13 8086 - Setul de instrucțiuni

Sunt prezentate în detaliu instrucțiunile de bază ale familiei de microprocesoare Intel. În general, se poate utiliza orice mod de adresare a memoriei; unde este cazul, se va specifica tipul de adresare interzis.

Setul de instrucțiuni conține 7 clase:

- **instrucțiuni de transfer,** care deplasează date între memorie şi registrele interne sau între porturile de intrare-ieşire şi registrele interne, fără a executa nici un fel de prelucrare a datelor;
- **instrucțiuni aritmetice**, care prelucrează date în format numeric prin aplicarea operațiilor de adunare, scădere, înmulțire, împărțire;
- **instrucțiuni logice**, care prelucreză date la nivel de bit;
- **instrucțiuni pentru şiruri**, specifice operațiilor cu date alfanumerice (caractere);
- **instrucțiuni pentru controlul programului** (salturi şi apeluri de proceduri);
- **instrucțiuni specifice întreruperilor** hard şi soft;
- **instrucțiuni pentru controlul procesorului**.

13.1 Instrucțiuni de transfer

Instrucțiunile din această clasă produc copierea unui octet sau cuvânt de la o adresă sursă (*sursa*) la o adresă destinație (*dest*), fără a afecta data sursă. Destinația poate fi registru, locație de memorie sau port de ieşire iar sursa poate fi registru, locație de memorie, port de intrare sau date imediate specificate în instrucțiune.

În specificarea destinației şi sursei vor fi utilizate notațiile originale, introduse de *Intel*:

segment : offset - adresă completă, prin specificarea registrului segment implicat în calculul adresei şi prin specificarea sursei adresei efective ;

(reg) - conținutul registrului "reg";

((reg)) - conținutul locației de memorie care are adresa efectivă (offset-ul) în registrul "reg".

De exemplu, (BX) înseamnă "conținutul registrului BX", iar ((BX)) are semnificația "conținutul locației de memorie cu adresa efectivă în BX" (registrul segment implicat în adresare este implicit DS în acest caz).

De asemenea, ES : ((BX)) înseamnă "conținutul locației de memorie adresate cu ES şi BX".

Cu excepția instrucțiunilor de lucru cu stiva, PUSH, POP, sursa şi destinația nu pot fi simultan locații de memorie.

Cu excepția instrucțiunilor SAHF şi POPF, nici o instrucțiune de transfer nu modifică bistabilii de condiții din registrul F.

13.1.1. Instrucțiuni de transfer generale: *MOV, PUSH, POP, XCHG*

MOV *dest, sursa*	(*Move data - Transferă date*)
descriere formală:	dest ← sursa
descriere:	se copiază operandul sursă în operandul destinație; operanzii pot fi de tip byte sau word
operanzi:	reg, reg reg, mem mem, reg reg, data mem, data
fanioane afectate:	nici unul

Observații:

- sursa şi destinația nu pot fi simultan operanzi în memorie;
- nu pot fi utilizate ca 'reg' registrele F şi IP;
- operazii trebuie să aibă aceeaşi dimensiune;
- registrul CS nu poate apărea ca destinație;
- nu pot fi transferate date imediate într-un registru de segment (8086);
- operanzii nu pot fi simultan registre de segment (8086).

Exemple:

```
MOV  BX, CX                  ;transfer pe 16 biți reg - reg
MOV  CL,  AL                 ;transfer pe 8 biți  reg - reg
MOV  [1200], AX              ;transfer pe 16 biți mem - reg
MOV  byte  ptr  [BX+200], 9A ;transfer imediat în memorie
```

Ultima instrucțiune utilizează operatorul *ptr* pentru evitarea ambiguității privind dimensiunea operandului transferat în memorie:

MOV [BX+200], 9A

poate fi interpretată prin " transferă octetul 9A la adresa DS:BX+200" sau "transferă cuvântul 009A la adresa DS:[BX+200], caz în care în locația DS:[BX+201] se pune 00 H.

Forma *byte ptr* precizează că se transferă un octet.

PUSH *sursa ;*	(*Push data - Salvează date în stivă)*
descriere formală:	(SP) ← (SP) - 2 SS: ((SP)+1) ← sursa (high) SS: ((SP)) ← sursa (low)
descriere:	se decrementează registrul SP cu 2 şi se copiază operandul sursă (word) în memoria stivă, cu octetul mai semnificativ la adresa mai mare, în vârful stivei;
operanzi:	registru general de 16 biţi, registru segment; locaţie de memorie de 16 biţi;
fanioane afectate:	nici unul

Exemple:

```
PUSH      CX              ; salvează în stivă conținutul lui CX.
PUSH  [BX]                ; salvează locația de 16 biți
                          ; cu offset - ul în BX
PUSH  ES: [BX] [SI + 20]  ; salvează locația de memorie de 16 biți
                          ; cu offset-ul dat de BX + SI + 20,
                          ; aflată în segmentul cu baza în ES.
PUSH  AL                  ; instrucțiune incorectă: operand pe un octet
```

POP *dest*	(*Pop data - Restaurează date din stivă)*
descriere formală:	dest (high) ← SS: ((SP)+1) dest (low) ← SS: ((SP)) (SP) ← (SP) + 2
descriere:	se copiază octeţii din stivă de la adresele (SP)+1 şi (SP) în destinaţie şi apoi se incrementează registrul SP cu 2.
operanzi:	registru general de 16 biţi, registru segment; locaţie de memorie de 16 biţi;
fanioane afectate:	nici unul

Exemple:

```
POP  BX          ; copiază cuvântul din vârful stivei în BX
POP  DS          ; copiază cuvântul din vârful stivei în DS
POP  ES : [DI]   ; copiază cuvântul din vârful stivei în memorie, în
                 ; segmentul de date suplimentar, offset-ul în DI
```

```
POP  [BP+7]          ; copiază cuvântul din vârful stivei, tot în
                     ; segmentul stivă (implicit), offset-ul dat de BP+7.
POP  SS : [BX+9]     ; copiază cuvântul din vârful stivei tot în segmentul
                     ; stivă offset - ul fiind dat de BX+9.
POP  AL              ; instrucțiune incorectă: operand pe un octet .
POP  CS              ; instrucțiune incorectă: destinație CS.
```

Observaţii:

1. După o secvență de salvare în stivă (cu PUSH), secvența de refacere trebuie să conțină destinațiile în ordine inversă. Dacă secvența de salvare a fost:

```
PUSH  AX
PUSH  DX
PUSH  [BP]
```

atunci secvența de refacere trebuie să fie:

```
POP  [BP]
POP  DX
POP  AX
```

2. De regulă, numărul operațiilor PUSH trebuie să coincidă cu numărul operațiilor POP.

3. Instrucțiunile PUSH şi POP mai pot fi utilizate pentru transfer indirect între registre sau locații de memorie:

```
PUSH  DX
POP   ES
```

Se copiază prin intermediul stivei conținutul registrului DX în ES, lasă indicatorul de stivă SP neschimbat iar conținutul anterior al registrului ES se pierde.

```
PUSH   AX       ; salvează AX
PUSH   CX       ; salvează CX
POP     AX      ; conținutul lui CX se transferă în AX
POP     CX      ; conținutul inițial al lui AX se transferă în CX
```

XCHG *dest, sursa*	(*Exchange - Schimbă reciproc)*
descriere formală:	dest ← sursa sursa ← dest.
descriere:	se transferă conținutul sursei în destinație şi reciproc; registrele segment nu pot apărea ca operanzi; cel puțin un operand trebuie să fie registru.
operanzi:	reg. - reg., reg. - mem., mem. - reg.
fanioane afectate:	nici unul

Exemple:

```
XCHG  CL, AL
XCHG  CX, DX
XCHG  DS : [DX], AX
```

Exemple de instrucţiuni incorecte:

```
XCHG  AL, BX          ; operanzi de lungime diferită
ECHG  ES, AX          ; registrul segment ES apare ca operand
```

13.1.2. Instrucțiuni de transfer specifice acumulatorului: *IN, OUT*

IN *AL, port* ***IN*** *AX, DX*	(*Input data - Citeşte date de la un port de intrare)*
descriere formală:	AL ← data 8 AX ← data 16
descriere:	se transferă în AL conținutul unui port de intrare de 8 biți specificat printr-o adresă de un octet (00 . . FF), sau se transferă în AX conținutul unui port de intrare de 16 biți cu adresa specificată în DX.
operanzi:	AL, data 8 AX, data 16
fanioane afectate:	nici unul

Procesoarele i286, 386, 486 şi Pentium, acceptă orice registru în locul lui AL sau AX.

OUT *port, AL* ***OUT*** *DX, AX*	(*Output data - Scrie date într-un un port de ieşire)*
descriere formală:	port ← AL port ← AX
descriere:	se transferă conținutul registrului AL la portul de ieşire de 8 biți specificat printr-o adresă de un octet (00 . . FF), sau se transferă conținutul registrului AX la portul de ieşire de 16 biți specificat prin adresa de 16 biți ce se află în DX.
operanzi:	data 8, AL DX, AX
fanioane afectate:	nici unul

Variantele noi de procesoare, (i286, 386, 486, Pentium) acceptă orice registru în locul lui AL sau AX.

Instrucțiunile IN şi OUT realizează interacțiunea procesorului cu dispozitivele periferice (imprimantă, tastatură, interfață de proces etc.).

Dacă porturile sunt organizate ca locații de memorie, ele pot fi adresate în toate modurile de adresare a memoriei; în acest caz, schimbul de date cu procesorul se poate face cu instrucțiunea MOV. Asemenea sisteme de intrări/ieşiri se numesc de tip *memory - mapped* (intrări/ieşiri organizate ca locații de memorie).

Exemple:

Să considerăm că un echipament periferic necesită un port de stare şi un port de date, ambele pe 8 biți. Într-un sistem de intrări / ieşiri obişnuit, vor exista două porturi de intrare, de exemplu, 0E7H şi 0E8H pentru transfer de date de la periferic la procesor:

```
IN  AL, 0E7H        ; Citire stare
IN  AL, 0E8H        ; Citire date
```

Într-un sistem de tip *memory - mapped*, vor exista două adrese, de regulă consecutive, pentru transfer de date la procesor:

```
MOV  ES, 0D700H     ; Se încarcă adresa în ES
MOV  AL,  ES: [0]   ; Citire stare de la 0D700
MOV  AL,  ES: [1]   ; Citire date  de la 0D701
```

Pentru scriere date în porturile de ieşire:

```
MOV AX, 27FFH           ; Se încarcă data în acc.
OUT  0E8H, AL       ; Scrie FF în portul E8, de 8 biți
MOV  DX,  0789H     ; Se încarcă o adresă de 16 biți în DX
OUT   DX,  AX       ; Scrie 27FF în portul 789 de 16 biți
MOV  ES, 0D800H     ; Se încarcă adresa în ES
MOV  ES: [2], AL        ; Scriere date în port de 8 biți
MOV  ES: [3], AX        ; Scriere date în port de 16 biți
```

XLAT	(*Translate - Translatează)*
descriere formală:	(AL) ← DS: ((BX) + (AL))
descriere:	Se transferă în AL conținutul locației (de 8 biți) de la adresa (BX) + (AL). Instrucțiunea este folosită împreună cu tabele de translatare existente în memorie, utile pentru conversia unor tipuri de date dintr-un cod în altul.
operanzi:	AL, data(8)
fanioane afectate:	nici unul

Exemplu: pentru conversia unei valori numerice cuprinse între 0 şi 15 la cifra hexazecimală corespunzătoare se poate folosi secvența:

```
.data
      TAB       DB   '0123456789ABCDEF'
.code
      MOV  AL, 10                ; se încarcă în AL valoarea numerică 10
      MOV  BX, OFFSET  TAB       ; iar în BX adresa tabelei
      XLAT                       ; în AL se obține 0AH
```

13.1.3. Instrucțiuni de transfer specifice adreselor: *LEA, LDS, LES*

Aceste instrucțiuni transferă o adresă efectivă într-un registru general sau o adresă completă de 32 de biți într-o pereche de registre.

LEA *reg, sursa*	(*Load Effective Address - Încarcă adresa efectivă)*
descriere formală:	registru ← adresa efectivă (*offset*) a operandului sursă
descriere:	Se copiază adresa efectivă a operandului sursă (operand în memorie, specificat printr-un mod oarecare de adresare) în registrul general specificat. Instrucțiunea LEA se foloseşte pentru încărcarea registrelor de bază sau de segment cu adresele efective ale unor operanzi din memorie, în vederea unor adresări ulterioare
operanzi:	registru de 16 biți, offset - ul unui operand din memorie
fanioane afectate:	nici unul

Exemple:

```
LEA  BX,  ALFA
LEA  DI,  ALFA  [BX] [SI]
```

Un calcul similar al adresei efective se obține şi cu operatorul OFFSET utilizat cu instrucțiunea MOV, calcul care se face însă la asamblare. Secvența:

```
.data
      vector            dw  10,  20,  40,  60,  80
.code
      LEA    BX,  VECTOR    ; se încarcă în BX offset - ul primului
      MOV  SI,    4         ; element al vectorului (10)
      MOV  AX,   [BX] [SI]
```

va încărca în AX al treilea element al tabloului VECTOR.

LDS *reg, sursa* ***LES*** *reg, sursa*	(*Load Data Segment - Încarcă DS)* (*Load Extra Segment - Încarcă ES)*
descriere formală:	(reg.) ← (sursa) low -16 biți (DS) sau (ES) ← (sursa) high - 16 biți
descriere:	Se încarcă perechea de registre DS : reg sau ES : reg cu o adresă completă de 32 de biți. Instrucțiunile LDS, LES se folosesc de regulă cu directiva *Define Double Word* pentru sursă.
operanzi:	reg. este registru general de 16 biți, sursa este un operand de tip double-word, aflat în memorie, care conține o adresă completă de 32 de biți
fanioane afectate:	nici unul

Exemplu:

```
.data
        x           db    00
        y           db    FF
        adr_x       dd    x       ; variabilele adr_x şi adr_y, care conțin
        adr_y       dd    y       ; adresele complete ale variabilelor x, y
.code
LDS  SI,  adr_x                   ; se încarcă în DS : SI şi ES : DI
LES  DI,  adr_y
MOV  byte ptr  [SI],  77          ; se accesează variabilele x, y folosind
MOV  byte ptr  [DI],  99          ; adresarea indexată, pentru inițializare.
```

13.1.4. Instrucțiuni de transfer specifice indicatorilor de condiții:

LAHF, SAHF, PUSHF, POPF

LAHF ***SAHF***	(*Load AH with Flags - Încarcă AH cu indicatorii din F)* (*Store AH into Flags - Depune AH în indicatori - F)*
descriere formală:	AH ← Flags 0 - 7 pentru LAHF Flags 0 - 7 ← AH pentru SAHF
descriere:	Se încarcă registrul AH cu octetul inferior al registrului indicatorilor de condiții (F) - pentru LAHF Se încarcă în octetul inferior al registrului indicatorilor de condiții (F), conținutul registrului AH
operanzi:	op. impliciți : registrele AH şi F
fanioane afectate:	nici unul la LAHF, toate la SAHF (se schimbă F0 - F7)

PUSHF ***POPF***	(*Push Flags - Salvează registrul F în stivă)* (*Pop Flags - Reface registrul F din stivă)*
descriere formală:	PUSHF: (SP) ← (SP) - 2 SS: ((SP)+1, (SP)) ← F POPF: F ← SS: ((SP)+1, (SP)) (SP) ← (SP) + 2
descriere:	Se plasează registrul F în vârful stivei (PUSHF) Se reface registrul F din vârful stivei (POPF)
operanzi:	nu are operanzi
fanioane afectate:	nici unul la PUSH, toate la POPF (se schimbă F)

Observații:

Instrucțiunile LAHF şi SAHF transferă numai partea mai puțin semnificativă a registrului indicatorilor de condiții (F0 - F7).

Partea mai semnificativă a registrului F poate fi citită şi modificată cu instrucțiunile POPF şi PUSHF:

```
PUSHF
POP  AX          ; încarcă în AX  registrul de flaguri (F0 -F15)
. . . . . .      ; se pot citi şi modifica toți indicatorii !
PUSH  AX         ; salvează AX în stivă
POPF             ; transferă AX (din stivă) în registrul de flaguri
```

Cu excepția instrucțiunilor explicite SAHF şi POPF, nici o instrucțiune de transfer nu modifică indicatorii de condiții.

13.2 Instrucțiuni aritmetice

Realizează operațiile aritmetice elementare (adunare, scădere, înmulțire, împărțire) între doi operanzi, rezultatul înlocuind, de regulă, primul operand.

Sunt afectați indicatorii de condiții SF, ZF, AF, PF, CF. OF, numiți din acest motiv indicatori aritmetici.

La fiecare instrucțiune se specifică indicatorii afectați, cei neafectați rămân la valoarea veche.

Există situații în care unii indicatori au valori nedefinite.

13.2.1. Instrucțiuni de adunare şi scădere

ADD dest, sursa	(*Add - Adună)*
descriere formală:	(dest) ← (dest) + (sursa)
descriere:	Se face adunarea celor doi operanzi, rezultatul se obține în destinație; valoarea op. sursă nu se modifică.
operanzi:	reg, reg (de 8 sau 16 biți) reg, mem mem, reg reg, data mem, data (operanzii au aceeaşi dimensiune)
fanioane afectate:	AF, CF, PF, SF, ZF, OF; (toate)

Observații:

1. Cei doi operanzi nu pot fi simultan locații de memorie;
2. Cei doi operanzi trebuie să aibă aceeaşi dimensiune (8 sau 16 biți);
3. În caz de ambiguitate în ceea ce priveşte dimensiunea, se utilizează operatorul ***ptr,*** ca în exemplele de mai jos.

```
ADD     AX, BX              ; adunare pe 16 biți
ADD     AX, 1               ; adunare pe 16 biți
ADD     AL, 1               ; adunare pe 8 biți
ADD     word ptr [DI], -1   ; adunare pe 16 biți, destinație memorie
                            ; sursa imediată
```

ADC dest, sursa	(*Add with Carry- Adună cu transport)*
descriere formală:	(dest) ← (dest) + (sursa) + (CF)
descriere:	Se face adunarea celor doi operanzi şi apoi se adună şi valoarea lui CF (0 sau 1), rezultatul se obține în destinație; valoarea operandului sursă nu se modifică.
operanzi:	reg, reg (de 8 sau 16 biți) reg, mem mem, reg reg, data mem, data (operanzii au aceeaşi dimensiune)
fanioane afectate:	AF, CF, PF, SF, ZF, OF; (toate)

Adunarea cu carry, ADC, se foloseşte în cazul adunării de operanzi cu lungime mai mare de 2 octeți, caz în care poate să apară transport intermediar la adunările parțiale pe 8 sau 16 biți.

În exemplul următor se adună doi operanzi pe 4 octeți fiecare:

```
. data
      ALFA      dd          123A567FH
      BETA      dd          EED2E248H
      REZ             dd          ?
.code
      MOV       AX,   word ptr ALFA          ; cuvântul inferior în AX
      ADD       AX,   word ptr BETA          ; adunare pe 16 biți, CF = 1
      MOV       word ptr REZ,  AX            ; stocare rezultat
      MOV       AX,   word ptr [ALFA + 2 ]   ; cuvântul superior în AX
      ADC       AX,   word ptr [BETA + 2 ]   ; adunare pe 16 biți cu carry
      MOV       word ptr [REZ + 2], AX  ; stocare rezultat final.
```

INC dest	(*Increment - Incrementează = adună 1)*
descriere formală:	(dest) ← (dest) + 1
descriere:	Conținutul destinației creşte cu 1.
operanzi:	registru sau operand în memorie de tip octet sau cuvânt.
fanioane afectate:	AF, PF, SF, ZF, OF; (fără CF)

DAA	(*Decimal Adjust for Addition - Corectie zecimală a acumulatorului, după adunare)*
descriere formală:	dacă (AL_{0-3}) > 9 sau AF = 1, atunci: (AL) ← (AL) + 6, AF ← 1, dacă (AL_{4-7}) > 9 sau CF = 1, atunci: (AL) ← (AL) + 60H, CF ← 1.
descriere:	-se efectuează corecția rezultatului din acumulatorul AL după adunare cu operanzi BCD împachetați (4 biți / digit); -corecția se face prin adunare a valorii 6 la AL, dacă a avut loc o depăşire a valorii maxime a cifrei BCD - MPS şi a valorii 60H dacă a avut loc o depăşire a valorii maxime a cifrei BCD - MS (mai semnificative); - în ambele cazuri se poziționează indicatorii de transport pentru a marca depăşirea.
operanzi:	instrucțiunea are operand implicit: AL;
fanioane afectate:	CF, AF, PF, SF, ZF; OF este nedefinit.

Exemplu: considerăm adunarea valorilor BCD 55 şi 27, care se reprezintă prin octeții 55H şi 27H. După adunarea lor se obține rezultatul 7CH, care este corect ca rezultat în cod Hexazecimal, dar incorect în BCD. Operația de coreție (se adună 6 la AL) conduce la rezultatul 82H, corect în BCD: 55 + 27 = 82.

```
.data
        X_BCD     db    55H
        Y_BCD     db    27H
        REZ       db    ?
.code
        MOV       AL,   X_BCD
        ADD       AL,   Y_BCD
        DAA
        MOV       REZ, AL
```

AAA	(*ASCII Adjust for Addition - Corectie ASCII a acumulatorului)*
descriere formală:	dacă (AL_{0-3}) > 9 sau AF = 1, atunci: (AL) ← (AL) + 6 (AH) ← (AH) + 1 AF ← 1, CF ← 1, (AL) ← (AL) & OFH
descriere:	-se efectuează corecția rezultatului din acumulatorul AX după adunare cu operanzi BCD despachetați (8 biți / digit); -corecția se face prin adunare a valorii 6 la AL, dar se incrementează şi AH care stochează cifra mai semnificativă
operanzi:	instrucțiunea are operanzi impliciți: AL, AH;
fanioane afectate:	AF, CF; restul sunt nedefinite;

Exemplu:

Considerăm că AX şi BX conțin 0309H şi 0104H, adică 39 şi 14 în BCD; după adunare se obține 040DH - incorect; instrucțiunea AAA corectează rezultatul şi se obține 0503H, corect.

```
MOV  AX,  0309H
MOV  BX,  0104H
ADD   AX,  BX          ; AX = 040DH
AAA                    ; AX = 0503H
```

SUB dest, sursa	(*Subtract - Scade)*
descriere formală:	(dest) ← (dest) - (sursa)
descriere:	Se face scăderea celor doi operanzi, rezultatul se obține în destinație; valoarea op. sursă nu se modifică.
operanzi:	reg, reg (de 8 sau 16 biți) reg, mem mem, reg reg, data mem, data (operanzii au aceeaşi dimensiune)
fanioane afectate:	AF, CF, PF, SF, ZF, OF; (toate)

Observații:

Scăderea poate fi interpretată ca o adunare a operandului destinație cu complementul față de 2 al sursei, dar cu inversarea rolului lui CF, adică dacă la această "adunare " apare transport, CF = 0 şi dacă nu apare transport, CF = 1. Exemplu:

```
MOV  AL, 1
SUB  AL, 05EH    ; rezultatul este 0A3H, există împrumut, deci CF=1
```

Dacă luăm complementul față de 2 al sursei obținem 0A2H care adunat cu destinația = 1 conduce la 0A3H, fără transport, deci CF = 1 conform convenției; în caz de ambiguitate privind lungimea operandului se utilizează operatorul ***ptr ;*** exemple:

```
SUB word ptr [DI], 3     ;destinație în memorie, sursa imediată
SUB [DI], 3              ;este ambiguă: se scade octet sau cuvânt?
SUB [DI], 3333H          ;se scade cuvânt.
```

SBB dest, sursa	(*Subtract with Borrow- Scade cu împrumut)*
descriere formală:	(dest) ← (dest) - (sursa) - (CF)
descriere:	Se face scăderea celor doi operanzi şi apoi se scade şi valoarea lui CF (0 sau 1), rezultatul se obține în destinație; valoarea operandului sursă nu se modifică.
operanzi:	reg, reg (de 8 sau 16 biți) reg, mem mem, reg reg, data mem, data (operanzii au aceeaşi dimensiune)
fanioane afectate:	AF, CF, PF, SF, ZF, OF; (toate)

Observaţie:

Instrucţiunea SBB se utilizează la scăderi de operanzi pe mai mult de doi octeţi:

```
.data
      ALFA  dd   145A789FH
      BETA  dd   92457ABCH
      REZ    dd  ?
.code
      MOV  AX,  word ptr ALFA
      SUB   AX,  word ptr  BETA        ;aici poate să apară împrumut!
      MOV  word ptr REZ,  AX
      MOV  AX,  word ptr  [ALFA+2]
      SBB   AX,  word ptr  [BETA+2]       ; se ia în consideraţie
                                          ; împrumutul precedent
      MOV  word ptr [REZ+2], AX
```

DEC dest	(*Decrement - Decrementează = scade 1)*
descriere formală:	(dest) ← (dest) - 1
descriere:	Se face scăderea unităţii din destinaţie
operanzi:	registru sau operand în memorie de tip octet sau cuvânt.
fanioane afectate:	AF, PF, SF, ZF, OF; (fără CF)

NEG dest	(*Negate - Schimbă semnul)*
descriere formală:	(dest) ← 0 - (dest)
descriere:	Se face schimbarea semnului operandului
operanzi:	registru sau operand în memorie de tip octet sau cuvânt.
fanioane afectate:	AF, CF, PF, SF, ZF, OF; (toate)

Observaţie:

Schimbarea semnului poate duce uneori la aceeaşi valoare în cazul depăşirii domeniului admisibil:

```
MOV  AL,  -128
NEG  AL
```

va lăsa AL neschimbat (80H), deoarece -128 şi 128 au aceeaşi reprezentare internă.

CMP dest, sursa	(*Compare - Compară)*
descriere formală:	(dest) - (sursa)
descriere:	Se face simularea unei scăderi, fără a se genera rezultat; efectul se obține numai în bistabilii de condiții. (d>s, d=s, d<s,); d, s nu se schimbă.
operanzi:	registru sau operand în memorie de tip octet sau cuvânt - ca la ADD sau SUB
fanioane afectate:	AF, CF, PF, SF, ZF, OF; (toate)

Instrucțiunea se folosește pentru testarea unei condiții de salt.

- Dacă d>s, rezultă ZF=0, CF=0;
- Dacă d=s, rezultă ZF=1, CF=0;
- Dacă d<s, rezultă ZF=0, CF=1.

DAS	(*Decimal Adjust for Subtraction - Corectie zecimală după scădere)*
descriere formală:	dacă (AL_{0-3}) > 9 sau AF = 1, atunci: (AL) ← (AL) - 6 AF ← 1 dacă (AL_{4-7}) > 9 sau CF = 1, atunci: (AL) ← (AL) - 60H CF ← 1
descriere:	-se efectuează corecția rezultatului din acumulatorul AL după adunare cu operanzi BCD împachetați (4 biți / digit); -corecția se face prin scădere a valorii 6 sau 60H sau 66H din AL, şi marcarea împrumutului în AF sau CF; analogie cu DAA;
operanzi:	instructiunea are operand implicit: AL;
fanioane afectate:	AF, CF, PF, SF, ZF; OF este nedefinit;

Exemplu:

```
.code
        MOV  AL, 52H
        SUB  AL,  24H          ; AL = 2EH
        DAS                    ; AL = 28H, rezultat corect: 52-24=28.
```

AAS	(*ASCII Adjust for Subtraction- Corectie ASCII a acumulatorului după scădere)*
descriere formală:	dacă (AL_{0-3}) > 9 sau AF = 1, atunci: (AL) ← (AL) - 6 (AH) ← (AH) - 1 AF ← 1, CF ← 1, (AL) ← (AL) & OFH
descriere:	-se efectuează corecţia rezultatului din acumulatorul AX după scădere de operanzi BCD despachetaţi (8 biţi / digit); -corecţia se face prin scăderea valorii 6 din AL, dar se decrementează şi AH care stochează cifra mai semnificativă în cod BCD; analogie cu AAA
operanzi:	instrucţiunea are operanzi impliciţi: AL, AH;
fanioane afectate:	AF, CF; restul sunt nedefinite;

13.2.2. Instrucţiuni de înmulţire (CBW, CWD, MUL, IMUL, AAM)

Operaţiile de înmulţire se realizează între acumulator şi un al doilea operand; rezultatul operaţiei este pe 16 sau 32 biţi: dacă operanzii sunt octeţi, rezultatul este pe 16 biţi iar dacă operanzii sunt cuvinte, rezultatul este pe 32 de biţi. Se folosesc următoarele noţiuni definite diferit la operaţiile pe 8 sau 16 biţi:

	operanzi pe 8 biţi	operanzi pe 16 biţi
acumulator	AL	AX
acumulator extins	AX	DX:AX
extensia acumulatorului	AH	DX
extensia de semn a acum.	AH	DX

Observaţie:

Prin instrucţiunile CBW, CWD, se obţin extensiile de semn ale acumulatorului în acumulator extins.

CBW	(*Convert Byte to Word - Conversie octet la cuvânt)*

descriere formală:	dacă (AL_7) = 0 , atunci: (AH) ← 00H altfel, (AH) ← FFH
descriere:	-se extinde bitul de semn din AL la întreg registrul AH; operația este echivalentă cu reprezentarea lui AL în complement față de 2, pe un număr dublu de biți. De exemplu, dacă AL = -3 (0FDH), instrucțiunea va forța în AX valoarea 0FFFDH, care este chiar reprezentarea în complement față de 2 a valorii -3.
operanzi:	instructiunea are operanzi impliciți: AL, AH;
fanioane afectate:	nici unul;

CWD	(*Convert Word to DoubleWord - Conversie cuvânt la dublu cuvânt)*
descriere formală:	dacă (AX_{15}) = 0 , atunci: (DX) ←0000H altfel, (DX) ← FFFFH
descriere:	-se extinde bitul de semn din AX la întreg registrul DX; operația este echivalentă cu reprezentarea lui AX în complement față de 2, pe un număr de 32 biți.
operanzi:	instructiunea are operanzi impliciți: AX, DX;
fanioane afectate:	nici unul;

MUL sursa	(*Multiply - Inmulţeşte fără semn)*
descriere formală:	(AH:AL) ← (AL) * (sursa) ; (înmul. pe 8 biți) (DX:AX) ← (AX) * (sursa); (înmul. pe 16 biți)
descriere:	Se face înmulțirea operandului din acum. cu operandul sursă, rezultatul (pe un număr dublu de biți) fiind stocat în acum. extins.
operanzi:	sursa este registru sau locație de memorie de 8 sau 16 biți; variantele superioare de microprocesoare permit ca sursa să fie şi dată imediată.
fanioane afectate:	CF, OF; (restul nedefinite)

```
.data
      ALFA      db    10H
      BETA      dw    200H
```

```
.code
        MOV  AL,  10H
        MUL  ALFA       ; (AX) = (AL ) *  ALFA
        MOV  AX,  20H
        MUL  BETA       ;  (DX:AX) = (AX)  *  BETA
        MOV  AX,  100H
        MOV  BX,  20H
        MUL  BX                 ;  (DX:AX) = (AX)  *  (BX)
```

Observație: Dacă un operand este de tip byte iar celălalt de tip word, se face conversia celui de tip byte la word, deci în octetul superior se pune 00H cu MOV.

Instrucțiunea MUL nu poate conduce la depășiri. Dacă operanzii au cele mai mari valori posibile pe 8 biți, se obține:

$$(2^8 - 1) \cdot (2^8 - 1) = 2^{16} - 2^9 + 1 < 2^{16} - 1 \quad ,$$

numărul maxim reprezentabil pe 16 biți fără semn, iar dacă operanzii au cele mai mari valori posibile pe 16 biți, se obține:

$$(2^{16} - 1) \cdot (2^{16} - 1) = 2^{32} - 2^{17} + 1 < 2^{32} - 1,$$

numărul maxim reprezentabil pe 32 de biți fără semn.

IMUL sursa	*(Integer Multiply - Inmulțește cu semn)*
descriere formală:	(AH:AL) ← (AL) * (sursa) ; (înmul. pe 8 biți) (DX:AX) ← (AX) * (sursa); (înmul. pe 16 biți)
descriere:	Se face înmulțirea operandului din acum. cu operandul sursă, rezultatul (pe un număr dublu de biți) fiind stocat în acumulatorul extins.
operanzi:	operanzi cu semn: sursa este registru sau locație de memorie de 8 sau 16 biți; variantele evoluate de microprocesoare permit ca sursa să fie şi dată imediată.
fanioane afectate:	CF, OF; (restul nedefinite)

Observație: Dacă extensia acumulatorului este extensie de semn, atunci CF şi OF se poziționează în 0; altfel devin 1.

Nu pot apărea depășiri privind dimensiunea rezultatului.

Dacă sunt necesare conversii de la byte la word sau de la word la doubleword, se vor folosi instrucțiunile de conversie pentru operanzi cu semn CBW, CWD. Exemplu:

```
.data
        ALFA        db    -103
        BETA        dw    -137
        REZ         dd    ?
.code
```

```
MOV  AL, ALFA
CBW                              ; conversie la word
IMUL  BETA                       ; înmulțire cu semn
MOV  word ptr REZ,  AX           ; memorarea rezultatului de
MOV  word ptr [REZ+2], DX        ; 32 biți cu p.m.s. la adrese mari.
```

AAM	(*ASCII Adjust for Multiply - Corectie ASCII a acumulatorului după înmulțire)*
descriere formală:	(AH) ← (AL) DIV 10 (împărțire întreagă) (AL) ← (AL) MOD 10 (restul)
descriere:	-se efectuează corecția rezultatului din acumulatorul AX după o înmulțire pe 8 biți de operanzi BCD despachetați (8 biți / digit); -corecția se face prin împărțirea valorii din AL la 10 ; restul rămâne în AL iar câtul se transferă în AH.
operanzi:	instrucțiunea are operanzi impliciți: AL, AH;
fanioane afectate:	PF, SF, ZF ; restul sunt nedefinite;

Exemplu:

```
MOV  AL,  5
MOV  CL,  9
MUL  CL
AAM
```

După înmulțire AX va conține valoarea 2DH (45 Z).

Corecția prin AAM conduce la: 45 : 10 = 4 rest 5, deci AH = 4, AL = 5 şi ca urmare, AX = 0405H, adică reprezentarea BCD tip despachetat pentru valoarea zecimală 45.

13.2.3. Instrucțiuni de împărțire (DIV, IDIV, AAD)

Împărțirea se face în condițiile în care deîmpărțitul este de lungime dublă față de împărțitor, iar câtul şi restul sunt de aceeaşi lungime cu împărțitorul.

DIV sursa	(*Divide - Împarte fără semn)*
descriere formală:	(AL) ← (AX) / (sursa) ; (sursa pe 8 biți) (AH) ← (AX) mod (sursa) ; (restul) (AX) ← (DX:AX) / (sursa) ; (sursa pe 16 biți) (DX) ← (DX:AX) mod (sursa) ; (restul)
descriere:	Se face împărțirea operandului din acum. cu operandul sursă, rezultatul (de dimensiune egală cu sursa) fiind stocat în acum. sau acum. extins.

operanzi:	sursa este registru sau locaţie de memorie de 8 sau 16 biţi; variantele superioare de microprocesoare permit ca sursa să fie şi dată imediată.
fanioane afectate:	toate flagurile sunt nedefinite.

Observaţie: Împărţirea poate duce la depăşiri, deoarece se poate obţine câtul mai mare decât valoarea maximă reprezentabilă pe 8 biţi, respectiv 16 biţi sau dacă împărţitorul este 0. Rezultatele sunt nedefinite şi se generează o întrerupere de tip soft pe nivelul 0 (*Divide Overflow* - Depăşire la împărţire).

Exemplu:

```
MOV  AX, 1000
MOV  CL,  3
DIV    CL
```

Câtul ar trebui să fie 333 iar restul 1; deoarece 333 nu se poate reprezenta pe 8 biţi, se generează întrerupere care de obicei opreşte programul executabil şi produce pe ecran un mesaj de eroare.

Exemple de operaţii de împărţire fără semn care arată pregătirea operanzilor în cazul în care nu se încadrează în unul din cele două tipuri de împărţire: cuvânt la octet sau dublu cuvânt la cuvânt.

```
.data
      B1    db    ?
      B2    db    ?
      W1    dw    ?
      W2    dw    ?
      D1    dd    ?
.code
      MOV   AL,  B1                ;impartire octet la octet
      MOV  AH,  0                  ;
      DIV  B2                      ; AL = cat,  AH = rest
;
      MOV  AX, W1                  ; impartire cuvant la octet
      DIV    B1                    ; AL = cat,  AH = rest
;
      MOV   AX,  word ptr D1 ;impartire dublu cuvant la cuvant
      MOV  DX,  word ptr [D1+2]
      DIV  W1                      ; AX = cat,  DX = rest
;
      MOV   AX,  W1                ;impartire cuvant la cuvant
      MOV  DX,  0                  ;
      DIV  W2                      ; AX = cat,  DX = rest
;
      MOV   AX,  word ptr D1 ;impartire dublu cuvant la octet
```

```
MOV  DX,  word ptr [D1+2]
MOV  BL,  B1
MOV  BH,  0
DIV  BX                          ; AX = cat,  DX = rest
```

IDIV sursa	(*Integer Divide - Împarte cu semn)*
descriere formală:	(AL) ← (AX) / (sursa) ; (sursa pe 8 biți) (AH) ← (AX) mod (sursa) ; (restul) (AX) ← (DX:AX) / (sursa) ; (sursa pe 16 biți) (DX) ← (DX:AX) mod (sursa) ; (restul)
descriere:	Se face împărțirea operandului din acum. cu operandul sursă, rezultatul (de dimensiune egală cu sursa) fiind stocat în acum. sau acum. extins. Operanzii sunt cu semn (0 pentru +, 1 pentru -)
operanzi:	Sursa este registru sau locație de memorie de 8 sau 16 biți; variantele superioare de microprocesoare permit ca sursa să fie şi dată imediată. Câtul trebuie să fie în domeniul [-128, +127] la împărțirea pe 8 biți şi în domeniul [-32768, +32767] la împărțirea pe cuvânt.
fanioane afectate:	toate flagurile sunt nedefinite.

Observație: Împărțirea poate duce la depăşiri, deoarece se poate obține câtul mai mare decât valoarea maximă reprezentabilă pe 8 biți, respectiv 16 biți sau dacă împărțitorul este 0. Rezultatele sunt nedefinite şi se generează o întrerupere de tip soft pe nivelul 0 (Divide Overflow - Depăşire la împărțire).

Exemplu:

```
MOV  AX,  500
MOV  BL,   2
IDIV   BL
```

Câtul ar trebui să rezulta 250, valoare ce nu se poate reprezenta pe un octet cu semn. Se generează întrerupere pe nivelul 0.

Calculul restului la împărțirea cu semn se face conform formulei:

$$x = y*(x / y) + x \bmod y,$$

unde x / y este câtul iar x mod y este restul; de exemplu în secvența:

```
MOV  AX,  -10
MOV  BL,  -3
IDIV  BL                    ;AL = 3,  AH = FFH
```

câtul rezultă +3 iar restul, conform formulei, -1, reprezentat pe un octet ca FFH .

Tipurile posibile de împărțire sunt aceleaşi ca la DIV, cu observația că operanzii trebuie pregătiți folosind instrucțiunile CBW, CWD.

```
.data
        B1    db    ?
        B2    db    ?
        W1    dw    ?
        W2    dw    ?
        D1    dd    ?
.code
        MOV   AL,  B1                ;impartire octet la octet
        CBW                          ;
        IDIV  B2                     ; AL = cat,  AH = rest
;
        MOV  AX, W1                  ; impartire cuvant la octet
        IDIV   B1                    ; AL = cat,  AH = rest
;
        MOV   AX,  word ptr D1       ;impartire dublu cuvant la cuvant
        MOV  DX,  word ptr [D1+2]
        IDIV  W1                     ; AX = cat,  DX = rest
;
        MOV   AX,  W1                ;impartire cuvant la cuvant
        CWD                          ;
        IDIV  W2                     ; AX = cat,  DX = rest
;
        MOV   AL,  B1                ;impartire dublu cuvant la octet
        CBW
        MOV  BX,  AX                 ; BX = deîmpărțit
        MOV  AX,  word ptr D1
        MOV  DX,  word ptr [D1+2]    ;DX:AX = împărțitor
        IDIV  BX                     ; AX = cat,  DX = rest
```

Aplicație: conversia unui cuvânt de 16 biți din binar în baza 10;

AAD	(*ASCII Adjust for Division - Corectie ASCII a acumulatorului* ***înainte de*** *împărțire)*
descriere formală:	(AL) ← (AH) * 10 + AL (AH) ← 0

descriere:	Operația de corecție trebuie făcută înainte de împărțirea unui număr pe două cifre BCD reprezentat pe un cuvânt, la o cifră BCD reprezentată pe un octet; corecția se face prin înmulțirea lui AH cu 10 şi adunarea lui AL, adică se face conversia numărului în BCD în cod binar, pe 8 biți; de aceea se pune 00H în AH.
operanzi:	instrucțiunea are operanzi impliciți: AL, AH;
fanioane afectate:	PF, SF, ZF ; restul sunt nedefinite;

Exemplu:

Dacă AX = 0305H, adică valoarea BCD 35 şi BL = 2, secvența:

```
AAD             ; AX = 23H
DIV  BL         ; AL = 11H, AH = 1
```

produce câtul 17 şi restul 1 (corect).

13.3 Instrucțiuni logice (NOT, AND, TEST, OR, XOR)

Realizează funcțiile logice de bază, pe octet sau cuvânt. Operațiile se fac la nivel de bit, între biții de acelaşi rang din cei doi operanzi. Instrucțiunea NOT are un singur operand, celelalte au câte doi operanzi. Ca şi instrucțiunile aritmetice, cele logice afectează în general indicatorii de condiții.

NOT dest.	(*Not - Negare logică bit cu bit)*
descriere formală:	dest ← not (dest) ;
descriere:	Se face înlocuirea valorii fiecărui bit cu valoarea complementară (0 cu 1 şi 1 cu 0), adică se realizează complementul față de 1 al operandului.
operanzi:	destinație este registru sau locație de memorie de 8 sau 16 biți;
fanioane afectate:	nici unul.

AND dst, sursa	(*And - Şi logic bit cu bit)*

descriere formală:	(dest) ← (dest) and (sursa) ;
descriere:	Se realizează operația logică ȘI bit cu bit între cei doi operanzi; sursa nu se modifică, rezultatul se obține în destinație.
operanzi:	destinație este registru sau locație de memorie de 8 sau 16 biți iar sursa poate fi un registru, o locație de memorie sau o constantă pe 8 sau 16 biți.
fanioane afectate:	SF, ZF, PF, CF = 0, OF = 0, AF nedefinit.

TEST dst, sursa	(*Test - Testează bit cu bit)*
descriere formală:	(dest) and (sursa) ;
descriere:	Se realizează operația logică ȘI bit cu bit între cei doi operanzi; nu se modifică nici sursa nici destinația, deci nu se generează rezultat, dar se pozitionează indicatorii de condiții ca la instr. AND
operanzi:	destinație este registru sau locație de memorie de 8 sau 16 biți iar sursa poate fi un registru, o locație de memorie sau o constantă pe 8 sau 16 biți.
fanioane afectate:	SF, ZF, PF, CF = 0, OF = 0, AF nedefinit.

OR dest, sursa	(*Or - Sau logic bit cu bit)*
descriere formală:	(dest) ← (dest) sau (sursa) ;
descriere:	Se realizează operația logică SAU bit cu bit între cei doi operanzi; sursa nu se modifică, rezultatul se obține în destinație.
operanzi:	destinație este registru sau locație de memorie de 8 sau 16 biți iar sursa poate fi un registru, o locație de memorie sau o constantă pe 8 sau 16 biți.
fanioane afectate:	SF, ZF, PF, CF = 0, OF = 0, AF nedefinit.

XOR dst, sursa	(*Exclusive Or - Sau exclusiv logic bit cu bit)*
descriere formală:	(dest) ← (dest) xor (sursa) ;

descriere:	Se realizează operația logică SAU EXCLUSIV bit cu bit între cei doi operanzi; sursa nu se modifică, rezultatul se obține în destinație.
operanzi:	destinație este registru sau locație de memorie de 8 sau 16 biți iar sursa poate fi un registru, o locație de memorie sau o constantă pe 8 sau 16 biți.
fanioane afectate:	SF, ZF, PF, CF = 0, OF = 0, AF nedefinit.

Operații tipice în care sunt utilizate frecvent instrucțiunile logice:

1. Stergerea rapidă a unui registru , cu poziționarea indicatorilor de condiții:

```
XOR  BX,  BX
XOR  CL,  CL
```

2. Forțarea unor biți la valoarea 1, restul rămânând neschimbați:

```
MASCA  EQU          00110101B
OR  AL,  MASCA
```

Pseudoinstrucțiunea EQU definește constante simbolice iar sufixul B indică scrierea în baza 2. Biții cu valoare 1 în *masca* vor fi forțați în 1 în registrul AL iar cei cu valoare 0 în *masca* vor rămâne nemodificați în AL.

3. Forțarea unor biți în 0, restul; neschimbați:

```
 MASCA  EQU          00111111B
AND  AL,  MASCA
```

Biții cu valoarea 0 în *masca*, vor deveni 0 în AL, iar cei cu valoare 1 în masca vor rămâne neschimbați.

4. Testarea unui singur bit dintr-un operand:

```
TEST  AL,  0 0 1 0 0 0 0 0 B
JZ      ETICHETA
```

Dacă bitul marcat cu "1" în mască este "0" în AL, se execută salt.

5. Poziționarea indicatorilor de condiții fără a modifica operandul:

```
OR    AX,  AX
AND  AX,  AX          ; poziționează indicatorii conform
TEST AX,  AX          ; valorii din AX
```

6. Complementarea unui grup de biți, fără modificarea celorlalți:

```
MASCA      EQU  0 0 1 1 0 0 0 1 B
```

```
MOV  BL,  AL              ; salvare
AND  AL,  NOT MASCA       ; selectie biti 1 care nu se modifică
NOT  BL                   ; complementare
AND  BL,  MASCA           ; selectie biti 1 care se modifică
OR   AL,  BL              ; rezultat final
```

Expresia NOT MASCA se evaluează la asamblare, rezultând o constantă cu toți biții negați (aici 1100 1110). Acelaşi rezultat se obține cu secvența:

```
MASCA     EQU  0 0 1 1 0 0 0 1 B
XOR  AL,  MASCA
```

13.4 Instrucțiuni de deplasare (SHL, SAL, SHR, SAR) şi de rotație (ROL, RCL, ROR, RSR)

Acest grup de instrucțiuni realizează operații de deplasare şi rotație la nivel de bit, spre stânga sau spre dreapta. Instrucțiunile au doi operanzi: primul este operandul supus prelucrării iar al doilea este un contor care arată numărul de biți cu care se face deplasarea. Deplasarea cu un bit la dreapta este echivalentă cu împărțirea prin 2 iar la stânga cu înmulțirea cu 2 a operandului supus prelucrării.

Forma generală a instrucțiunilor este:

OPERATIE operand, contor

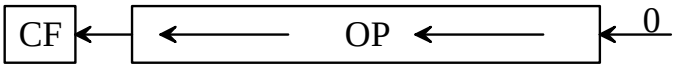
Operand este un registru sau o locație de memorie de 8 sau 16 biți iar contor este constanta 1 dacă se doreşte deplasare cu un bit, sau registrul CL dacă se doreşte deplasare cu 2 sau mai mulți biți. Procesoarele superioare lui I 8086 acceptă o constantă în locul lui CL.

Indicatorii de condiții sunt afectați astfel: la deplasare se modifică toți conform rezultatului, cu excepția lui AF - nedefinit. La rotații se modifică numai CF şi OF.

Indicatorul OF este poziționat numai dacă se face o deplasare sau rotație cu un bit: la deplasare stânga, dacă b.c.m.s.(OP) <> CF, OF = 1, altfel OF = 0; la deplasare dreapta, dacă cei doi b.c.m.s.(OP) sunt diferiți, OF = 1, altfel OF = 0.

Deplasările sunt logice sau aritmetice, în funcție de natura operanzilor (fără semn, respectiv cu semn).

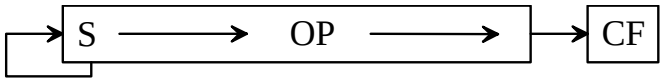
SHL OP, CON ***SAL OP, CON***	(*Shift Logic Left - Deplsare logică spre stânga)* (*Shift Arithmetic Left - Deplsare aritmetica stânga)*
descriere formală:	CF ← b.c.m.s.(OP) ; (OP) ← (OP) * 2 ; CL ← CL -1 ; (dacă este specificat) se repetă de câte ori arată contor.
descriere:	Bitul cel mai semnificativ (b.c.m.s.) trece în CF iar ceilalți biți se deplasează cu o poziție spre stânga; bitul b_0 = 0.
operanzi:	Registru sau locație de 8 sau 16 biți
fanioane afectate:	Toate, AF nedefinit.



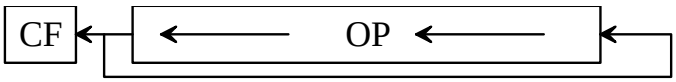
SHR OP, CON	(*Shift Logic Right - Deplsare logică spre dreapta)*
descriere formală:	CF ← b.c.m.p.s.(OP) ; (OP) ← (OP) / 2 ; CL ← CL -1 ; (dacă este specificat) se repetă de câte ori arată contor.
descriere:	Bitul cel mai puțin semnificativ trece în CF iar ceilalți biți se deplasează cu o poziție spre dreapta; bitul b_7 = 0.
operanzi:	Registru sau locație de 8 sau 16 biți
fanioane afectate:	Toate, AF nedefinit.

SAR OP, CON	(*Shift Arithmetic Right - Deplasare aritmetică spre dreapta*)

descriere formală:	CF ← b.c.m.p.s.(OP) ; (OP) ← (OP) / 2 ; operație cu semn CL ← CL -1 ;(dacă este specificat) se repetă de câte ori arată contor.
descriere:	Bitul cel mai puțin semnificativ trece în CF iar ceilalți biți se deplasează cu o poziție spre dreapta; bitul b_7 (b_{15}) = bitul de semn (se conservă bitul de semn).
operanzi:	Registru sau locație de 8 sau 16 biți
fanioane afectate:	Toate, AF nedefinit.

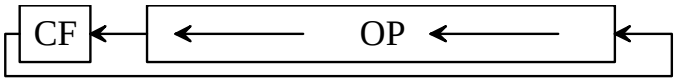


ROL OP, CON	(*Rotate Left - Rotație la stânga)*
descriere formală:	CF ← b.c.m.s.(OP) ; (OP) ← (OP) * 2 +CF ; CL ← CL -1 ; (dacă este specificat) se repetă de câte ori arată contor.
descriere:	Bitul cel mai semnificativ trece în CF iar ceilalți biți se deplasează cu o poziție spre stânga; bitul b_0 = CF actual.
operanzi:	Registru sau locație de 8 sau 16 biți
fanioane afectate:	Toate, AF nedefinit.

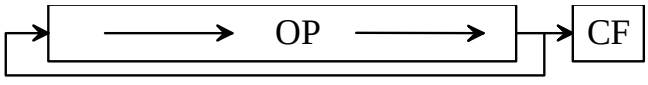


RCL OP, CON	(*Rotate Left through Carry - Rotație la stânga prin Carry)*

descriere formală:	CF ← b.c.m.s.(OP) ; (OP) ← (OP) * 2 + CF (inițial) ; CL ← CL -1 ; (dacă este specificat) se repetă de câte ori arată contor.
descriere:	Bitul cel mai semnificativ trece în CF iar ceilalți biți se deplasează cu o poziție spre stânga; bitul b_0 = CF înainte de depl. lui b7 (b15) în CF
operanzi:	Registru sau locație de 8 sau 16 biți
fanioane afectate:	Toate, AF nedefinit.

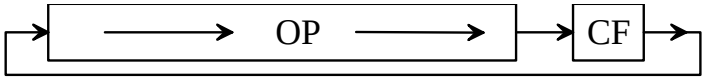


ROR OP, CON	(*Rotate Right - Rotație spre dreapta)*
descriere formală:	CF ← b.c.m.p.s.(OP) ; (OP) ← (OP) / 2 ; b.c.m.s.(OP) ← CF actual CL ← CL -1 ; (dacă este specificat) se repetă de câte ori arată contor.
descriere:	Bitul cel mai puțin semnificativ trece în CF iar ceilalți biți se deplasează cu o poziție spre dreapta; bitul b_7 = CF sau b_{15} = CF.
operanzi:	Registru sau locație de 8 sau 16 biți
fanioane afectate:	Toate, AF nedefinit.



RCR OP, CON	(*Rotate Right through Carry - Roteşte spre dreapta prin Carry)*

descriere formală:	CF ← b.c.m.p.s.(OP) ; (OP) ← (OP) / 2 ; operație fără semn b.c.m.s.(OP) ← CF (anterior) CL ← CL -1 ; (dacă este specificat) se repetă de câte ori arată contor.
descriere:	Bitul cel mai puțin semnificativ trece în CF iar ceilalți biți se deplasează cu o poziție spre dreapta; bitul b_7 = CF anterior sau b_{15} = CF anterior.
operanzi:	Registru sau locație de 8 sau 16 biți
fanioane afectate:	Toate, AF nedefinit.



Exemple de rotații şi deplasări:

1. Înmulțirea şi împărțirea cu puteri ale lui 2 se execută mult mai eficient prin deplasări la stânga respectiv dreapta:

```
MOV  CL,  4
MOV  AH,  0
SHL   AX,  CL     ; înmulțirea lui AL cu 16, rezultatul în AX.
```

Secvența:

```
MOV  CL,  3
SAL  BX,  CL
```

realizează împărțirea lui BX prin 8 a numărului din BX, considerat cu semn.

2. Înmulțirea unui operand cu un număr care nu este putere a lui 2.

Secvența următoare are ca efect înmulțirea valorii N din AL cu 13 prin deplasări şi adunări repetate; rezultatul se obține în BX.

```
MOV  AH,  0
MOV  BX,  AX          ; Salvare N în BX
MOV  DX,  AX          ; Salvare N în DX
MOV  CL,  3           ; Pregătire contor pentru  3 depl.
SHL   AX,  CL         ; AX = N * 8
ADD  BX,  AX          ; BX = N * 8 + N
MOV  AX,  DX          ; Refacere N în AX
MOV  CL,  2           ; Pregătire contor pentru 2 depl.
```

```
SHL  AX, CL          ; AX = N * 4
ADD  BX, AX          ; BX ← N* 8 + N * 4 + N = 13 N
```

13.5 Instrucțiuni pentru operații cu şiruri

Prin "şir" se înțelege o secvență de octeți (caractere) sau cuvinte aflate la adrese succesive de memorie. Operațiile de bază pot fi executate repetat dacă se pune un prefix de repetare.

Operațiile de bază: se realizează cu octeți (B) sau cuvinte (W) şi sunt grupate în 5 categorii:

Move String (Copiază şir)	MOVSB,	MOVSW
Compare String (Compară şiruri)	CMPSB	CMPSW
Load String (Încarcă şir în AL/AX)	LODSB	LODSW
Store String (Depune AL/AX în şir)	STOSB	STOSW
Scan String (Compară şir cu AL/AX)	SCASB	SCASW

Toate operațiile folosesc registrele DS:SI ca adresă sursă şi/sau ES:DI ca adresă destinație. Toate instrucțiunile produc actualizarea adreselor implicate în operație, în funcție de valoarea indicatorului DF (ce poate fi poziționat prin instrucțiunile CLD - Clear Direction, sau STD - Set Direction):

dacă DF = 0 adresele cresc (SI, DI se incrementează cu 1 sau 2)

dacă DF = 1 adresele scad (SI, DI dse decrementează cu 1 sau 2).

CLD produce DF = 0; STD produce DF = 1.

Vom utiliza notațiile: (SI) ← (SI) + *delta* (DI) ← (DI) + *delta* unde *delta* este +1, -1, +2, -2, în funcție de starea bistabilului DF şi de dimensiunea operanzilor (octeți sau cuvinte).

MOVSB ***MOVSW***	(*Move String Byte - Copiază şir de octeți)* (*Move String Word - Copiază şir de cuvinte)*
descriere formală:	((DS : SI)) ← ((ES : DI)) ; (SI) ← (SI) + *delta* ; (DI) ← (DI) + *delta* ;
descriere:	Se transferă un octet sau un cuvânt de la adresa sursă dată de (DS:SI) la adresa dată de (ES:DI) şi se actualizează adresele în funcție de starea indicatorului DF şi de dimensiunea operanzilor; conținutul locațiilor sursă nu se modifică !

operanzi:	Locații de memorie de 8 sau 16 biți;
fanioane afectate:	Nici unul.

CMPSB ***CMPSW***	(*Compare String Byte - Compară şiruri de octeți)* (*Compare String Word - Compară şiruri de cuvinte)*
descriere formală:	((DS : SI)) – ((ES : DI)) ; (SI) ← (SI) + *delta* ; (DI) ← (DI) + *delta* ;
descriere:	Se evaluează diferența dintre octeții sau un cuvintele de la adresa sursă dată de (DS:SI) şi cele de la adresa dată de (ES:DI) şi se actualizează adresele în funcție de starea indicatorului DF şi de dimensiunea operanzilor. **Nu se modifică valoarile opranzilor !**
operanzi:	Locații de memorie de 8 sau 16 biți;
fanioane afectate:	AF, CF, PF, SF, ZF, OF; (toate)

LODSB ***LODSW***	(*Load String Byte - Încarcă şir de octeți în acc.)* (*Load String Word - Încarcă şir de cuvinte în acc.)*
descriere formală:	(AL) ← ((DS : SI)) ; transfer de octet (AX) ← ((DS : SI)) ; transfer de cuvânt (SI) ← (SI) + *delta* ;
descriere:	Se încarcă un octet sau un cuvânt de la adresa sursă dată de (DS:SI) în acumulator (AL sau AX) şi se actualizează adresa sursă în funcție de DF şi de dimensiunea operandului; conținutul locațiilor sursă nu se modifică !
operanzi:	AL, AX, locații de memorie de 8 sau 16 biți;
fanioane afect.	Nici unul.

STOSB ***STOSW***	(*Store String Byte - Depune acc. în şir)* (*Store String Word - Depune acc. în şir)*
descriere formală:	((ES : DI)) ← (AL) ; transfer de octet ((ES : DI)) ← (AX) ; transfer de cuvânt (DI) ← (DI) + *delta* ;

descriere:	Se depune un octet sau un cuvânt din acumulator (AL sau AX) în memorie, la adresa dată de (ES:DI) şi se actualizează adresa de destinație în funcție de starea indicatorului DF şi de dimensiunea operandului; conținutul acumulatorului nu se modifică !
operanzi:	AL, AX, locații de memorie de 8 sau 16 biți;
fanioane afect.	Nici unul.
SCASB ***SCASW***	(*Scan String Byte - Compară şir de octeți cu AL)* (*Scan String Word - Compară şir de cuvinte cu AX)*
descriere formală:	(AL) - ((ES : DI)) ; Comparație de octeti (AX) - ((ES : DI)) ; Comparație de cuvânte (DI) ← (DI) + *delta* ;
descriere:	Se evaluează diferența dintre octetul sau un cuvântul de la adresa dată de (ES:DI) şi cele din acumulator (Al sau AX) şi se actualizează adresele în funcție de starea indicatorului DF şi de dimensiunea operanzilor. **Nu se modifică valoarile opranzilor !**
operanzi:	AL, AX, locații de memorie de 8 sau 16 biți;
fanioane afect.	AF, CF, PF, SF, ZF, OF; (toate).

Instrucțiunile se utilizează la testarea sau căutarea unui anumit octet sau cuvânt într-un şir.

Observație:

Pe lângă formele fără operanzi, descrise mai sus, asamblorul recunoaşte şi forme în care operanzii apar explicit; adresarea se face însă implicit tot cu SI şi DI fiind specificat şi un prefix de segment pentru adresa sursă. În acest caz mnemonica se scrie fără litera B sau W de la sfârşit dar este obligatorie specificarea dimensiunii operandului cu operatorul PTR.

13.6 Prefixe de repetare

Prefixele de repetare se utilizează pentru a produce execuția repetată a unei operații de bază cu şiruri, în funcție de valoarea unui contor de repetare sau a valorii unei condiții logice.

Aceste prefixe nu sunt instrucţiuni în sine, ci participă la formarea unor instrucţiuni compuse pentru realizarea buclelor repetitive de tip FOR, REPEAT sau WHILE, din limbajele de nivel înalt.

REP INSS ***REPE INSS*** ***REPZ INSS***	(*Repeat - Repetă instrucţiunea de bază specificată)*
descriere formală:	Cât timp CX <> 0 execută Instrucţiune_pt._şiruri CX ← CX - 1 dacă (Iss = CMPS sau Iss = SCAS) şi ZF = 0, STOP.
descriere:	Numărul maxim de repetări este cel dat de CX (dacă CX este iniţial 0, Iss nu se execută). În cazul instrucţiunilor CMPS şi SCAS (care poziţionează ZF) ieşirea din buclă este forţată dacă ZF = 0 (rezultat nenul); bucla se execută deci cât timp rezultatul este 0 şi CX <> 0.
operanzi:	ca la Iss corspunzător
fanioane afectate:	ca la Iss corespunzător

De obicei, scrierea cu REP se foloseşte la MOVS, LODS şi STOS iar scrierea cu REPE, REPZ la CMPS şi SCAS.

Exemple:

Secvenţa de mai jos transferă 100 octeţi de la adresa SURSA la adresa DEST locaţiile fiind presupuse în segmentul curent adresat cu DS.

```
.data
      SURSA     db    100   dup   (?)
      DEST      db    100   dup   (?)
.code
      CLD                         ; adrese crescatoare
      MOV       AX,  DS           ; pregatire
      MOV       ES,  AX           ; adrese
      LEA       SI,  SURSA        ; Adresa sursa
      LEA       DI, DEST          ; Adresa destinaţie
      MOV       CX,  100          ; Contorul se initializeaza
                                  ; cu nr. de octeti
      REP       MOVSB             ;Instructiunea compusa repetitiva
```

Secvența următoare identifică primul octet care diferă de un octet dat, dintr-un şir de 200 octeți:

```
.data
      SIR   db    200   dup   (?)
.code
      MOV         AX,  DS           ; pregatire
      MOV         ES,  AX           ; adrese
      LEA         DI,  SIR
      CLD
      MOV  AL,   'A'                ; octetul model este 41H
      MOV         CX,  200          ; numarul maxi de repetari
      REPE        SCASB             ; Instructiunea compusa repetitiva
```

Examinând bistabilul ZF la ieşirea din buclă se deduce rezultatul căutării:

- Dacă ZF = 0, a avut loc o ieşire forțată din buclă, deci registrul DI decrementat arată adresa primului octet din şir diferit de cel din AL.
- Dacă ZF = 1, toți octeții comparați au fost identici cu cel de referință, din AL.

REPNE Iss ***REPNEZ Iss***	(*Repeat While Not Equal / Not Zero - Repetă instrucțiunea de bază cât timp diferit / diferit de zero)*
descriere formală:	Cât timp CX <> 0 execută Instrucțiune_pt._şiruri CX ← CX - 1 dacă (Iss = CMPS sau Iss = SCAS) şi ZF = 1, STOP.
descriere:	Numărul maxim de repetări este cel dat de CX (dacă CX este inițial 0, Iss nu se execută). În cazul instrucțiunilor CMPS şi SCAS (care poziționează ZF) ieşirea din buclă este forțată dacă ZF = 1 (rezultat = 0); bucla se execută deci cât timp rezultatul este nenul şi CX <> 0.

La ieşirea din buclă se poate examina ZF pentru a stabili dacă a fost o ieşire forțată sau nu. Practic, acest prefix se foloseşte numai cu

operațiile CMPS şi SCAS. Pentru celelalte se preferă scrierea cu prefixul REP.

Exemple:

1. Se determină ultimul caracter egal cu un caracter dat prin parcurgerea şirului în sens invers.

```
.data
      SIR   db    30    (?)
.code
      LEA   DI,  SIR
      ADD   DI,  29
      MOV  CX,  30
      STD
      MOV   AL,  '$'          ; Se caută primul octet egal cu '$'
      REPNE      SCASB
```

2. Secvența de mai jos transferă 100 octeți de la adresa SURSA la adresa DEST ambele locații fiind presupuse în segmentul curent adresat cu DS.

```
.data
      SURSA      db    100   dup   (?)
      DEST       db    100   dup   (?)
.code
      CLD                                ; adrese crescatoare
      MOV        AX,  DS                 ; pregatire
      MOV        ES,  AX                 ; adrese
      LEA        SI,  SURSA        ; Adresa sursa
      LEA        DI, DEST          ; Adresa destinație
      MOV   CX,  100               ; Contorul se initializeaza
                                   ; cu nr. de octeti
      REP        MOVSB             ; Instructiunea compusa repetitiva
```

13.7 Instrucțiuni de apel procedură şi de salt (CALL, RET, JMP)

Aceste instrucțiuni au ca efect transferul execuției la o adresă de program specificată printr-o etichetă (salt) sau prin nume de procedură, caz în care saltul care se execută este cu revenire în punctul de apel.

13.7.1. Instrucțiuni de apel/revenire la/din procedură

Procedurile se definesc în textul sursă după modelul:

```
nume_proc  PROC [FAR | NEAR]
. . . . . .
. . . . . .
      RET
nume_proc  ENDP
```

unde nume_proc este numele procedurii iar parametrii FAR sau NEAR (opționali) indică tipul procedurii.

O procedură FAR poate fi apelată şi din alte segmente de cod decât cel în care este definită; o procedură NEAR poate fi apelată numai din segmentul de cod în care este definită.

Dacă parametrii lipsesc, tipul procedurii este dedus din directivele de definire a segmentelor (modelul LARGE determină ca toate procedurile să fie de tip FAR.

Instrucțiunea RET (*Return*), care produce revenirea în programul apelant, are trei variante:

- RETN (*Return Near*);
- RETF (*Return Far*);
- RET (*Return*) când tipul de revenire este dedus din tipul procedurii.

CALL nume_P *Call - Apel de procedură* ***CALL FAR ptr nume_P*** ***CALL NEAR ptr nume_P***	
descriere formală: *pentru NEAR*	(SP) ← (SP) - 2 SS: ((SP)+1) ← (IP) (high) SS: ((SP)) ← (IP) (low) (IP) ← Offset (al primei instrucțiuni din procedură)
descriere formală: *pentru FAR*	(SP) ← (SP) - 2 SS: ((SP)+1) ← (CS) (high) SS: ((SP)) ← (CS) (low) (SP) ← (SP) - 2 SS: ((SP)+1) ← (IP) (high) SS: ((SP)) ← (IP) (low) (CS) ← adresa de segment (a primei instrucțiuni din procedură) (IP) ← Offset (al primei instrucțiuni din procedură)

descriere: *pentru NEAR*	Se salvează în stivă contorul de program curent IP; acesta conține adresa efectivă (offset-ul) a instrucțiunii ce urmează după CALL, numită adresă de revenire; această adresă nu trebuie modificată în nici un fel, în caz contrar revenirea în programul apelant după execuția procedurii nu mai este posibilă. **Registrul CS nu se modifică**. Se încarcă apoi în IP adresa (offset-ul) primei instrucțiuni din corpul procedurii, adică se transferă controlul către procedură
descriere : *pentru FAR*	Ceea ce diferă față de apelul de tip NEAR este că se salvează în stivă adresa completă de revenire (pe 32 biți) formată din conținutul lui IP şi conținutul lui CS. Similar, transferul controlului către procedură se face prin încărcarea în CS:IP a adresei complete de 32 biți corespunzătoare primei instrucțiuni din procedură.

Observație:

Instrucțiunea CALL de tip FAR este una din puținele instrucțiuni care modifică explicit registrul CS.

Salvarea în stivă se face automat dar numai pentru CS şi IP.

Registrele generale şi indicatorii de condiții se salvează în stivă la începutul subrutinei, prin instrucțiuni PUSH, introduse de programator.

La încheierea părții executabile a unei proceduri, registrele salvate în stivă trebuie restabilite cu POP, în ordinea inversă introducerii.

RETN [N] ***RETF [N]*** ***RET [N]***	*Return - Revenire din procedură*
descriere formală: *pentru NEAR*	(IP) (high) ← SS: ((SP)+1) (IP) (low) ← SS: ((SP)) (SP) ← (SP) + 2 (SP) ← (SP) + 2 [(SP) ← (SP) + N] opțional; N este o constantă

descriere formală: *pentru FAR*	(IP) (high) ← SS: ((SP)+1) (IP) (low) ← SS: ((SP)) (SP) ← (SP) + 2 (CS) (high) ← SS: ((SP)+1) (CS) (low) ← SS: ((SP)) (SP) ← (SP) + 2 [(SP) ← (SP) + N] opțional; N este o constantă
descriere: *pentru NEAR*	Se reface din stivă contorul de program curent IP; acesta conține adresa efectivă (offset-ul) a instrucțiunii ce urmează după CALL, numită adresă de revenire; Dacă în formatul instrucțiunii RET există constanta opțională N, atunci se adună această constantă la SP (Return cu descărcarea stivei)
descriere : *pentru FAR*	Se reface din stivă perechea de registre CS:IP cu actualizarea registrului SP şi dacă este prazentă constanta N, se adună la SP.

Pentru ca mecanismul de apel / revenire să funcționeze corect, trebuie îndeplinite condițiile:

1. Tipul instrucțiunii CALL şi cel al instr. RET, să coincidă (FAR sau NEAR)
2. Registrul SP din momentul execuției instr. RET să aibă aceeaşi valoare ca în momentul execuției instr. CALL cu care face pereche (să indice adresa de revenire).
3. Adresa de revenire salvată temporar în stivă să nu fi fost afectată de către procedură.

Încălcarea uneia din condiții este o eroare frecventă de programare. În asemenea cazuri, funcționarea programului este compromisă deoarece controlul execuției este iremediabil pierdut.

13.7.2. Instrucțiunea JMP, de salt necondiționat

JMP *tinta*	*Jump - Salt la ținta (adresa de salt)*
descriere formală: *de tip SHORT*	(IP) ← (IP) + distanța dintre offset - ul curent şi cel țintă.

descriere formală: *de tip NEAR*	(IP) ← offset - ul adresei ţintă.
descriere: *de tip FAR*	(IP) ← offset - ul adresei ţintă. (CS) ← segmentul adresei ţintă.
descriere : *pentru SHORT*	Se execută salt în program, adresa ţintă fiind pe un octet cu semn şi se specifică printr-o etichetă sau printr-o expresie. Codul instrucţiunii conţine diferenţa dintre offset-ul curent şi cel al ţintei, care se memorează intern pe un octet cu semn, de unde restricţia de domeniu [-128, +127].
descriere : *pentru NEAR*	Se execută salt în program, adresa ţintă fiind în acelaşi segment de cod cu instrucţiunea JMP şi se specifică printr-o etichetă sau printr-o expresie.
descriere : *pentru FAR*	Se execută salt în program, adresa ţintă fiind în alt segment de cod faţă de instrucţiunea JMP şi se specifică printr-o etichetă sau printr-o expresie.

Etichetele au asociat un tip (NEAR sau FAR) şi sunt:
- un nume de procedură;
- o etichetă definită cu semnul ':'
- o etichetă definită cu directiva LABEL.

Exemple:
```
et7:
et100     LABEL  FAR
Miami     LABEL  NEAR
```

Tipurile de salt se deduc din atributele expresiei ţintă sau în cazul etichetelor după tipul acestora.

Expresiile din sintaxa instrucţiunii JMP pot fi:

a) un registru care conţine offset-ul ţintei;

b) o variabilă de tip WORD care conţine offset-ul ţintei;

c) o expresie cu indici reprezentând un cuvânt din memorie care conţine offset-ul ţintei;

d) o referire anonimă la un cuvânt din memorie care conţine offset-ul ţintei;

```
JMP     BX                      ; tipul a)
JMP     W_ALFA                  ; tipul b)
JMP     W_TAB_PROC  [SI]        ; tipul c)
LEA     BX, W_TAB_PROC
JMP     WORD  PTR  [BX] [SI]    ;tipul d)
```

În cazul salturilor de tip FAR, expresia din sintaxă poate fi:

a) o variabilă de tip DW care conține adresa completă a țintei

b) o expresie cu indici reprezentând un dublu cuvânt din memorie care conține adresa completă a țintei;

c) o referire anonimă la un cuvânt din memorie care conține adresa țintei.

13.7.3. Instrucțiuni de salt condiționat

Realizează salturi la o adresă țintă în funcție de valoarea unor indicatori de condiții.

Caracteristici:

- toate instrucțiunile de salt condiționat sunt de tip SHORT (directe) deci adresa țintă trebuie să fie la o distanță între -128, +127 de octeți față de adresa instrucțiunii de salt din program;
- există mai multe variante pentru aceeaşi instrucțiune;
- dacă nu este îndeplinită condiția specificată în instrucțiune, saltul nu are loc, deci execuția continuă cu instrucțiunea următoare celei de salt;
- există instrucțiuni pentru condiția directă, cât şi pentru condiția negată;
- indicatorii de condiții nu sunt afectați.

Forma generală:

JXXX eticheta

unde XXX este condiția specificată prin maxim trei litere.

Instrucțiune	Condiție de salt	Interpretare
JE, JZ	ZF = 1	Zero, Equal
JL, JNGE	SF <> OF	Less, Not Greater or Equal
JLE, JNG	SF<>OF sau ZF=1	Less or Equal, Not Greater
JB, JNAE, JC	CF = 1	Bellow, Not Above or Equal, Carry
JBE, JNA	CF = 1 sau ZF = 1	Bellow or Equal, Not Above
JP, JPE	PF = 1	Parity, Parity Even
JO	OF = 1	Overflow

JS	SF = 1	Sign
JNE, JNZ	ZF = 0	Not Zero, Not Equal
JNL, JNE	SF = OF	Not Less, Greater or Equal
JNLE, JG	SF = OF şi ZF = 0	Not Less or Equal, Greater
JNB, JAE, JNC	CF = 0	Not Bellow, Above or Equal, Not Carry
JNBE, JA	CF = 0 şI ZF = 0	Not Bellow or Equal, Above
JNP, JPO	PF = 0	Not Parity, Parity Odd
JNO	OF = 0	Not Overflow
JNS	SF = 0	Not Sign

Există două categorii de instrucțiuni pentru noțiunile de "mai mic" şi "mai mare":

- cele care conțin cuvintele ***above*** şi ***bellow*** care se folosesc în cazul comparării a doi operanzi fără semn;
- cele care conțin cuvintele ***less*** şi ***greater***, care se folosesc în cazul comparării unor operanzi cu semn.

```
MOV  AL, 0FFH          MOV  AL, 0FFH
MOV  BL, 1             MOV  BL, 1
CMP  AL, BL            CMP  AL, BL
JA   ET_1              JG   ET_1
```

Dacă interpretăm cele două valori fără semn, atunci rezultă că (AL)>(BL), iar dacă ele sunt interpretate cu semn, rezultă că (AL) < (BL), deoarece -1 < 1.

În primul exemplu, saltul la eticheta_1 are loc iar în al doilea exemplu, nu.

13.8 Instrucțiuni pentru controlul buclelor de program

JCXZ, LOOP, LOOPZ, LOOPE, LOOPNZ, LOOPNE

JCXZ *eticheta*	*Jump if CX is Zero - Salt dacă CX este zero)*

descriere formală:	dacă (CX) = 0 (IP) ← (IP) + distanța dintre offset - ul curent şi cel țintă.
descriere :	Dacă CX este 0000 H, se efectuează saltul la instrucțiunea marcată cu eticheta specificată . Indicatori: nici unul.

13.8.1. Instrucțiuni de ciclare (LOOPxx)

Sunt de fapt salturi condiționate de valoarea indicatorului ZF şi a registrului CX. Se aseamănă cu prefixele de repetare şi sunt câte două instrucțiuni cu acelaşi efect.

LOOP eticheta	(*Loop - Bucleză până la eticheta)*
descriere formală:	CX ← CX - 1 dacă CX <> 0, atunci (IP) ← (IP) + dist. dintre offset - ul curent şi cel țintă.
descriere:	Se decrementează CX şi dacă acesta este diferit de zero se efectuează salt la eticheta specificată.
operanzi:	CX, IP
fanioane afectate:	ZF

Se calculează suma elementelor unui tablou, care sunt numere întregi pe doi octeți (100 de întregi) iar rezultatul se obține în variabila SUMA.

```
.data
      TAB       dw        100  dup  (0)
      SUMA      dw        ?
.code
      XOR       AX,  AX          ; initializeaza  SUMA
      MOV       CX,  100
      MOV       SI,   AX         ; initializeaza  indice
NEXT:
      ADD       AX,  TAB [SI]
      ADD       SI,  2           ; actualizare indice pentru word
```

```
        LOOP    NEXT            ; cicleaza
        MOV     SUMA,  AX       ; rezultat
```

Instrucțiunile din interiorul buclei se execută de atâtea ori cât este valoarea inițială din CX. Dacă CX este inițial 0, după decrementare el devine 65 535 şi ca urmare execuția se repetă tot de atâtea ori. Protecția față de o asemenea situație se face cu instrucțiunea JCXZ:

```
JCXZ end_bucla
start_bucla
        . . .
LOOP start_bucla
end_bucla
```

astfel că dacă CX este inițial zero, corpul buclei nu se execută.

LOOPZ eticheta ***LOOPE eticheta***	(*Loop While Zero / Equal - Buclează cât timp este zero / egal)*
descriere formală:	CX ← CX - 1 dacă CX <> 0 şi ZF = 1 atunci (IP) ← (IP) + distanța dintre offset - ul curent şi cel țintă.
descriere:	Se decrementează CX şi dacă acesta este diferit de zero şi ZF este 1 (rezultatul ultimei operații aritmetice este zero) se efectuează salt la eticheta specificată. De obicei, pentru poziționarea indicatorului se utilizează o instrucțiune de comparație.
operanzi:	CX, IP
fanioane afectate:	ZF

Exemplu:

Se determină primul întreg nenul dintr-un tablou de 100 de numere.

```
.code
        MOV     CX,  100
        LEA     BX,  TAB
        MOV     SI,  -2
next:
        ADD     SI,  2
        CMP     WORD ptr  [BX] [SI], 0
        LOOPZ   next
        JNZ     gata
```

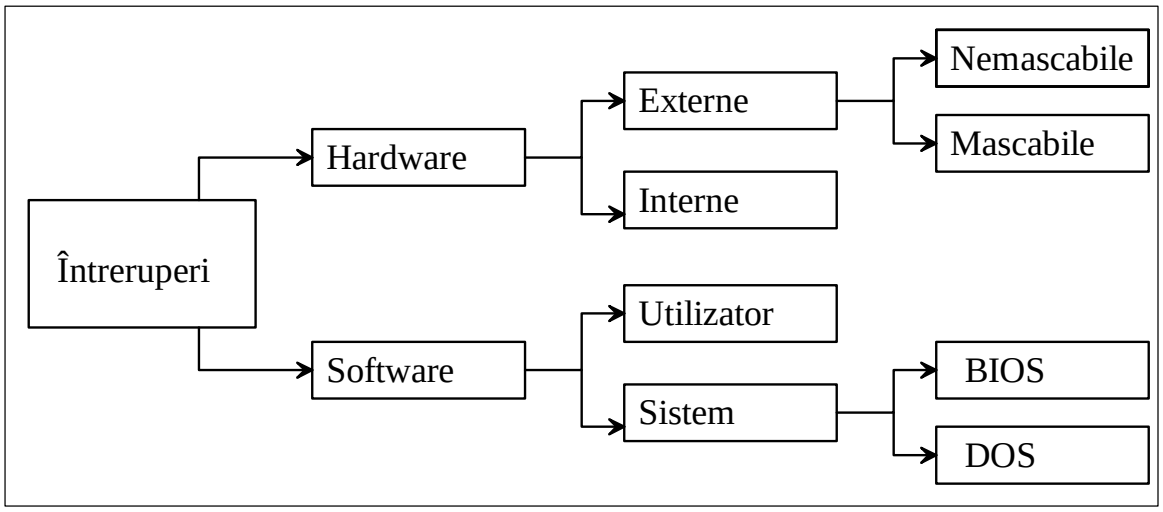
Dacă ZF este zero la ieşirea din buclă, atunci s-a identificat primul întreg nenul, iar SI conține adresa acestuia. În caz contrar, toate elementele tabloului sunt nule.

LOOPNZ eticheta ***LOOPNE eticheta***	(*Loop While Not Zero / Not Equal - Bucleză cât timp este diferit de zero / diferit)*
descriere formală:	CX ← CX - 1 dacă CX <> 0 şi ZF = 0 atunci (IP) ← (IP) + distanța dintre offset-ul curent şi cel țintă.
descriere:	Se decrementează CX şi dacă acesta este diferit de zero şi ZF este 0 (rezultatul ultimei operații aritmetice nu este zero) se efectuează salt la eticheta specificată. De obicei, pentru poziționarea indicatorului se utilizează o instrucțiune de comparație.
operanzi:	CX, IP
fanioane afectate:	ZF

13.9 Întreruperi

O *întrerupere* opreşte temporar execuția unui program şi transferă controlul unei rutine (subprogram) specifice de tratare, ce corespunde cauzei ce a generat întreruperea.

Mecanismul prin care se face acest transfer este în esență de tip apel de procedură, ceea ce implică revenirea în programul întrerupt după execuția rutinei de tratare. Tipurile de întreruperi sunt ilustrate în figura următoare.



Întreruperile hardware externe sunt activate de cereri de întrerupere generate de dispozitive periferice inteligente, sub forma unor semnale electrice, aplicate pe intrările INT şi NMI ale procesorului; cele interne apar ca urmare a unor condiții speciale de funcționare a procesorului (de exemplu, modul de lucru pas cu pas).

Întreruperile mascabile pot fi dezactivate prin comenzi de program şi sunt cele produse de semnale aplicate pe intrarea INT a procesorului; cele nemascabile nu pot fi dezactivate prin comenzi de program şi sunt cele produse de semnale aplicate pe intrarea NMI.

Într-un sistem cu procesor 8086 pot exista maxim 256 de întreruperi distincte. Fiecare din aceste nivele poate avea asociată o procedură de tip FAR, numită rutină de tratare. Adresele acestor rutine sunt înregistrate într-o tabelă de întreruperi aflată la adresele 00000 - 003FFH, ocupând deci 1024 octeți. Fiecare nivel ocupă 4 octeți, primii 2 reprezentând offset-ul iar următorii 2 adresa de segment a procedurii.

La apariția unei întreruperi au loc următoarele acțiuni:

- se salvează în stivă registrele F, CS, IP;
- se pun în zero indicatorii IF şi TF;
- se furnizează procesorului un octet (0 - 255) numit *vector de întrerupere* care identifică nivelul asociat întreruperii curente;
- prin intermediul tabelei de întreruperi se execută salt intersegment la adresa rutinei de tratare;

Vectorul de întrerupere poate fi furnizat procesorului în unul din următoarele moduri:

- în cazul întreruperilor hard interne nivelul este implicit;
- în cazul întreruperilor hard externe, nivelul este transmis prin magistrala de date în cadrul ciclului maşină de tratare, de către dispozitivul care a generat întreruperea.
- în cazul întreruperilor soft, nivelul este conținut în instrucțiune.

13.10 Instrucțiuni specifice întreruperilor (INT, IRET, INTO)

INT n	*Interrupt - Întrerupere Software de nivel n.*
descriere formală:	(SP) ← (SP) - 2 SS: ((SP)+1) ← (Flags) (high) SS: ((SP)) ← (Flags) (low) IF ← 0, TF ← 0 (SP) ← (SP) - 2 SS: ((SP)+1) ← (CS) (high) SS: ((SP)) ← (CS) (low) (CS) ← (4*n+2) (SP) ← (SP) - 2 SS: ((SP)+1) ← (IP) (high) SS: ((SP)) ← (IP) (low) (IP) ← (4*n)
descriere:	Se încarcă în CS şi IP, după salvarea lor în stivă, conținutul de la adresele fizice 4*n+2 şi 4*n, adică cei 4 octeți corespunzători nivelului n din tabela de întreruperi; la adresa CS:IP se află rutina de tratare n.
indicatori:	IF, TF.

IRET	*Interrupt Return- Revenire din întrerupere Software*
descriere formală:	(IP) (high) ← SS: ((SP)+1) (IP) (low) ← SS: ((SP)) (SP) ← (SP) + 2 (CS) (high) ← SS: ((SP)+1) (CS) (low) ← SS: ((SP)) (SP) ← (SP) + 2 (Flags) (high) ← SS: ((SP)+1) (Flags) (low) ← SS: ((SP)) (SP) ← (SP) + 2
descriere:	Se refac din stivă CS , IP şi F după execuția rutinei de tratare întrerupere. Revenirea nu se poate face cu RET deoarece trebuie refăcut şi registrul indicatorilor.
indicatori:	IF, TF.

INTO	*Interrupt if Overflow -Întrerupere în caz de depăşire.*
descriere formală:	dacă OF = 1, se execută INT 4
descriere:	dacă OF = 1, se execută INT 4.
indicatori:	IF, TF.

Nivelurile predefinite de întrerupere sunt:

- 0 - depăşire la împărțire ;
- 1 - execuție pas cu pas (TF = 1);
- 2 - întrerupere externă nemascabilă (cerere pe intrarea NMI);
- 3 - execuție pas cu pas (instr. INT 3)
- 4 - depăşire (instr. INTO)

La calculatoarele IBM - PC se mai pot cita întreruperile hardware de la ceasul de timp real (nivelul 8) şi de la tastatură (nivelul 9).

Întreruperile software în gama 20H - 2FH sunt folosite de sistemul de operare DOS, iar cele din gama 10H - 1AH de către subsistemul de intrări - ieşiri al sistemului de operare, BIOS.

13.11 Instrucțiuni pentru controlul procesorului

Toate instrucțiunile din această clasă sunt fără operanzi.

CLC (*Clear Carry Flag*) şterge CF, adică CF = 0;
STC (*Set Carry Flag*) setează CF, adică CF = 1;
CMC (*Complement Carry Flag*) negare CF: CF ← not(CF);
CLD (*Clear Direction Flag*) şterge DF, adică DF = 0;
STD (*Set Direction Flag*) setează DF, adică DF = 1.
CLI (*Clear Interrupt Flag*), determină IF = 0;
STI (*Set Interrupt Flag*) determină IF = 1.

Când IF = 0, întreruperile mascabile (de tip INT) sunt dezactivate (mascate); o cerere de întrerupere de acest tip nu este luată în considerare. O secvență de program care trebuie protejată la întrerupere începe cu CLI şi se termină cu STI (de exemplu, modificarea tabelei de întreruperi, generarea unui interval de timp foarte precis, etc.).

HALT (Oprire procesor) determină intrarea temporară a procesorului într-o stare de inactivitate din care poate ieşi la apariția unei cereri de întrerupere sau la comanda RESET externă.

LOCK (Blocare magistrală) este un prefix ce se poate utiliza înaintea oricărei instrucțiuni şi are ca efect interdicția de cedare a magistralelor unui alt dispozitiv pe durata execuției instrucțiunii pe care o precedă.

WAIT (Aşteaptă) realizează sincronizarea procesorului cu un coprocesor aritmetic. Instrucțiunea introduce procesorul în stare de inactivitate până când coprocesorul răspunde cu semnal electric pe linia de intrare TEST.

NOP (*Non Operation*) - introduce o întârziere în program de câteve perioade de tact, executând de fapt instrucțiunea XCHG AX, AX, care, evident, nu face nimic.

14 Dezvoltarea programelor în limbaj de asamblare

14.1 Tipuri de date

Limbajul de asamblare 80x86 operează cu anumite tipuri de date fundamentale, recunoscute de procesor şi utilizate în formatul instrucțiunilor.

Pentru fiecare tip de date este caracteristic domeniul de valori, care depinde de numărul de octeți necesari pentru reprezentare.

- **Byte** (un octet)

Poate fi reprezentat în memorie sau într-un registru de 8 biți. Un byte poate fi interpretat în următoarele moduri:

- număr întreg pe 8 biți cu sau fără semn;
- caracter alfanumeric în cod ASCII.

Directiva pentru definirea datelor de acest tip este DB sau db (*Define Byte*).

- **Word** (2 octeți)

Poate fi reprezentat în memorie sau într-un registru de 16 biți şi este interpretat în următoarele moduri:

- număr întreg pe 16 biți cu sau fără semn;
- secvență de două caractere ASCII;
- adresă de memorie de 16 biți.

Directiva pentru definirea datelor de acest tip este DW sau dw (*Define Word*). Partea mai puțin semnificativă este memorată la adrese mici, conform regulii generale introduse de Intel.

- **Double word** (4 octeți)

Poate fi reprezentat în memorie, ocupând 4 locații de 8 biți, într-o pereche de registre de 16 biți sau într-un registru de 32 de biți (la procesoarele de 32 şi 64 de biți) şi este interpretat în următoarele moduri:

- număr întreg pe 16 biți cu sau fără semn;
- număr real în simplă precizie;
- adresă de memorie de 16 biți.

Directiva pentru definirea datelor de acest tip este DD sau dd (*Define Double Word*). Partea mai puțin semnificativă este memorată la adrese mici. În cazul memorării adreselor pe 32 de biți, adresa de segment este memorată la adrese mari iar offset-ul la adrese mici.

- **Quad word** (8 octeți)

Ocupă 8 locații succesive de 8 biți sau o pereche de registre de 32 de biți (la procesoarele de 32 şi 64 de biți). Semnificația unui qword poate fi:

- număr întreg pe 64 de biți cu sau fără semn;
- număr real în dublă precizie.

Directiva pentru definirea datelor de acest tip este DQ sau dq (*Define Quad Word*).

- **Ten Bytes** (10 octeți)

Ocupă în memorie 10 locații succesive de 8 biți sau un registru intern al coprocesoarelor aritmetice 80x87. Semnificația este:

- număr întreg reprezentat ca secvență de cifre BCD cu semn explicit;
- număr real reprezentat în precizie extinsă.

Directiva pentru definire este DT sau dt (*Define Ten Word*)

În cazul reprezentării întregilor ca secvență de cifre BCD, se reprezintă două cifre pe octet; se rezervă o cifră BCD (4 biți) pentru semn, rezultând 19 cifre zecimale şi semn. Asambloarele acceptă şi numere de 20 de cifre zecimale dacă cifra cea mai semnificativă reprezentată pe 4 biți nu determină conflict cu bitul de semn.

Teoretic, valoarea maximă reprezentată este

+9 99 99 99 99 99 99 99 99 99,

iar valoarea minimă este

-9 99 99 99 99 99 99 99 99 99;

se acceptă însă şi valori de tipul:

+79 99 99 99 99 99 99 99 99 99 sau

- 79 99 99 99 99 99 99 99 99 99,

în care cifra cea mai semnificativă este reprezentată doar pe 3 biți, al patrulea bit fiind cel de semn.

În declarațiile de mai jos se utilizează toate tipurile de date descrise.

```
.model small
.data
      b1    db    -1,  10,  17H,  0FFH
      b2    db    'a', 'b'
      b3    db    "abcdef" , 0

      w1    dw    1234H, -1, 'AB'
      w2    dw    w1

      d1    dd    12345678H, -1
      d2    dd    1.0, -1.0, 0.5
      d3    dd    d1
```

```
        q1      dq      1000000000000002H, -1
        q2      dq      1.0, -1.0
        t1      dt      1234567890000012345
        t2      dt      -1234567890000012345
        t3      dt      9999999999999999999
        t4      dt      -9999999999999999999
        t5      dt      7999999999999999999
        t6      dt      -7999999999999999999
        t7      dt      1.0
    end
```

Liniile care încep cu punct sunt directive care stabilesc modelul de memorie (.model) respectiv un segment de date (.data). Directiva end marchează sfârşitul programului.

Se definesc date cu cele 5 tipuri de directive (db, dw, dd, dq, dt) prin asociere cu nume simbolice b1, b2, . . . , w1, w2, . . ., etc.

După asamblare, fişierul listing are conținutul de mai jos.

```
Turbo Assembler Version 2.0  05/30/01 10:07:55  Page 1
    A.ASM
1   0000                                .model small
2   0000                                .data
3   0000 FF 0A 17 FF            b1      db    -1,  10,  17H,  0FFH
4   0004 61 62                  b2      db    'a', 'b'
5   0006 61 62 63 64 65 66 00           b3    db    "abcdef" , 0
6
7   000D 1234 FFFF 41 42                w1    dw    1234H,-1, 'AB'
8   0013 000Dr                  w2      dw    w1
9
10  0015 12345678 FFFFFFFF              d1    dd    12345678H, -1
11  001D 3F800000 BF800000+             d2    dd    1.0, -1.0, 0.5
12       3F000000
13  0029 00000015sr                     d3    dd    d1
14
15  002D 1000000000000002+
16       FFFFFFFFFFFFFFFF               q1    dq  1000000000000002H,-1
17  003D 3FF0000000000000+
18       BFF0000000000000               q2    dq    1.0, -1.0
19
20  004D 01234567890000012345           t1    dt    1234567890000012345
21  0057 81234567890000012345           t2    dt  -1234567890000012345
22  0061 09999999999999999999           t3    dt    9999999999999999999
23  006B 89999999999999999999           t4    dt -9999999999999999999
24  0075 79999999999999999999           t5    dt  7999999999999999999
25  007F F9999999999999999999           t6    dt -7999999999999999999
26  0089 3FFF8000000000000000           t7    dt    1.0
27                              end
```

Prima coloană conţine numărul liniei din fişierul sursă. Cele 4 caractere grupate, de pe fiecare linie, reprezintă adresa de memorie (offset-ul) din interiorul segmentului de date iar câmpurile ce urmează sunt câmpurile de date corespunzătoare fiecărei linii din programul sursă. Dacă datele sunt listate pe mai multe rânduri, apare semnul + pentru a arăta continuarea. Simbolurile r şi s indică o adresă **r**elativă (deplasament) sau de **s**egment. De exemplu, variabila d3 conţine adresa completă (32 de biţi) a variabilei d1: adresa de segment s = 0000 şi adresa relativă r = 0015H.

La întregii BCD pe 10 octeţi, se observă memorarea explicită a bitului de semn: reprezentările pentru t1 şi t2 diferă numai prin bitul de semn.

Listingul nu indică adresa pentru toţi octeţii de date, ci numai adresa primului octet de date al liniei. Listingul următor permite vizualizarea zonei de memorie în care a fost stocat modulul de date de mai sus:

```
ds:0000 FF 0A 17 FF 61 62 61 62 63 64 65 66 00 34 12 FF
ds:0010 FF 42 41 0D 00 78 56 34 12 FF FF FF FF 00 00 80
ds:0020 3F 00 00 80 BF 00 00 00 3F 15 00 68 53 02 00 00
ds:0030 00 00 00 00 10 FF FF FF FF FF FF FF FF 00 00 00
ds:0040 00 00 00 F0 3F 00 00 00 00 00 00 F0 BF 45 23 01
ds:0050 00 00 89 67 45 23 01 45 23 01 00 00 89 67 45 23
ds:0060 81 99 99 99 99 99 99 99 99 99 09 99 99 99 99 99
ds:0070 99 99 99 99 89 99 99 99 99 99 99 99 99 99 79 99
ds:0080 99 99 99 99 99 99 99 99 F9
```

Acest listing permite observarea modului în care sunt memorate variabilele pe mai mulţi octeţi. De exmplu, variabila pe 4 octeţi,
d1= 1234 5678 H este memorată cu octeţii mai semnificativi la adrese mari:

0015: **78** 0016: **56** 0017: **34** 0018: **12**

Similar, variabila de tip word 'AB' (aflată la adresa 11H) este memorată prin secvenţa de octeţi **42 41**.

Listingul se obţine la încărcarea modulului de date în memorie, când adresele de segment sunt reale, adică ele corespund unor adrese fizice concrete. De exemplu, adresa variabilei d1 (dată de d3) apare în listing **5368:0015**, adică adresa de segment este 5368H iar deplasamentul este 0015H; rezultă adresa fizică:

```
5  3  6  8  0 +
   0  0  1  5
-------------
5  3  6  9  5 H
```

Variabila t1 (de 10 octeți) începe la adresa 004DH şi este reprezentată prin secvența: `45 23 01 00 00 89 67 45 23 01`, cu cifrele mai puțin semnificative la adrese mici.

14.2 Programe

Pentru conceperea, dezvoltarea şi rularea programelor în limbaj de asamblare, sunt necesare instrumente software adecvate. De exemplu, pe un calculator compatibil IBM - PC, cu sistem de operare DOS, se pot utiliza produsele Borland:

TASM (Turbo Assembler) - traduce instrucțiunile în cod maşină;

TLINK (Turbo Linker) - editorul de legături;

TLIB (Turbo Librarian) - bibliotecarul;

TD (Turbo Debugger) - program depanator.

Se utilizează extensiile implicite ale fişierelor: **.ASM** pentru fişiere sursă, **.OBJ** pentru fişere obiect, **.EXE** sau **.COM** pentru fişiere executabile.

În limbajul de asamblare lipsesc instrucțiunile de intrare - ieşire de nivel înalt ca de exemplu READ şi WRITE în Pascal.

Pentru simularea unor asemenea instrucțiuni se scriu proceduri şi macroinstrucțiuni stocate în fişierul IO.ASM (fişier sursă) şi IO.H (fişier header).

Aceste fişiere asigură următoarele operații de bază:

- introducerea şi afişarea caracterelor de la tastatură;
- afişarea unor mesaje imediate;
- introducerea şi afişarea numerelor întregi pe 16 biți cu sau fără semn;
- inițializarea registrelor DS şi ES la intrarea în program;
- terminarea programului cu ieşire în sistemul de operare.

Structura unui program în limbaj de asamblare:

```
.model    MMMMM
include   io.h
.stack    NNNN
.data
          ; definiții de date
.code
          ; definiții de proceduri
start:
     init_ds_es
          ; program principal
     exit_dos
end  start
```

S-au utilizat următoarele notații:

- **MMMMM** este modelul de memorie, care poate fi: tiny, small, medium, compact, large sau huge; modelele uzuale sunt small şi large, în care toate adresele, procedurile, salturile şi revenirile din proceduri sunt implicit de tip NEAR, respectiv de tip FAR.
- **NNNN** este dimensiunea rezervată segmentului stivă; o valoare uzuală este 1024
- **include io.h** este o directivă care include în textul sursă fişierul io.h care trebuie să fie în acelaşi director cu fişierul sursă;
- **init_ds_es** este o macroinstrucțiune care inițializează registrele DS şi ES cu adresele segmentelor de date; registrele CS şi SS sunt inițializate automat la încărcarea programului executabil de pe disc;
- **exit_dos** este o macroinstrucțiune (definită în io.h) care determină terminarea programului şi revenirea în sistemul de operare DOS;
- **start** este o etichetă care marchează începutul programului principal;
- **end start** este o directivă care marchează sfârşitul programului principal al cărui început este la eticheta start.

O altă variantă este scrierea programului principal sub forma unei proceduri (având, de exemplu, numele *_main*) şi precizarea punctului de start prin numele procedurii:

```
.model      MMMMM
include     io.h
.stackNNNN
.data
            ; definiții de date
.code
            ; definiții de proceduri
_main proc
      init_ds_es
            ; program principal
      exit_dos
_main endp
end  _main
```

Un modul de program, care nu este de program principal (deci conține definiții de date şi/sau proceduri), nu are etichetă în directiva end. Într-o aplicație dezvoltată modular (în mai multe fişiere sursă), un singur modul poate fi modul de program principal.

Asamblorul nu face deosebire între literele mici şi mari; de obicei programele se scriu cu litere mici iar cu litere mari unele directive şi tipuri de date definite de utilizator, pentru a fi mai vizibile.

Asamblarea unui fişier sursă se face cu una din comenzile:

```
C:\> tasm nume.asm
```

C:\> tasm nume

care are ca efect generarea unui fişier obiect NUME.OBJ.

Dacă este necesar şi un fişier listing, se dă cmanda:

C:\> tasm nume , ,nume

şi ca urmare rezultă şi un fişier listing, NUME.LST.

Operaţia de asamblare se face pentru fiecare fişier sursă în parte. Legarea între ele a modulelor obiect se face cu comanda:

C:\> tlink nume_1 nume_2 nume_3 . . . [, nume_exe] [/v]

în care parantezele drepte indică parametrii opţionali; nume_1, nume_2, . . . sunt nume date de programator diverselor module obiect, ce trebuie legate între ele; nume_exe este numele fişierului executabil rezultat în urma operaţiei de linkeditare (dacă lipseşte din listă, se consideră numele primului modul obiect) iar /v este o opţiune de depanare simbolică (se introduce dacă dorim să executăm programul sub controlul depanatorului TD). Ca efect al comenzii, rezultă un fişier executabil.

Când există un singur modul sursă, comanda de linkeditare va fi:

C:\> tlink nume io

prin care se leagă şi modulul io.obj, care conţine procedurile de intrare / ieşire.

Definiţia unui şir constant sau rezervarea de spaţiu pentru un şir variabil, se pot face prin directiva Define Byte (constantele simbolice cr şi lf sunt definite în fişierul io.h). Definiţia unui întreg sau rezervarea de spaţiu pentru un întreg se face cu directiva Define Word. Definirea de spaţiu la nivel de caracter se face cu directiva Define Byte.

```
.data
        sir_a       db    'un sir de caractere', cr, lf, 0
        sir_b       db    70 dup (0)
        numar_1     dw    -200
        numar_2     dw    ?
        u_1         dw    0FFFFH
        u_2         dw    -1
        car_1       db    'B'
        car_2       db    ?
```

Caracterul '0' este utilizat ca marcator de sfârşit de şir de caractere.

Afişarea unui şir de caractere pe ecran se poate face cu macroinstrucţiunea puts (Put String):

```
puts  sir_a         ; prima formă de utilizare
lea   si, sir_a     ; a doua formă de utilizare
puts  [si]          ;
```

Citirea unui şir de caractere de la tastatură se poate face cu macroinstrucţiunea gets (Get String):

```
gets  sir_a        ; prima formă de utilizare
lea   bx, sir_a    ; a doua formă de utilizare
gets  [bx]         ;
```

Afişarea unui şir constant de caractere (mesaj pe ecran) se poate face cu macroinstrucţiunea putsi (Put String Immediate), care nu necesită definirea şirului şi nici prezenţa explicită a caracterului terminal '0'.

Introducerea unui întreg cu sau fără semn se poate face cu macroinstrucţiunea geti (Get Integer), fără parametri, care pune întregul citit in registrul AX.

Afişarea unui întreg cu sau fără semn se poate face cu macroinstrucţiunile puti (Put Integer) sau putu (Put Unsigned).

```
geti                    ; Citeste intreg si il incarca in AX
puti   ax               ; Afiseaza intregul din ax
mov    numar_1, ax      ; Depune cuvant in ax
puti   numar_1          ; Afiseaza intreg cu semn din memorie
putu   u_1              ; Afiseaza ca numar fara semn
lea    di, numar_2
puti   [di]             ; Afiseaza ca numar cu semn
putu   [di]             ; Afiseaza ca numar fara semn
```

Introducerea unui caracter de la tastatură se poate face cu macroinstrucţiunea getc (Get Character), care pune caracterul în registrul AL iar afişarea unui caracter se poate face cu macroinstrucţiunea putc (Put Character):

```
putc 'A'
putc car_1
lea  bx, car_2
putc [bx]
```

Toate macroinstrucţiunile descrise mai sus conservă registrele procesorului, deci nu sunt necesare salvări şi restaurări explicite.

În programul următor sunt utilizate macroinstrucţiunile de introducere şi afişare date; se fac următoarele operaţii:

- se citesc de la tastatură cel mult 20 de întregi cu semn;
- se afişează valorile introduse;
- se sortează crescător aceste valori;
- se afişează valorile sortate.

Se consideră modelul de memorie large, adică toate adresele sunt implicit de 32 de biți. Pentru sortare se utilizează metoda bulelor (indicii tabloului a sunt în domeniul 0, . . . ,n-1). Se compară două câte două elementele tabloului; dacă nu sunt în ordinea dorită, se inversează pozițiile lor în tablou.

Algoritmul de sortare, în Pascal, este:

```
for  i = 1  to  n -1
for  j = n-1  downto  i
      if  ( a[ j -1] > a[ j ])
      schimba  (a[ j ], a[ j - 1]);
```

Programul demonstrativ este următorul:

```
.model          large
include         io.h
.stack      1024
.data
            vec   dw    20 dup      (?)
            n     dw    ?
.code
tipvec      proc  far
      ; procedura de afisare a vectorului
      ; Date de intrare:
      ; ds:si = adresa primului element al vectorului
      ; cx = numar de elemente

      jcxz        tipend      ; Nu sunt date de afisat
tip:
      puti  [si]              ; Afisare intreg cu semn
      putsi <' '>             ; Spatiu
      add   si, 2             ; Actualizare adresa
      loop  tip               ; Bucla repetitiva cu contor cx
tipend:
      ret
tipvec endp
bubble      proc far
      ; Procedura de sortare
      ; Date de intrare
      ; ds:bx = adresa primului element al tabloului
      ; cx = dimensiunea tabloului (numarul de elemente)
      ; Variabile i : asociata cu si
      ; Variabila j : asociata cu di

      cmp cx, 1
      jbe   algend                ; sortarea nu are obiect n=1
      mov   si, 1                 ; i = 1
fori:
```

```
        mov   di, cx
        dec   di                        ; j = n - 1
forj:
        shl   di, 1                     ; intregii sunt pe doi octeti
        mov   ax, [bx][di-2]            ; a[j - 1]
        cmp   ax, [bx][di]              ; compara cu a[j]
        jle   nextj                     ; mai mic sau egal
        xchg  ax, [bx][di]              ; schimba a[j]
        mov   [bx][di-2],  ax           ; cu a[j-1]
nextj:

        shr   di, 1                     ; refacerea indicelui
        dec   di                        ; bucla for de tip downto
        cmp   di, si
        jae   forj                      ; cat timp j >= i
nexti:
        inc   si                        ; bucla for de tip to
        cmp   si, cx
        jb    fori                      ; cat timp i < n
algend:
        ret
bubble endp

;       Programul principal
start:
        init_ds_es
        putsi       <'Introduceti datele', cr, lf>
        mov   cx, 20                    ; numarul maxim de elemente
        lea   bx, vec                   ; adresa tabloului
iar:
        geti                            ; citire intreg cu semn
        test  ax, ax                    ; este 0 ?
        jz    gata                      ; daca Da, atunci gata
        mov   [bx], ax                  ; depunere in tablou
        add   bx, 2                     ; actualizare adresa
        loop  iar                       ; bucla repetitiva dupa cx
gata:
        mov   ax, 20                    ; calculeaza numarul de
        sub   ax, cx                    ; elemente introduse
        mov   n, ax
        putsi       <'Vector nesortat', cr, lf>
        lea   si, vec                   ; adresa tablou
        mov   cx, n                     ; numar de elemente
        call  tipvec                    ; afisare tablou nesortat

        lea   bx, vec                   ; adresa tablou
        mov   cx, n                     ; numar de elemente
        call  bubble                    ; sortare tablou

        putsi       <cr, lf, 'Vector sortat', cr, lf>
```

```
        lea    si, vec                 ; adresa tablou
        mov    cx, n                   ; numar de elemente
        call   tipvec                  ; afisare tablou sortat
        exit_dos                       ; iesire in sistemul de operare DOS
end     start
```

În segmentul de date se rezervă spațiu pentru tabloul vec de maxim 20 de numere întregi şi pentru numărul n = dimensiunea tabloului.

Procedura tipvec primeşte în SI adresa tabloului şi în CX numărul de elemente, realizând afişarea pe ecran a elementelor, considerate întregi cu semn.

Procedura bubble conține algoritmul de sortare; indicii i şi j sunt prezenți în registrele SI şi DI. Elementele tabloului fiind pe 2 octeți, pentru adresarea memoriei se înmulțeşte di cu 2 (shl di, 1). Astfel, elementul de indice 0 se va afla la deplasament 0, cel de indice 1 la deplasament 2 etc. Înmulțirea se face prin deplasare logică la stânga iar refacerea prin deplasare logică la dreapta.

Secvența de interschimbare a două elemente din memorie se face cu xchg, cu ajutorul registrului ax.

Comparațiile se fac în mod diferit: pentru operanzi fără semn (i, j) se utilizează instrucțiunile de salt condiționat de tip "*Above*" sau "*Below*" iar pentru operanzi cu semn (elementele tabloului), cele de tip "*Greater*" sau "*Less*".

Programul principal începe cu bucla de citire a datelor. La ieşirea din buclă se calculează numărul de elemente efectiv introduse, ca diferență între numărul maxim admis (20) şi valoarea curentă din CX. Acest număr se depune în variabila n pentru utilizări viitoare.

Utilizând procedura tipvec se afişează vectorul nesortat, apoi acesta este sortat cu procedura bubble şi afişat tot cu tipvec.

Programul ilustrează modul de utilizare a macroinstrucțiunilor, construcția şi utilizarea procedurilor şi structura generală a unui program în asamblare.

14.2.1. Directive de asamblare

Sunt comenzi către programul asamblor, efectul lor manifestându-se exclusiv în faza de asamblare. Prin intermediul directivelor se definesc date, etichete şi proceduri, se structurează segmente, se definesc şi se utilizează macroinstrucțiuni, se controlează în general procesul de asamblare.

Segmentare. Definirea segmentelor

Un modul de program în limbaj de asamblare poate utiliza:

- o porțiune dintr-un segment;
- un segment;
- porțiuni de segmente diferite;
- mai multe segmente.

Directivele *SEGMENT* şi *END*

Instrucțiunile şi datele trebuie să fie organizate în segmente de memorie. Directiva SEGMENT determină:

- numele segmentului;
- alinierea;
- combinarea cu alte segmente;
- continuitatea (adiacența) segmentelor.

Forma generală a directivei SEGMENT este:

```
nume SEGMENT  [tip_aliniere] [tip_combinare] ['nume_clasa']
.
.
nume ENDS
```

Parametrii din paranteze sunt opționali; dacă există, trebuie să fie specificați în ordinea indicată. Semnificația parametrilor este:

- **tip_aliniere** - specifică la ce limită va fi încărcat segmentul în memorie:

- PARA (implicit) - aliniere la paragraf: segmentul fizic, adică adresa pe 20 de biți, va fi încărcată la prima adresă absolută divizibilă prin 16

x x x x 0 H.

- BYTE - fără aliniere: segmentul se încarcă la următorul octet liber.
- WORD - aliniere la cuvânt: segmentul se încarcă la prima adresă pară.
- DWORD - aliniere la dublu cuvânt: segmentul se încarcă la prima adresă divizibilă cu 4.
- PAGE - aliniere la pagină: segmentul se va încărca la prima adresă divizibilă cu 256).

Exemple:

```
DATA_1    SEGMENT BYTE
          x     db    7     dup   (?)
DATA_1    ENDS

DATA_2    SEGMENT WORD
          y     dw    512   dup   (?)
          z     dw    ?
DATA_2    ENDS
     DATA_3    SEGMENT PARA
          zz    db    8     dup   (?)
DATA_3    ENDS
```

Dacă prima adresă disponibilă este 2000H, cele trei segmente vor fi încărcate la următoarele adrese fizice:

DATA_1 2000:0 . . . 2000:6
DATA_2 2000:8 . . . 2000:406
DATA_3 2041:0 . . . 2041:7

Adresele de segment se obțin din primele 4 cifre ale adresei fizice de început. Offset - urile la execuție diferă în general de cele din programul sursă; de exemplu, pentru variabila y, offset - ul din program este 0 iar la execuție este 8.

Operatorul OFFSET, care furnizează deplasamentul unei variabile sau al unei etichete în cadrul unui segment, va produce offsetul de la execuție.

Instrucțiunea:

```
mov cx,  OFFSET  y
```

va încărca în CX valoarea 8.

Dacă dorim ca offset - ul la asamblare să coincidă cu offset - ul la execuție, putem folosi tipul de aliniere PARA (adresa fizică de bază a segmentului se termină cu 0).

- **tip_combinare** - specifică dacă segmentul respectiv se combină cu alte segmente la link-editare şi modul în care se combină; variantele sunt:

  - necombinabil (implicit) - nu se scrie nimic;

  - PUBLIC - segmentul curent va fi concatenat cu alte segmente cu acelaşi nume şi cu atributul PUBLIC. Aceste segmente pot fi în alte module de program. Se va forma în final un singur segment cu numele respectiv, cu o unică adresă de început şi cu lungimea egală cu suma lungimilor segmentelor cu acelaşi nume.

  - COMMON - specifică faptul că segmentul curent şi toate segmentele cu acelaşi nume şi cu tipul COMMUN se vor suprapune în memorie, adică vor începe la aceeaşi adresă fizică; lungimea unui segment COMMUN este cea mai mare dintre lungimile segmentelor componente.

  - STACK - marchează segmentul stivă al programului; dacă sunt mai multe segmente cu tipul STACK, ele vor fi tratate ca PUBLIC. În exemplul următor se defineşte un segment stivă şi se face şi o inițializare explicită a lui.

```
stiva           SEGMENT  STACK
                db    512   dup (?)
                stiva_index label  WORD
stiva           ENDS
cod        SEGMENT
```

```
                    mov  ax,   stiva
                    mov  ss,   ax
                    mov  sp,   OFFSET   stiva_index
cod         ENDS
```

- AT <expresie> specifică faptul că segmentul va fi plasat la o adresă fizică absolută de memorie. Exemplul următor arată definirea explicită a tabelei vectorilor de întrerupere.

```
INTR_TAB        SEGMENT AT 0
          dd    PROC_NIV_0
          dd    PROC_NIV_1
          . . . . . .
INTR_TAB        ENDS
```

Aceste forme se utilizează când programul este dezvoltat pentru echipamente dedicate. În cazul unui calculator de tip IBM - PC, există funcții DOS pentru accesul la tabela de întreruperi.

- **'nume_clasă'** - specifică un nume de clasă pentru segment, extinzând astfel numele segmentului; ca nume de clasă, se folosesc de obicei 'code', 'data', 'stack'. De exemplu, dacă segmentul are şi atributul 'nume clasă', atunci atributele COMMON sau PUBLIC vor acționa numai asupra segmentelor cu acelaşi nume şi acelaşi nume de clasă.

1. Directive pentru definirea simplificată a segmentelor

Au fost introduse în variantele recente ale asambloarelor. Avantajul major este că se respectă acelaşi format ca la programele în limbaj de nivel înalt, adică se vor genera segmente cu nume şi atribute identice cu cele generate de compilatoarele de limbaje de nivel înalt. Toate directivele încep cu un punct.

Modele de memorie

.model <tip>

unde tip poate fi: tiny, small, medium, large sau huge. Semnificația lor este:

- tiny - toate segmentele (cod, date, stivă) se pot genera într-un spațiu de 64kB şi formează un singur grup de segmente; se foloseşte la programele de tip COM; toate salturile, apelurile şi definițiile de proceduri sunt implicit de tip NEAR.
- small - datele şi stiva sunt grupate într-un singur segment iar codul în segment separat; fiecare din cele două nu poate depăşi 64kB. Toate

salturile, apelurile şi definiţiile de proceduri sunt implicit de tip NEAR.

- medium - datele şi stiva sunt grupate într-un singur segment (de cel mult 64kB) dar codul poate fi în mai multe segmente separate, deci poate depăşi 64kB. Toate salturile, apelurile şi definiţiile de proceduri sunt implicit de tip FAR.
- compact - codul generat ocupă cel mult 64kB dar datele şi stiva sunt în segmente separate (pot depăşi 64 kB). Apelurile şi salturile sunt implicit de tip NEAR. Se utilizează adrese complete (segment şi offset) când se accesează date definite în alte segmente.
- large - atât datele cât şi codul generat pot depăşi 64kB.
- huge - asemănător modelului large, dar se utilizează adrese complete normalizate în care offset - ul este redus a minim (în domeniul 0 - 15), ceea ce face ca o adresă fizică să fie descrisă într-un mod unic (segment, offset). La modelele compact şi large, o structură compactă de date (tablou) nu poate depăşi limitele unui segment fizic (64kB); la modelul huge, nu mai există această restricţie.

Se utilizează următoarea terminologie:

- modele de date reduse: small, compact;
- modele de cod redus: small, medium;
- modele de date extinse: medium, large, huge;
- modele de cod extins: compact, large, huge.

Definirea segmentelor

Formele generale sunt:

```
.stack dimensiune
.code [nume]
.data
.data?                  ; date neiniţializate
.fardata [nume]         ; segmente de date utilizate
.fardata [nume]         ; prin adrese complete
.const                  ; definire de constante
```

Dacă parametrul [nume] lipseşte, se atribuie nume implicite segmentelor generate, astfel:

Segment	Nume implicit	Segment	Nume implicit
.fardata	_FAR_DATA	.fardata?	_FAR_BSS

.data?	_BSS	.const	CONST
.data	DATA	.stack	STACK
.code	TEXT, nume_fişier_sursă_TEXT (la modele de cod mare)		

2. Directive pentru legarea modulelor

Când programul se compune din mai multe module asamblate separat, este necesar să se specifice simbolurile care sunt definite într-un modul şi utilizate în alte module. Aceste simboluri sunt nume de variabile, etichete sau nume de proceduri. În mod normal, un simbol este vizibil numai în modulul în care a fost definit. Cele vizibile în mai multe module de program se numesc simboluri globale. Ele sunt :

- simboluri **publice** - se declară ca publice în modulul în care sunt definite şi pot fi utilizate şi în alte module;
- simboluri **externe** - se declară ca externe în modulele în care se folosesc, ele fiind definite în alte module.

Un simbol global trebuie, aşadar, declarat ca public în modulul în care este definit şi ca extern în modulele în care se utilizează, altele decât cel în care a fost definit.

Declararea unui simbol ca public, respectiv extern, se face cu directivele PUBLIC şi EXTERN.

Directiva PUBLIC are forma generală:

```
PUBLIC    nume,      nume, . . .
```

unde lista de nume conţine nume de variabile, etichete, proceduri sau constante numerice simbolice.

Directiva EXTERN are forma generală:

```
EXTRN     nume: tip,  nume: tip, . . .
```

în care <tip> precizează tipul simbolului, care poate fi:

- BYTE, WORD, DWORD, QWORD, când simbolul este o variabilă;
- NEAR, FAR, când simbolul este etichetă sau nume de procedură;
- ABS, când simbolul este o constantă numerică simbolică.

3. Directiva END

Marchează sfârşitul logic al unui modul de program şi este obligatorie în toate modulele. Ceea se află după END este ignorat de programul asamblor. Sintaxa este:

END [punct_start]

în care punct_start este o etichetă opțională sau un nume de procedură ce marchează punctul în care se transferă controlul după încărcarea programului în memorie. Într-o aplicație compusă din mai multe module şi care se constituie într-un unic program executabil, un singur modul trebuie să aibă punctul de start.

4. Contoare de locații şi directiva ORG

Contoarele de locații controlează procesul de asamblare, arătând *offset*-ul în cadrul segmentului curent la care se vor asambla instrucțiunea sau datele următoare. Un contor de locații poate fi accesat explicit prin simbolul $.

La prima utilizare a unui nume de segment, contorul de locații este inițializat cu zero. Dacă se revine într-un segment care a mai fost utilizat, contorul de locații revine la ultima valoare folosită în cadrul acelui segment, ca în exemplul următor:

```
.data                   ; $=0
     db   5    dup(?)   ; $=5
.code                   ; $=0
.data
     db   7             ; $=5
```

Contoarele de locații sunt utile la calculul unor deplasamente sau dimensiuni. În exemplul următor se defineşte un tablou de cuvinte TW şi o variabilă NW care conține numărul de cuvinte din tablou:

```
TW  dw   1, 2, 3, 4, 5, 6, 7, 8, 9
NW  dw   ($ - TW)/2
```

Expresia $ - TW reprezintă numărul de octeți de la adresa tabloului TW până la adresa curentă.

Prin împărțire la 2 se obține numărul de elemente din tablou.

Directiva ORG (*Origin* - Inițializează contorul de locații)

Directiva modifică explicit contorul de locații curent, având sintaxa:

ORG <expresie>

Exemplu:

```
ORG  $+7        ; Sare 7 octeți la asamblare
ORG  100H       ; Sare la offset-ul absolut 100H
```

5. Definirea şi inițializarea datelor

Asamblorul recunoaşte trei categorii sintactice de bază:

- constante;
- variabile;
- etichete (inclusiv nume de procedură).

Constantele pot fi absolute (numere) sau simbolice. Cele simbolice sunt nume generice asociate unor valori numerice.

Variabilele identifică datele din memorie iar etichetele identifică programe sau proceduri (cod).

Instrucțiuni ca MOV, ADD, MUL, etc. utilizează variabile şi constante iar cele de tipul JMP, CALL, utilizează etichete.

Variabilele şi etichetele sunt asociate cu anumite atribute cum ar fi segmentul în care sunt definite, offset-ul în cadrul segmentului etc.

6.Constante

Constantele numerice absolute pot fi:

- constante binare - se utilizează sufixul B sau b;
- constante octale - se utilizează sufixul O, Q, o sau q;
- constante zecimale - se utilizează sufixul D sau d;
- constante hexazecimale - se utilizează sufixul H sau h şi prefixul 0 dacă prima cifră este mai mare ca 9; pentru cifrele 10 . . . 15 se utilizează simbolurile A, B, C, D, E, F sau a, b, c, d, e, f.
- constante ASCII - se scrie unul sau mai multe caractere între semne apostrof sau ghilimele.

Constantele simbolice se definesc cu directiva EQU, cu sintaxa:

<nume> EQU <expresie>

De exemplu, liniile de program:

```
NR      EQU     0FFH
CONTOR  EQU     100
```

definesc constantele simbolice NR şi CONTOR cu valorile 0FFH, respectiv 100.

În program se pot utiliza constantele simbolice astfel definite, în orice context în care este permisă prezența unei valori numerice; la execuție, constantele simbolice vor fi înlocuite cu valorile prin care au fost definite.

7. Variabile

Pentru definirea variabilelor se utilizează directivele DB, DW, DD, DQ sau DT, care au fost definite la 'Tipuri de date'.

Sintaxa definirii variabilelor este:

<nume_var> directiva <lista_de_valori>

unde nume_var este identificatorul da variabilă iar lista_de _valori este lista valorilor inițiale, care poate conține:

- constante numerice absolute sau simbolice;
- simbolul ? - cu semnificația de 'locație neinițializată' dar rezervată;
- o adresă, adică un nume de variabilă sau de etichetă; se poate folosi la DW, DD;
- un şir de caractere ASCII;
- operatorul DUP( ) - repetarea de un număr de ori a expresiei din paranteză - care poate conține ca argument: o constantă numerică, o listă de valori, simbolul ? sau operatorul DUP( ).

Exemple de definiții:

```
var_1   db   2 dup ( 7, 3 dup ( 0 ))
var_2   db   1, 2, 3, ?, ?, ?
adr_1   dw   var_1
adr_2   dd   var_2
```

Prima definiție este echivalentă cu:

```
var_1   db   7, 0, 0, 0, 7, 0, 0, 0
```

Atributele datelor definite sunt: segment - cel curent, offset - cel curent.

Variabilele care se utilizează în programe pot fi simple sau indexate. La cele indexate trebuie să ținem seama de tipul de bază. De exemplu, după definirea tabloului

```
T    dw   10 dup ( ? )
```

elementele sale se accesează prin T[0], T[2], T[4], . . . , deoarece fiecare element este de tip word; T[1] este octetul superior al primului element.

8. **Definirea etichetelor**

Etichetele se utilizează pentru specificarea punctelor țintă la instrucțiunile de salt sau pentru o specificare alternativă a datelor. În ambele cazuri, numele etichetei este un nume simbolic asociat adresei curente de memorie. Atributele etichetelor sunt: segment, offset şi tip.

Modalități de definire:

- prin nume urmat de caracterul **:** se defineşte o etichetă de tip NEAR;
- prin directiva PROC - numele procedurii este interpretat ca o etichetă cu tipul derivat din tipul procedurii;
- prin utilizarea directivei LABEL, cu sintaxa:

<nume> LABEL <tip>

Dacă ceea ce urmează reprezintă instrucțiuni (cod), tipul etichetei va fi de regulă NEAR sau FAR şi eticheta va fi folosită ca punct țintă în instrucțiunile de tip JMP sau CALL. Dacă urmează definiții de date, tipul etichetei va fi de regulă BYTE, WORD, DWORD etc.

Operatori

Limbajul de asamblare dispune de operatori cu care se pot construi expresii de tip aritmetic şi logic. Expresiile sunt evaluate la asamblare, producând valori numerice.

Este esențială deosebirea dintre operațiile efectuate de instrucțiunile executabile (cod maşină) şi cele care se fac la asamblare.

9. Operatori aritmetici şi logici

Operatorii aritmetici sunt: +, - , * , / , MOD, SHL, SHR. Primii 4 au semnificațiile obişnuite; MOD produce restul la împărțire iar ultimii doi produc deplasare la stânga, respectiv la dreapta. De exemplu instrucțiunea:

```
mov bx,  1 SHL 3          ; 1 deplasat la stanga cu 3 biti
```

este echivalentă cu:

```
mov bx,  1000B
```

Operațiile aritmetice sunt evaluate la asamblare:

```
CONTOR  dw  ($ - TAB) / 2     ; contor va primi valoarea dată de
                              ; diferența dintre contorul curent
                              ; de locații şi adresa împărțită la 2
```

Operatorii logici sunt: NOT, AND, OR, XOR. Operațiile se execută la nivel de bit. Ei nu trebuie confundați cu instrucțiunile executabile cu acelaşi nume. Exemplu:

```
and ah,  (1 SHL 3) OR (1 SHL 6)
```

are ca efect încărcarea registrului ah cu 1001000B.

Operatorul de atribuire =

Defineşte constante simbolice, fiind similar cu EQU, cu deosebirea că permite modificarea valorii inițiale (redefinirea). Secvența:

```
NR  EQU 7
NR  EQU 1
```

produce eroare la asamblare, deoarece EQU nu permite modificarea lui NR=7, dar

```
NR = 7
NR = 1
```

este secvență fără erori.

Operatorul ' = ' este utilizat în special în definițiile de macroinstrucțiuni.

10. Operatori care întorc valori

Se aplică variabilelor şi etichetelor, întorcând valori asociate acestora.

- SEG - aplicat variabilelor sau etichetelor, furnizează adresa de segment asociată.
  Exemplu:
  ```
  mov ax,  SEG var_x
  mov ds,  ax
  ```
- OFFSET - similar cu SEG, furnizează însă offset-ul asociat variabilei sau etichetei. Exemplu:
  ```
  mov bx,  OFFSET var_x
  ```
- THIS - Creează un operand care are asociate o adresă de segment şi un offset identice cu cele ale contorului curent de locații. Sintaxa utilizării este:
  THIS <tip>

în care <tip> poate fi BYTE, WORD, DWORD, QWORD, TBYTE pentru definiții de date, respectiv NEAR, FAR pentru etichete. Operatorul THIS se utilizează de obicei cu directiva EQU. De exemplu, definiția constantei simbolice BETA:

```
BETA    EQU  THIS  WORD
```

este echivalentă cu definiția unei etichete

```
BETA    LABEL  WORD
```

- TYPE - Se aplică variabilelor şi etichetelor, întorcând tipul acestora, exprimat în număr de octeți pentru variabile (1, 2, 4, 8, 10 etc.) şi NEAR, FAR pentru etichete.
- LENGTH - se aplică numai variabilelor şi întoarce numărul de elemente definite în variabila respectivă.
  De exemplu definiția:
  ```
  x    dw   100 dup (?)
  ```

produce pentru LENGTH x valoarea 100.

- SIZE - se aplică numai variabilelor şi întoarce dimensiunea în octeți a variabilei respective. Corespunzător definiției de mai sus, expresia SIZE x are valoarea 200 (deoarece x conține 100 de cuvinte x 2 octeți = 200 octeți).