1  PRINT 'AL < -3.'  JMP exit  label1:  PRINT 'Al >= -3.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JNLE | etichetă | Salt scurt dacă primul operand nu este mai mic sau egal cu al doilea operand (ca rezultat al unei instrucțiuni CMP). Semnat.  Algoritm:  dacă (SF = OF) și (ZF = 0), atunci se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 2  CMP AL, -3  JNLE label1  PRINT 'AL <= -3.'  JMP exit  label1:  PRINT 'Al > -3.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JNO | etichetă | O tranziție scurtă dacă nu există o depășire.  Algoritm:  dacă OF = 0, atunci se efectuează tranziția  Exemplu:  ; -5 - 2 = -7 (între -128..127)  ; rezultatul comenzii SUB (subtragere) este corect,  ; prin urmare, OF = 0:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, -5  SUB AL, 2 ; AL = 0F9h (-7)  JNO label1  PRINT 'overflow!'  JMP exit  label1:  PRINT 'no overflow.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămâne neschimbată | | | | | | |
| JNP | etichetă | Salt scurt dacă nu există paritate sau dacă paritatea este impară. Se verifică doar cei mai mici 8 biți ai rezultatului. Setează prin comenzile CMP, SUB, ADD, TEST, AND, OR, XOR.  Algoritm:  dacă PF = 0, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 00000111b ; AL = 7  OR AL, 0 ; setează numai steagurile.  JNP eticheta1  PRINT 'paritatea este pară.'  JMP exit  label1:  PRINT 'paritatea este impară.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JNS | etichetă | Salt scurt dacă nu există niciun semn (dacă este pozitiv). Setează prin comenzile CMP, SUB, ADD, TEST, AND, OR, XOR.  Algoritm:  dacă SF = 0, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 00000111b ; AL = 7  OR AL, 0 ; setează numai steagurile.  JNS eticheta1  PRINT "există un semn.  JMP exit  label1:  PRINT "niciun semn.  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JNZ | etichetă | Salt scurt dacă "nu este zero". Setează prin comenzile CMP, SUB, ADD, TEST, AND, OR, XOR.  Algoritm:  dacă ZF = 0, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 00000111b ; AL = 7  OR AL, 0 ; doar setarea flagului  JNZ eticheta1  PRINT 'zero.'  JMP exit  label1:  PRINT 'not zero.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JO | etichetă | Un salt scurt pe supraîncărcare.  Algoritm:  dacă OF = 1, atunci se efectuează tranziția  Exemplu:  ; -5 - 127 = -132 (în afara intervalului -128..127)  ; rezultatul scăderii este incorect (124),  ; prin urmare, OF = 1:  include 'emu8086.inc'  #make\_COM#  org 100h  MOV AL, -5  SUB AL, 127 ; AL = 7Ch (124)  JO label1  PRINT 'no overflow.'  JMP exit  label1:  PRINT 'overflow!'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JP | etichetă | Salt scurt dacă există paritate sau dacă paritatea este pară. Se verifică numai cei mai mici 8 biți ai rezultatului. Setează prin comenzile CMP, SUB, ADD, TEST, AND, OR, XOR.  Algoritm:  dacă PF = 1, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 00000101b ; AL = 5  OR AL, 0 ; setează numai steagurile.  JP label1  PRINT 'paritatea este impară.'  JMP exit  label1:  PRINT 'paritatea este pară.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JPE | etichetă | Salt scurt dacă există paritate sau dacă paritatea este pară. Se verifică numai cei mai mici 8 biți ai rezultatului. Setează prin comenzile CMP, SUB, ADD, TEST, AND, OR, XOR.  Algoritm:  dacă PF = 1, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 00000101b ; AL = 5  OR AL, 0 ; setează numai steagurile.  Etichetă JPE1  PRINT 'paritatea este impară.'  JMP exit  label1:  PRINT 'paritatea este pară.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JPO | etichetă | Salt scurt dacă nu există paritate sau dacă paritatea este impară. Se verifică doar cei mai mici 8 biți ai rezultatului. Setează prin comenzile CMP, SUB, ADD, TEST, AND, OR, XOR.  Algoritm:  dacă PF = 0, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 00000111b ; AL = 7  OR AL, 0 ; setează numai steagurile.  JPO eticheta1  PRINT 'paritatea este pară.'  JMP exit  label1:  PRINT 'paritatea este impară.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JS | etichetă | Semnați tranziția (dacă este negativă). Setează prin comenzile CMP, SUB, ADD, TEST, AND, OR, XOR.  Algoritm:  dacă SF = 1, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 10000000b ; AL = -128  OR AL, 0 ; setează numai steagurile.  JS label1  PRINT "niciun semn.  JMP exit  label1:  PRINT "există un semn.  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| JZ | etichetă | Salt scurt la "zero". Setează prin comenzile CMP, SUB, ADD, TEST, AND, OR, XOR.  Algoritm:  dacă ZF = 1, se efectuează tranziția  Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AL, 5  CMP AL, 5  JZ eticheta1  PRINT 'AL nu este egal cu 5.'  JMP exit  label1:  PRINT 'AL este egal cu 5.'  ieșire:  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| LAHF | Fără operanzi | Încarcă cei 8 octeți inferiori ai registrului indicator în registrul NA.  Algoritm:  AH = Registrul de semnalizare  Biți AH: 7 6 5 4 3 2 1 1 0  [SF] [ZF] [0] [AF] [0] [PF] [1] [CF]  biții 1, 3, 5 sunt rezervați.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| LDS | REG, memorie | Încarcă patru octeți de memorie (cuvânt dublu) în registrele necesare care conțin adresa relativă și adresa segmentului. Adresa segmentului este plasată în registrul DS, iar adresa relativă (offset) este plasată în oricare dintre registrele generale sau de index sau într-un pointer de registru.  Algoritm:   * REG = primul cuvânt * DS = al doilea cuvânt   Exemplu:  #make\_COM#  ORG 100h  LDS AX, m  RET  m DW 1234h  DW 5678h  ÎNCHEIERE  AX stochează valoarea 1234h, DS stochează valoarea 5678h.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| LEA | REG, memorie | Încarcă adresa de execuție. Comanda LEA încarcă adresa relativă a celui de-al doilea operand (nu valoarea operandului!) în registrul specificat în comandă ca prim operand. Primul operand trebuie să fie un registru de uz general (nu un registru de segment), iar al doilea trebuie să fie o locație de memorie. Comanda   LEA reg, mem  este echivalentă cu comanda  MOV reg,offset mem  dar prima comandă are mai multe posibilități de a descrie adresa celulei care ne interesează. Comanda nu are nici un efect asupra indicatoarelor procesorului.  Algoritm:   * REG = adresa de memorie (offset)   De obicei, această comandă este înlocuită cu comanda MOV, dacă este posibil.  Exemplu:  #make\_COM#  ORG 100h  LEA AX, m  RET  m DW 1234h  ÎNCHEIERE  Se scrie o valoare în AX: 0104h. Comanda LEA ocupă 3 octeți, comanda RET ocupă 1 octet, începem de la adresa 100h, deci adresa "m" este 104h.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| LES | REG, memorie | Încarcă un pointer utilizând registrul ES. Citește un cuvânt dublu (32 de biți) care conține un pointer (adresa completă a unei celule) din memorie la adresa specificată și încarcă jumătatea inferioară a pointerului (adică adresa relativă) în registrul specificat în comandă și jumătatea superioară a pointerului (adică adresa segmentului) în registrul ES. Astfel, comanda   LES reg, mem  este echivalentă cu următorul grup de comenzi:   MOV reg, word ptr mem  MOV ES, word ptr mem+2  Primul operand al comenzii LES este un registru de uz general; al doilea este o celulă de memorie cu un conținut de două cuvinte. Pointerul conținut în această celulă poate fi adresa fie a procedurii, fie a câmpului de date.    Algoritm:   * REG = primul cuvânt * ES = al doilea cuvânt   Exemplu:  #make\_COM#  ORG 100h  LES AX, m  RET  m DW 1234h  DW 5678h  ÎNCHEIERE  AX stochează valoarea 1234h, ES stochează valoarea 5678h.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| LODSB | Fără operanzi | Încarcă un octet din DS:[SI] în registrul AL. Schimbă SI.  Algoritm:   * AL = DS:[SI] * Dacă DF = 0, atunci   + SI = SI + 1   altfel   * + SI = SI - 1   Exemplu:  #make\_COM#  ORG 100h  LEA SI, a1  MOV CX, 5  MOV AH, 0Eh  m: LODSB  INT 10h  LOOP m  RET  a1 DB DB "H", "e", "l", "l", "o   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| LODSW | Fără operanzi | Încarcă cuvântul din DS:[SI] în registrul AX. Modifică SI.  Algoritm:   * AX = DS:[SI] * Dacă DF = 0, atunci   + SI = SI + 2   altfel   * + SI = SI - 2   Exemplu:  #make\_COM#  ORG 100h  LEA SI, a1  MOV CX, 5  REP LODSW ; aceasta va avea ca rezultat o valoare de 555h în AH.  RET  a1 dw 111h, 222h, 333h, 444h, 555h   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| LOOP | etichetă | Dec CX, merge la marcaj dacă CX nu este zero.  Algoritm:   * CX = CX - 1 * Dacă CX <> 0 atunci   + face tranziția   altfel   * + nu faceți tranziția, continuați ciclul   Exemplu:  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV CX, 5  label1:  PRINTN 'loop!'  LOOP label1  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| LOOPE | etichetă | Diminuează CX, execută tranziția dacă CX este "non-zero" sau "egal". (ZF = 1).  Algoritm:   * CX = CX - 1 * dacă (CX <> 0) și (ZF = 1) atunci   + să facă tranziția   altfel   * + nu faceți tranziția, continuați ciclul   Exemplu:  ; Ciclul este executat de 5 ori sau până când  ; rezultatul din registrul AL se încadrează într-un octet.  ; Rezultatul va depăși 255 la a treia "rulare" (100+100+100+100),  ; prin urmare, ciclul va fi încheiat.  include 'emu8086.inc'  #make\_COM#  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 | | rămân neschimbate | | | | | | |
| LOOPNE | etichetă | Dec CX, efectuează o tranziție la marker dacă CX nu este zero și este îndeplinită condiția "nu este egal" (ZF = 0).  Algoritm:   * CX = CX - 1 * dacă (CX <> 0) și (ZF = 0) atunci   + face tranziția   altfel   * + nu faceți tranziția, continuați ciclul   Exemplu:  ; Ciclul este executat de 5 ori sau până când,  ; până când se găsește numărul "7".  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV SI, 0  MOV CX, 5  label1:  PUTC '\*'  MOV AL, v1[SI]  INC SI ; următorul octet (SI=SI+1).  CMP AL, 7  LOOPNE label1  RET  v1 db 9, 8, 7, 7, 6, 5   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| LOOPNZ | etichetă | Dec CX, efectuează o tranziție la marcă dacă CX nu este zero și ZF = 0.  Algoritm:   * CX = CX - 1 * dacă (CX <> 0) și (ZF = 0) atunci   + face tranziția   altfel   * + nu faceți tranziția, continuați ciclul   Exemplu:  ; Ciclul este executat de 5 ori sau până când,  ; până când se găsește numărul "7".  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV SI, 0  MOV CX, 5  label1:  PUTC '\*'  MOV AL, v1[SI]  INC SI ; următorul octet (SI=SI+1).  CMP AL, 7  LOOPNZ label1  RET  v1 db 9, 8, 7, 7, 6, 5   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| LOOPZ | etichetă | Dec CX, efectuează o tranziție la marcă dacă CX nu este zero și ZF = 1.  Algoritm:   * CX = CX - 1 * dacă (CX <> 0) și (ZF = 1) atunci   + face tranziția   altfel   * + nu faceți tranziția, continuați ciclul   Exemplu:  ; Ciclul este executat de 5 ori sau până când,  ; atâta timp cât valoarea din registrul AL nu depășește limita de octeți.  ; Rezultatul va depăși 255 la al treilea ciclu (100+100+100+100),  Prin urmare, ciclul se va încheia după a treia trecere.  include 'emu8086.inc'  #make\_COM#  ORG 100h  MOV AX, 0  MOV CX, 5  label1:  PUTC '\*'  ADD AX, 100  CMP AH, 0  LOOPZ label1  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| MOV | REG, memorie memorie, REG REG, REG memorie, imediată  REG, imediată   SREG, memorie memorie, SREG  REG REG, SREG SREG, REG | Copiază operand2 în operand1.  Comanda MOV nu poate:   * scrieți date în registrele CS și IP. * Copierea datelor dintr-un registru de segment în alt registru de segment (trebuie să copiați mai întâi datele în registrul de uz general). * copiați valoarea directă în registrul de segment (trebuie să copiați mai întâi datele în registrul de uz general).   Algoritm:  operand1 = operand2  Exemplu:  #make\_COM#  ORG 100h  MOV AX, 0B800h ; setați AX = B800h (memorie VGA).  MOV DS, AX ; copiați valoarea din AX în DS.  MOV CL, 'A' ; CL = 41h (cod ASCII).  MOV CH, 01011111b ; CL = atribute de culoare.  MOV BX, 15Eh ; BX = Poziția ecranului.  MOV [BX], CX ; w.[0B800h:015Eh] = CX.  RET ; întoarcerea la sistemul de operare.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| MOVSB | Fără operanzi | Copiază un octet de la DS:[SI] la ES:[DI]. Modifică registrele SI și DI.  Algoritm:   * ES:[DI] = DS:[SI] * Dacă DF = 0, atunci   + SI = SI + 1   + DI = DI + 1   altfel   * + SI = SI - 1   + DI = DI - 1   Exemplu:  #make\_COM#  ORG 100h  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 | | rămân neschimbate | | | | | | |
| MOVSW | Fără operanzi | Copiază **un cuvânt** din DS:[SI] în ES:[DI]. Schimbă registrele SI și DI.  Algoritm:   * ES:[DI] = DS:[SI] * Dacă DF = 0, atunci   + SI = SI + 2   + DI = DI + 2   altfel   * + SI = SI - 2   + DI = DI - 2   Exemplu:  #make\_COM#  ORG 100h  LEA SI, a1  LEA DI, a2  MOV CX, 5  REP MOVSW  RET  a1 DW 1,2,2,3,3,4,5  a2 DW 5 DUP(0)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| MUL | REG memorie | Înmulțire fără semn.  Algoritm:  în cazul în care operandul este un **octet**: AX = AL \* operand.  dacă operandul este un **cuvânt**: (DX AX) = AX \* operand.  Exemplu:  MOV AL, 200 ; AL = 0C8h  MOV BL, 4  MUL BL ; AX = 0320h (800)  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | ? | ? | r | ? | ? |   CF=OF=0 în cazul în care secțiunea cu cel mai mare rezultat este zero. |
| NEG | REG memorie | Negarea. Face ca operandul să fie negativ (adunare la doi).  Algoritm:   * Inversează toți biții operandului * Adăugați unu la operandul inversat   Exemplu:  MOV AL, 5 ; AL = 05h  NEG AL ; AL = 0FBh (-5)  NEG AL ; AL = 05h (5)  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | r | r | r | r | r | |
| NOP | Fără operanzi | Nu funcționează. De obicei se utilizează pentru o ușoară întârziere a programului.  Algoritm:   * Să nu faci nimic   Exemplu:  ; nu faceți nimic de trei ori:  NOP  NOP  NOP  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| NU | REG memorie | Inversează fiecare bit al operandului.  Algoritm:   * dacă bitul este 1, îl comută la 0. * dacă bitul este 0, îl comută la 1.   Exemplu:  MOV AL, 00011011b  NOT AL ; AL = 11100100b  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| OR | REG, memorie memorie, REG REG, REG memorie, imediată  REG, imediată | Un OR logic între toți biții din cei doi operanzi. Rezultatul este scris în primul operand.  Se aplică următoarele reguli  1 SAU 1 = 1 1 1  SAU 0 = 1  0 SAU 1 = 1  0 SAU 0 = 0   Exemplu:  MOV AL, 'A' ; AL = 01000001b  OR AL, 00100000b ; AL = 01100001b ('a')  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | 0 | r | r | 0 | r | ? | |
| OUT | im.byte, AL im.byte, AX  DX, AL  DX, AX | Scoate datele din registrul **AL** sau **AX** către port. Primul operand este numărul portului. Dacă se dorește accesarea unui număr de port mai mare de 255, trebuie utilizat registrul **DX**.  Exemplu:  MOV AX, 0FFFh ; Activează toate  OUT 4, AX ; semafoare.  MOV AL, 100b ; Porniți al treilea magnet  OUT 7, AL ; a motorului pas cu pas.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| POP | REG  SREG SREG memorie | Obține o valoare pe 16 biți din stivă.  Algoritm:   * operand = SS:[SP] (partea de sus a stivei) * SP = SP + 2   Exemplu:  MOV AX, 1234h  PUSH AX  POP DX ; DX = 1234h  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| POPA | Fără operanzi | Descarcă toate registrele de uz general DI, SI, BP, SP, BX, DX, CX, AX din stivă.  (Valoarea SP este ignorată, este descărcată, dar nu este scrisă în registrul SP).  Notă: această comandă funcționează numai pe procesoarele **80186** și mai sus!  Algoritm:   * POP DI * POP SI * POP BP * POP xx (valoarea SP este ignorată) * POP BX * POP DX * POP CX * POP AX  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| POPF | Fără operanzi | Obține registrul de steaguri din stivă.  Algoritm:   * flags = SS:[SP] (partea de sus a stivei) * SP = SP + 2  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | sunt descărcate din stivă | | | | | | |
| PUSH | REG  SREG SREG memorie imediată | Scrie o valoare pe 16 biți în stivă.  Notă: **PUSH immediate** funcționează numai pe procesoarele 80186 și mai sus!  Algoritm:   * SP = SP - 2 * SS:[SP] (partea de sus a stivei) = operand   Exemplu:  MOV AX, 1234h  PUSH AX  POP DX ; DX = 1234h  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| PUSHA | Fără operanzi | Stivuiește toate registrele de uz general: AX, CX, DX, BX, SP, BP, SI, DI. Folosește valoarea originală a registrului SP (înainte de a executa PUSHA).  Notă: Această comandă funcționează numai pe procesoarele 80186 și mai sus!  Algoritm:   * PUSH AX * PUSH CX * PUSH DX * PUSH BX * PUSH SP * PUSH BP * PUSH SI * PUSH DI  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| PUSHF | Fără operanzi | Scrie registrul de indicatori în stivă.  Algoritm:   * SP = SP - 2 * SS:[SP] (partea de sus a stivei) = steaguri  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| RCL | memorie, imediat  REG, imediat  memorie, CL  REG, CL | Deplasare (rotație) ciclică spre stânga prin intermediul unui transport. Numărul de rotații este setat în cel de-al doilea operand. Dacă **immediate** este mai mare decât unu, asamblorul generează mai multe comenzi **RCL xx, 1,** deoarece 8086 are cod mașină doar pentru această comandă (același principiu funcționează pentru toate comenzile de decalare/rotație).  Algoritm:  Scrieți bitul cel mai din stânga în flagul CF, deplasați toți biții spre stânga, scrieți valoarea flagului CF în bitul cel mai din dreapta (bitul 0).  Exemplu:  STC ; setează reportul (CF=1).  MOV AL, 1Ch ; AL = 00011100b  RCL AL, 1 ; AL = 00111001b, CF=0.  RET   |  |  | | --- | --- | | C | O | | r | r |   OF=0 în cazul în care primul operand își păstrează semnul inițial (+ sau -). |
| RCR | memorie, imediat  REG, imediat  memorie, CL  REG, CL | Deplasare (rotație) ciclică spre dreapta prin intermediul unui transport. Numărul de rotații este setat în cel de-al doilea operand.  Algoritm:  Scrieți bitul cel mai din dreapta (bitul 0) în flagul CF, deplasați toți biții spre dreapta, scrieți valoarea flagului CF în bitul cel mai din stânga.  Exemplu:  STC ; setează reportul (CF=1).  MOV AL, 1Ch ; AL = 00011100b  RCR AL, 1 ; AL = 10001110b, CF=0.  RET   |  |  | | --- | --- | | C | O | | r | r |   OF=0 în cazul în care primul operand își păstrează semnul inițial (+ sau -). |
| REP | echipa potrivită | Repetă următoarele comenzi MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW. Numărul de repetări este specificat în CX.  Algoritm:  check\_cx:  Dacă CX <> 0 atunci.   * executați comanda dorită * CX = CX - 1 * revenire la tag check\_cx   altfel   * ieșiți din ciclul REP  |  | | --- | | Z | | r | |
| REPE | echipa potrivită | Repetă următoarele comenzi CMPSB, CMPSW, SCASB, SCASW până când ZF = 1, dar nu mai mult de CX ori.  Algoritm:  check\_cx:  Dacă CX <> 0 atunci   * executați comanda dorită * CX = CX - 1 * Dacă ZF = 1, atunci:   + întoarcere la tag check\_cx   altfel   * + ieșiți din ciclul REPE   altfel   * ieșiți din ciclul REPE   Exemplu: vezi [**cmpsb.asm**](http://www.avprog.narod.ru/progs/Samples/cmpsb.asm) în directorul Samples.   |  | | --- | | Z | | r | |
| REPNE | echipa potrivită | Repetă următoarele comenzi CMPSB, CMPSW, SCASB, SCASW până când ZF = 0, dar nu mai mult de CX ori.  Algoritm:  check\_cx:  Dacă CX <> 0 atunci   * executați comanda dorită * CX = CX - 1 * Dacă ZF = 0, atunci:   + întoarcere la tag check\_cx   altfel   * + ieșiți din ciclul REPNE   altfel   * ieșiți din ciclul REPNE  |  | | --- | | Z | | r | |
| REPNZ | echipa potrivită | Repetă următoarele comenzi CMPSB, CMPSW, SCASB, SCASW până când ZF = 0, dar nu mai mult de CX ori.  Algoritm:  check\_cx:  Dacă CX <> 0 atunci   * executați comanda dorită * CX = CX - 1 * Dacă ZF = 0, atunci:   + revenire la tag check\_cx   altfel   * + ieșiți din ciclul REPNZ   altfel   * ieșiți din ciclul REPNZ  |  | | --- | | Z | | r | |
| REPZ | echipa potrivită | Repetă următoarele comenzi CMPSB, CMPSW, SCASB, SCASW până când ZF = 1, dar nu mai mult de CX ori.  Algoritm:  check\_cx:  Dacă CX <> 0 atunci   * executați comanda dorită * CX = CX - 1 * Dacă ZF = 1, atunci:   + revenire la tag check\_cx   altfel   * + ieșiți din ciclul REPZ   altfel   * ieșiți din ciclul REPZ  |  | | --- | | Z | | r | |
| RET | Fără operanzi sau un număr par | Întoarcerea de la o procedură din apropiere.  Algoritm:   * Preluare din stivă:   + IP * dacă există un operand: SP = SP + operand   Exemplu:  #make\_COM#  ORG 100h ; pentru fișierul COM.  CALL p1  ADD AX, 1  RET ; întoarcerea la sistemul de operare.  p1 PROC ; declarație de procedură.  MOV AX, 1234h  RET ; revenire la program.  p1 ENDP   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| RETF | Fără operanzi sau un număr par | Întoarcerea de la o altă procedură.  Algoritm:   * Preluare din stivă:   + IP   + CS * dacă există un operand: SP = SP + operand  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| ROL | memorie, imediat  REG, imediat  memorie, CL  REG, CL | Deplasare (rotație) ciclică spre stânga. Numărul de rotații este stabilit în cel de-al doilea operand.  Algoritm:  scrieți bitul cel mai din stânga în flagul CF, deplasați toți biții la stânga, scrieți flagul CF în bitul cel mai din dreapta.  Exemplu:  MOV AL, 1Ch ; AL = 00011100b  ROL AL, 1 ; AL = 00111000b, CF=0.  RET   |  |  | | --- | --- | | C | O | | r | r |   OF=0 în cazul în care primul operand își păstrează semnul inițial (+ sau -). |
| ROR | memorie, imediat  REG, imediat  memorie, CL  REG, CL | Deplasare (rotație) ciclică spre dreapta. Numărul de rotații este stabilit în cel de-al doilea operand.  Algoritm:  scrieți bitul cel mai din dreapta în flagul CF, deplasați toți biții la dreapta, scrieți flagul CF în bitul cel mai din stânga.  Exemplu:  MOV AL, 1Ch ; AL = 00011100b  ROR AL, 1 ; AL = 00001110b, CF=0.  RET   |  |  | | --- | --- | | C | O | | r | r |   OF=0 în cazul în care primul operand își păstrează semnul inițial (+ sau -). |
| SAHF | Fără operanzi | Scrieți date din registrul NA în cei 8 biți inferiori ai registrului de indicatori.  Algoritm:  registrul de steaguri = AH  NA Biți: 7 6 5 4 4 3 2 1 1 0  [SF] [ZF] [0] [AF] [0] [PF] [1] [CF]  biții 1, 3, 5 sunt rezervați.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | r | r | r | r | r | |
| SAL | memorie, imediat  REG, imediat  memorie, CL  REG, CL | Deplasare aritmetică spre stânga. Numărul de deplasări este scris în cel de-al doilea operand.  Algoritm:   * cel mai din stânga bit pentru a scrie în CF, decalează toți biții spre stânga, * scrieți un zero pe bitul cel mai din dreapta.   Exemplu:  MOV AL, 0E0h ; AL = 11100000b  SAL AL, 1 ; AL = 11000000b, CF=1.  RET   |  |  | | --- | --- | | C | O | | r | r |   OF=0 în cazul în care primul operand își păstrează semnul inițial (+ sau -). |
| SAR | memorie, imediat  REG, imediat  memorie, CL  REG, CL | Deplasare aritmetică spre dreapta. Numărul de deplasări este scris în cel de-al doilea operand.  Algoritm:   * bitul cel mai din dreapta pentru a scrie în CF, deplasați toți biții spre dreapta, * scrieți un zero pe bitul cel mai din dreapta. * Bitul de caracter care este inserat în poziția cea mai din stânga are aceeași valoare ca și cel de dinaintea decalării.   Exemplu:  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 în cazul în care primul operand își păstrează semnul inițial (+ sau -). |
| SBB | REG, memorie memorie, REG REG, REG memorie, imediată  REG, imediată | Scădere cu o sarcină.  Algoritm:  operand1 = operand1 - operand2 - CF  Exemplu:  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 | |
| SCASB | Fără operanzi | Compară octeți (caută octeți în șir): AL din ES:[DI].  Algoritm:   * ES:[DI] - AL * setează stegulețe în funcție de rezultat:  OF, SF, ZF, AF, PF, CF * Dacă DF = 0, atunci   + DI = DI + 1   altfel   * + DI = DI - 1  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | r | r | r | r | r | |
| SCASW | Fără operanzi | Compară cuvinte: AX de ES:[DI].  Algoritm:   * ES:[DI] - AX * setează stegulețe în funcție de rezultat:  OF, SF, ZF, AF, PF, CF * Dacă DF = 0, atunci   + DI = DI + 2   altfel   * + DI = DI - 2  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | r | r | r | r | r | |
| SHL | memorie, imediat  REG, imediat  memorie, CL  REG, CL | Deplasați-vă spre stânga. Numărul de deplasări este specificat în cel de-al doilea operand. Bitul de semn este tratat ca un bit de date normal.  Algoritm:   * Scrieți cel mai din stânga bit din CF, deplasați toți biții spre stânga. * Scrieți zero în bitul cel mai din dreapta.   Exemplu:  MOV AL, 11100000b  SHL AL, 1 ; AL = 11000000b, CF=1.  RET   |  |  | | --- | --- | | C | O | | r | r |   OF=0 în cazul în care primul operand își păstrează semnul inițial (+ sau -). |
| SHR | memorie, imediat  REG, imediat  memorie, CL  REG, CL | Deplasați-vă spre dreapta. Numărul de deplasări este specificat în cel de-al doilea operand. Bitul de semn este tratat ca un bit de date normal.  Algoritm:   * Scrieți cel mai din dreapta bit din CF, deplasați toți biții spre dreapta. * Scrieți zero în bitul cel mai din stânga.   Exemplu:  MOV AL, 00000111b  SHR AL, 1 ; AL = 00000011b, CF=1.  RET   |  |  | | --- | --- | | C | O | | r | r |   OF=0 în cazul în care primul operand își păstrează semnul inițial (+ sau -). |
| STC | Fără operanzi | Setează indicatorul de transfer (CF).  Algoritm:  CF = 1   |  | | --- | | C | | 1 | |
| STD | Fără operanzi | Setează indicatorul de direcție (DF). Valorile registrelor SI și DI sunt decrementate de comenzile: CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW.  Algoritm:  DF = 1   |  | | --- | | D | | 1 | |
| STI | Fără operanzi | Setează indicatorul de întrerupere. Aceasta activează întreruperile hardware.  Algoritm:  IF = 1   |  | | --- | | I | | 1 | |
| STOSB | Fără operanzi | Scrie un octet din AL în ES:[DI]. Schimbă SI.  Algoritm:   * ES:[DI] = AL * Dacă DF = 0, atunci   + DI = DI + 1   altfel   * + DI = DI - 1   Exemplu:  #make\_COM#  ORG 100h  LEA DI, a1  MOV AL, 12h  MOV CX, 5  REP STOSB  RET  a1 DB 5 dup(0)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | rămân neschimbate | | | | | | |
| STOSW | Fără operanzi | Scrie un cuvânt de la AX la ES:[DI]. Schimbă SI.  Algoritm:   * ES:[DI] = AX * Dacă DF = 0, atunci   + DI = DI + 2   altfel   * + DI = DI - 2   Exemplu:  #make\_COM#  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 | | rămân neschimbate | | | | | | |
| SUB | REG, memorie memorie, REG REG, REG memorie, imediată  REG, imediată | Scădere.  Algoritm:  operand1 = operand1 - operand2  Exemplu:  MOV AL, 5  SUB AL, 1 ; AL = 4  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | r | r | r | r | r | r | |
| TEST | REG, memorie memorie, REG REG, REG memorie, imediată  REG, imediată | ȘI logic între toți biții a doi operanzi. Nu modifică operandul rezultat, ci doar stegulețele sunt afectate. Sunt activate următoarele stegulețe: **ZF, SF, PF.**  Se aplică următoarele reguli:  1 ȘI 1 = 1 1  1 ȘI 0 = 0  ȘI 1 = 0 0  ȘI 0 = 0   Exemplu:  MOV AL, 00000101b  TEST AL, 1 ; ZF = 0.  TEST AL, 10b ; ZF = 1.  RET   |  |  |  |  |  | | --- | --- | --- | --- | --- | | C | Z | S | O | P | | 0 | r | r | 0 | r | |
| XCHG | REG, memorie memorie, REG REG, REG | O permutare a doi operanzi.  Algoritm:  operand1 < - > operand2  Exemplu:  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 | | rămân neschimbate | | | | | | |
| XLATB | Fără operanzi | Traduce un octet din tabel. Copiază un octet din memorie la DS:[BX + AL fără semn de semnătură] în registrul AL.  Algoritm:  AL = DS:[BX + unsigned AL].  Exemplu:  #make\_COM#  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 | | rămân neschimbate | | | | | | |
| XOR | REG, memorie memorie, REG REG, REG memorie, imediată  REG, imediată | Un XOR logic (Exclusive OR) între toți biții a doi operanzi. Rezultatul este scris în primul operand.  Se execută următoarele reguli:  1 XOR 1 = 0  1 XOR 0 = 1 0 XOR  1 = 1  0 XOR 0 = 0 XOR 0 = 0   Exemplu:  MOV AL, 00000111b  XOR AL, 00000010b ; AL = 00000101b  RET   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | Z | S | O | P | A | | 0 | r | r | 0 | r | ? | |