UNIUNEA EUROPEANĂ
GUVERNUL ROMÂNIEI
Instrumente Structurale 2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Programare în limbaj de asamblare

45. Setul de instrucțiuni: instrucțiuni de transfer, aritmetice, de prelucrare la nivel de bit.

Setul de instrucţiuni

Din motive descriptive, setul de instrucţiuni de la procesorul 286 este împărţit în trei subseturi distincte:

- setul instrucţiunilor de bază;
- setul extins de instrucţiuni;
- setul de instrucţiuni de control sistem.

Setul de instrucţiuni de bază

Aceste instrucţiuni sunt grupate în şase tipuri, şi anume:

- transfer date;
- aritmetice;
- operare pe bit (logice, deplasare, rotire);
- operare pe şiruri;
- transfer control program;
- control procesor.

Aproape fiecare instrucţiune poate opera fie pe octet, fie pe cuvânt (sau dublu cuvânt, la 386/486). Operanzii instrucţiunilor pot fi interschimbabili, cu excepţia datelor imediate care pot fi numai „sursă", nu şi „destinaţie". Variabilele din memorie pot fi operate direct în memorie, fără să fie mutate în registre.

Setul de instrucţiuni poate fi considerat ca având două niveluri: nivel de asamblare şi nivel maşină. Pentru programatorul în limbaj de asamblare, procesorul apare ca având circa o sută de instrucţiuni. O instrucţiune în limbaj de asamblare (de ex. MOV) corespunde, de fapt, la mai multe forme de instrucţiuni maşină (pt. MOV sunt circa 30 de tipuri, în funcţie de tipul operandului: octet/cuvânt, registru/memorie, dată imediată etc.). La nivel maşină există circa trei sute de instrucţiuni. Instrucţiunile nivelului de asamblare simplifică viziunea programatorului asupra setului de instrucţiuni: programatorul scrie instrucţiunea (de ex. INC, de incrementare a unui operand – 8/16 biţi, registru sau locaţie de memorie), asamblorul examinează operandul şi determină instrucţiunea nivel maşină ce trebuie generată.

Instrucţiuni de transfer date

La rândul lor, aceste instrucţiuni sunt împărţite în patru clase:

- instrucţiuni de transfer cu scop general („clasice");
- instrucţiuni specifice acumulatorului;
- instrucţiuni de transfer adrese obiect;
- instrucţiuni de transfer registru indicatori.

Aceste instrucţiuni nu modifică indicatorii de condiţii, decât cu unele excepţii care vor fi menţionate corespunzător.

Instrucţiuni de transfer cu scop general

MOV <dest>,<sursa>

Transferă un octet/cuvânt (sau dublu cuvânt) de la sursă la destinaţie, fără a modifica nici un indicator din registrul respectiv. În funcţie de sursă şi destinaţie, poate avea formele:

```
MOV     <reg>,<reg>
MOV     <reg_16>,<reg_seg>
MOV     <reg_seg>,<reg_16>
```

Exemple de instrucțiuni:

```
mov     ax,bx
mov     al,ch
mov     ax,cs
mov     ds,ax
```

Registrul segment (reg_seg) CS nu poate fi destinație şi nici nu se pot transfera date direct între două registre segment.

```
MOV     <reg>,<mem>
MOV     <mem>,<reg>
```

Exemple de instrucțiuni:

```
mov     vector[bx][si],ax       ; vector este de tip 'word'
mov     sp,varf_stiva
mov     bl,byte ptr vector[bp][di]
```

Nu se pot transfera date direct între două locații de memorie.

```
MOV     <reg_seg>,<mem_16>
MOV     <mem_16>,<reg_seg>
```

Exemple:

```
mov     ds,adr_baza_seg
mov     [bx].salv_seg,cs
```

Realizează transferul între o locație de memorie de 16 biți şi un registru segment; CS nu poate fi destinație.

```
MOV     <reg>,<data_imed>
MOV     <mem>,<data_imed>
```

Exemple:

```
mov     ax,0
mov     cl,4
mov     byte ptr [si],2ch
mov     alfa[bp][si],100
```

Această instrucțiune nu se poate executa utilizând registrele segment.

Instrucțiuni de conversie (extensie de semn)

Aceste instrucțiuni realizează extensia de semn a acumulatorului (AL, AX, EAX) în extensia sa (AH, DX sau EAX, EDX). Nu se modifică nici un indicator. Aceste instrucțiuni pot fi folosite pentru a furniza un deîmpărțit de lungime dublă înainte de a efectua o împărțire cu semn.

CBW (Convert Byte to Word)

Realizează extensia de semn a lui AL în registrul AH.

CWDE **(Convert Word to Double word Extended)**

Realizează extensia de semn a lui AX în registrul EAX.

CWD (Convert Word to Double word)

Realizează extensia de semn a lui AX în registrul DX.

CDQ (Convert Double word to Quad word)

Realizează extensia de semn a lui EAX în registrul EDX.

La 386/486 mai există instrucțiunile de transfer cu extensia bitului de semn sau cu extensie de zerouri:

MOVSX <dest>,<sursa> (MOVe with Sign eXtension)

Realizează transferul operandului sursă la destinație, cu extensia bitului de semn, în octeții superiori. Destinația poate fi un registru de 16/32 biți, iar sursa poate fi un registru sau o locație de memorie de 8/16 biți. Deci, se poate face transferul cu extensia bitului de semn a unui:

■ octet la un cuvânt sau dublu cuvânt:

```
MOVSX    reg_16,reg/mem_8
MOVSX    reg_32,reg/mem_8
```

■ cuvânt la un dublu cuvânt:

```
MOVSX    reg_32,reg/mem_16
```

Nu este afectat nici un indicator.

MOVZX <dest>,<sursa> (MOVe with Zero eXtension)

Transferă sursa (8/16 biți) la o destinație (16/32 biți) şi extinde cu zerouri, la stânga, în octeții superiori, pentru a completa destinația la dimensiunea sa. Deci se poate face transferul cu extensia de zerouri a unui:

■ octet la un cuvânt sau dublu cuvânt:

```
MOVZX       reg_16,reg/mem_8
MOVZX       reg_32,reg/mem_8
```

■ cuvânt la un dublu cuvânt:

```
MOVZX       reg_32,reg/mem_16
```

Nu este afectat nici un indicator.

Instrucțiuni de interschimbare a datelor

XCHG <dest>,<sursa> (eXCHanGe)

Instrucțiunea schimbă, între ele, conținuturile celor doi operanzi, astfel încât fiecare dintre ei este atât sursă, cât şi destinație.

Instrucțiunea poate fi de forma:

```
XCHG    <reg>,<reg>
XCHG    <reg>,<mem>
XCHG    <mem>,<reg>
```

dar nu poate opera cu registre segment.

Instrucțiuni de transfer cu stiva

Stiva este organizată pe cuvinte, iar la 386/486 ea poate fi referită pe dublu cuvânt. Stiva este folosită pentru

- salvarea/refacerea adresei de revenire pentru apel, respectiv revenire din apel de procedură sau pentru întreruperi, respectiv revenire din întreruperi, când se salvează/reface pe lângă adresa de revenire a programului întrerupt şi registrul indicatori; administrarea stivei pentru astfel de operații este realizată, în mod automat, de către procesor;
- salvarea/refacerea conținutului unor resurse (registre, memorie etc.) la intrarea într-o procedură, respectiv la ieşirea din aceasta;
- pentru transferul parametrilor de intrare/ieşire între o procedură şi programul apelant, precum şi pentru alocarea dinamică de memorie (adică pe durata execuției programului) pentru variabilele locale unei proceduri.

PUSH <sursa> – depune in varful stivei <sursa>.
(SP) ← (SP) – 2
((SP)+1:(SP)) ← (sursa)

Operandul sursă poate fi un registru general sau o locație de memorie de 16 biți, respectiv un registru segment.

Exemple:

```
push    si
push    cs
push    beta[bx][si]
```

Începând cu procesorul 80386, instrucțiunea PUSH (E)SP depune în stivă valoarea registrului (E)SP, aşa cum era aceasta înaintea execuției instrucțiunii (deci fără a fi decrementată). Aceasta diferă de procesoarele anterioare (8086/186/286), unde instrucțiunea PUSH SP depune în stivă noua valoare a registrului SP (decrementată cu 2).

POP <dest> – extrage din varful stivei si duce la dest.
(dest) ← ((SP)+1:(SP))
(SP) ← (SP) + 2

Exemple:

```
pop     bx
pop     ds              ; cs nu poate fi destinatie
pop     beta[bp][di]
```

Încărcarea registrului CS prin intermediul stivei se poate face numai la execuția unei instrucțiuni de revenire dintr-o procedură, în context far, sau la revenirea dintr-o întrerupere.

Informațiile din stivă vor fi extrase în ordinea inversă celei în care au fost introduse:

```
; salvam registrele: ax, bx, cx
        push    ax
        push    bx
        push    cx
; refacem aceleasi registre cu aceleasi valori:
        pop     cx
        pop     bx
        pop     ax
```

Accesul la informația din stivă se poate face fără descărcarea stivei, utilizând adresarea bazată astfel:

```
; memorare informatii in stiva:
        mov     bp,sp           ; memorare „varful“ stivei
        push    ax              ; se va depune la adresa [bp-2]
        push    bx              ; [bp-4]
        push    cx              ; [bp-6]
; accesul la informatii se poate face, cu BP, in orice ordine
        mov     ax,[bp-2]
        mov     bx,[bp-4]
        mov     cx,[bp-6]
; descarcarea stivei se poate face modificand valoarea lui SP
        add             sp,6
```

Dacă dorim să nu modificăm valoarea lui BP, va trebui să-l salvăm şi pe acesta în stivă, ceea ce se întâmplă în mod frecvent astfel:

```
; depunerea de parametri in stiva, inainte de apelul procedurii:
        push    ax
        push    bx
        push    cx
; preluarea parametrilor din stiva, in cadrul procedurii:
; (s-a facut abstractie, in acest exemplu, de salvarea din stiva
; a adresei de revenire si de parametrii transmisi prin stiva)
        push    bp
        mov     bp,sp
        mov     ax,[bp+6]
        mov     bx,[bp+4]
        mov     cx,[bp+2]
; descarcarea stivei si refacerea registrului BP salvat:
        pop     bp
        add     sp,6
```

Instrucțiuni de transfer specifice acumulatorului

Aceste instrucțiuni permit transferul de date între registrul acumulator (AL, AX sau EAX) şi portul de I/O din spațiul de I/O, specificat ca operand. Specificarea portului se poate face direct în instrucțiune, pe 8 biți, iar pentru adrese mai mari se foloseşte adresarea indirectă prin DX.

IN <acc>,<port> **OUT <port>,<acc>**

Exemple:

```
in      al,port_oct             out     port_oct,al
in      ax,port_cuv             out     port_cuv,ax
in      eax,port_dcuv           out     port_dcuv,eax
in      al,dx                   out     dx,al
in      ax,dx                   out     dx,ax
in      eax,dx                  out     dx,eax
```

Toate operațiile de intrare/ieşire, indiferent de perifericul utilizat, presupun folosirea acestor instrucțiuni pentru a realiza transferul informației. În principiu, fiecărui periferic îi sunt asociate porturi pentru:

- citirea stării dispozitivului;
- transmiterea comenzii către periferic;
- realizarea transferului propriu-zis.

Se pot realiza şi transferuri de I/O pe şiruri (blocuri):

INS <dest>,DX **OUTS DX,<sursa>**

(INput from port to String ; OUTput String to port)

Aceste instrucțiuni transferă date la sau de la un port de I/O specificat de DX de la sursă sau la destinație. Operandul de memorie, dacă este sursă, este referit de DS:(E)SI, iar dacă este destinație este referit de ES:(E)DI; selecția între registrele de 16 sau 32 de biți (ESI sau SI, respectiv EDI sau DI) se execută în funcție de atributul de dimensiune de adresă.

După realizarea transferului unui element, registrele index sunt automat actualizate, pentru a face referire la următoarea adresă; dacă direcția de parcurgere DF=0, atunci vor fi incrementate, iar dacă DF=1, vor fi decrementate. Pasul de actualizare va fi 1, 2 sau 4, după cum se citeşte/scrie un octet, cuvânt sau dublu cuvânt. De fapt, în funcție de tipul transferului (octet, cuvânt sau dublu cuvânt), se vor genera instrucțiunile corespunzătoare:

INSB / INSW / INSD **OUTSB / OUTSW / OUTSD**

Aceste instrucțiuni pot fi precedate de prefixul de repetare REP, pentru a realiza transferuri pe blocuri de date. Dimensiunea blocului de date de transferat se va încărca în registrul (E)CX, care va contoriza numărul de octeți, cuvinte sau cuvinte duble transferate. Efectul acestui prefix este descris ulterior, la paragraful pentru instrucțiunile pe şiruri.

XLAT – instructiune de transfer dintr-o tabela de octeti sau
de transfer de octet, de la un cod la altul: (AL) ((BX) + (AL))

Conținutul acumulatorului este înlocuit de un octet dintr-o tabelă. Adresa de început a tabelei se află, în prealabil, în BX. Conținutul registrului AL este considerat ca adresă relativă în această tabelă de conversie (translație). Valoarea corespunzătoare din tabelă (de tip octet) este mutată în AL.

Referirea la o adresă în instrucțiunea XLAT, este necesară pentru a permite asamblorului să determine registrul segment care va fi utilizat la execuția instrucțiunii (registrul BX conține numai adresa relativă în cadrul segmentului din care face parte tabela de conversie):

XLAT [adr_tabela], XLAT [[rs:] adr_tabela] sau XLATB

Ultima formă poate fi utilizată fără operand, dacă tabela se află deja în segmentul curent referit de DS.

Instrucțiuni de transfer adrese

LEA <dest_reg>,<sursa_mem> (Load Effective Address)

Transferă în registrul destinație de 16 biți specificat de instrucțiune adresa relativă, în segment, a operandului sursă, care trebuie să fie un operand din memorie. Registrul destinație nu poate fi un registru segment. Această instrucțiune poate fi folosită pentru încărcarea registrelor BX, SI sau DI, care trebuie să conțină adrese de operanzi pentru instrucțiunea XLAT şi pentru instrucțiunile pe şiruri.

De exemplu instrucțiunea:

```
lea     bx, adr_tab
```

este echivalentă cu:

```
mov     bx, offset adr_tab
```

dar instrucțiunea:

```
lea     si, adr_tab[bx][di]
```

nu are un echivalent direct, care să utilizeze o singură instrucțiune, ci o secvență de instrucțiuni:

```
mov     si, offset adr_tab
add     si, bx
add     si, di
```

La procesoarele 386/486 destinația şi/sau sursa pot fi şi de 32 biți. Dacă destinația şi sursa au aceeaşi dimensiune (16 sau 32 biți) efectul este cel descris anterior. Pot apărea, însă, situațiile:

- **LEA <reg_16>,<mem_32>**, se vor încărca în registrul destinație de 16 biți numai ultimii 16 biți ai sursei de 32 de biți din memorie;
- **LEA <reg_32>,<mem_16>**, se va face extensia de semn a sursei de 16 biți, din memorie, la dimensiunea registrului destinație, de 32 de biți.
- **LDS <dest_reg_16>,<sursa_mem_32>** (Load pointer using DS)
- **LES <dest_reg_16>,<sursa_mem_32>** (Load pointer using ES)

Instrucțiuni de transfer indicatori

LAHF (Load AH from Flags)

(AH) ← (SF, ZF, *, AF, *, PF, *, CF)

Copiază în AH indicatorii SF, ZF, AF, PF, CF, în formatul în care aceştia se găsesc în registrul indicatori (biții 7÷0). Această instrucțiune a apărut la 8086, pentru a asigura compatibilitatea cu procesorul anterior 8080. Nu modifică nici un indicator.

SAHF (Store AH into Flags)

(SF, ZF, *, AF, *, PF, *, CF) ← (AH)

Transferă biții corespunzători din AH în registrul indicatori (biții 7÷0). Modifică doar aceşti indicatori; ceilalți nu sunt modificați.

PUSHF (PUSH Flags)

(SP) ← (SP) – 2

((SP)+1:(SP)) ← Indicatori (flags)

Transferă indicatorii în cuvântul din noul vârf al stivei. La 386/486, instrucțiunea PUSHFD pune în stivă registrul indicatori de 32 de biți (EFLAGS). Indicatorii nu sunt modificați.

POPF

Indicatori ← ((SP)+1:(SP))
(SP) ← (SP) + 2

Transferă cuvântul din vârful stivei (la care face referire SP) în registrul indicatori, modificându-le valorile anterioare. În mod asemănător, la 386 există instrucțiunea POPFD, care transferă din vârful stivei un cuvânt dublu în registrul EFLAGS.

Instrucțiuni aritmetice

Formatul datelor aritmetice

Operațiile aritmetice pot fi efectuate pe patru tipuri de numere:

- binare: fără semn;
  cu semn;
- zecimale: neîmpachetate;
  împachetate; (ambele fără semn).

Numerele binare pot fi de 8 sau 16 biți, iar la 386 şi de 32 de biți. Numerele zecimale sunt memorate în octeți:

- două cifre pe un octet, la formatul zecimal împachetat;
- o cifră pe un octet, la formatul zecimal neîmpachetat (prima tetradă conține zero, iar cea de-a doua conține o cifră de la 0 la 9).

Procesorul consideră, întotdeauna, că operanzii specificați în instrucțiunile aritmetice conțin date ce reprezintă numere valide pentru tipul instrucțiunii ce trebuie executată. Date incorecte vor conduce la rezultate neprevăzute.

Numerele binare fără semn se pot găsi în intervalele:

- [0..255], cele de tip octet,
- [0..65535], cele de tip cuvânt, şi
- [0..2^{32}-1], cele de tip cuvânt dublu.

Pentru aceste date sunt definite instrucțiunile de adunare, scădere, înmulțire şi împărțire.

Numerele binare cu semn sunt reprezentate în cod complementar (complement față de 2) şi pot avea valorile:

- [-128..+127], cele de tip octet,
- [-32268..+32267], cele de tip cuvânt, şi
- [-2^{31}…+2^{31}-1], cele de tip dublu cuvânt.

Sunt definite operațiile de înmulțire şi împărțire, specifice acestui tip de date, iar operațiile de adunare şi scădere sunt realizate cu instrucțiunile pentru numere binare fără semn. Dacă pentru numere fără semn depăşirea este detectată de indicatorul CF, pentru numere cu semn aceasta este detectată de indicatorul OF. Pentru determinarea depăşirii, în astfel de cazuri se pot utiliza instrucțiunile de salt condiționat, pentru CF sau OF, după operația respectivă.

Numerele în format zecimal neîmpachetat pot reprezenta valori în intervalul 0÷9, pe un octet fără semn. Operațiile aritmetice cu numere în format zecimal neîmpachetat se realizează în două etape. Operațiile de adunare, scădere şi înmulțire, de la numere binare fără semn, furnizează un rezultat intermediar în AL, care apoi este ajustat la o valoare corectă, de număr în format zecimal neîmpachetat. Împărțirea se face în mod asemănător, cu excepția ajustării care este realizată prima, asupra operandului numărător, în registrul AL, şi apoi se realizează împărțirea binară fără semn; dacă rezultatul nu este corect, mai poate urma o a treia etapă de ajustare finală.

Reprezentarea în format zecimal neîmpachetat este asemănătoare cu aceea a caracterelor numerice 0÷9 în codul ASCII, cu diferența că tetrada mai semnificativă din codul ASCII are valoarea 3H, în loc de 0H. Din acest motiv, formatul zecimal neîmpachetat mai este denumit format ASCII. Se pot realiza operații direct pe caracterele numerice ASCII, astfel:

- se pune pe 0 prima tetradă a numărului ASCII;
- se efectuează operația respectivă şi corecția necesară, care lasă această tetradă pe 0H;
- se pune prima tetradă a numărului pe 3H şi se obține direct codul ASCII al cifrei respective.

Numerele în format zecimal împachetat sunt memorate fără semn într-un octet care conține câte o cifră zecimală în fiecare jumătate de octet (tetradă sau patru biți). Prima cifră este cea mai semnificativă, şi întrucât fiecare tetradă poate conține o cifră de la 0 la 9, rezultă că valoarea unui astfel de număr este cuprinsă în intervalul 0÷99. Operațiile de adunare şi scădere se realizează în două etape, la fel ca la numerele în format zecimal neîmpachetat:

- se utilizează instrucțiunea binară fără semn respectivă (+, -), care va furniza un rezultat intermediar în AL;
- se realizează corecția rezultatului din AL la o valoare corectă în format zecimal împachetat.

Nu există instrucțiuni de ajustare pentru operațiile de înmulțire sau împărțire cu astfel de numere. După o astfel de instrucțiune, indicatorii pot fi:

- modificați, conform rezultatului;
- nemodificați, deci vor rămâne la valoarea avută anterior acestei instrucțiuni;
- nedefiniți, adică sunt modificați de instrucțiunea respectivă, dar nu reflectă starea rezultatului.

Instrucțiunile de adunare/scădere

ADD <dest>,<sursa>	(ADDition) (dest) ← (dest) + (sursa) Modifica: OF, SF, ZF, AF, PF, CF
ADC <dest>,<sursa>	(ADdition with Carry flag) (dest) ← (dest) + (sursa) + (CF) Modifica: OF, SF, ZF, AF, PF, CF
INC <dest>	(INCrement) (dest) ← (dest) + 1 Modifica: OF, SF, ZF, AF, PF Nu afecteaza: CF ;
AAA	(ASCII Adjust for Addition)

înlocuieşte conținutul registrul AL, obținut prin adunarea a două numere în format zecimal

neîmpachetat, cu un număr în format zecimal neîmpachetat corect, astfel:

```
if ((AL) & 0FH) > 9 or (AF)=1 then
        (AL) ← (AL) + 6;
        (AH) ← (AH) + 1;
        (AF) ← 1
        (CF) ← 1
        (AL) ← (AL) & 0FH
else
        (CF) ← 0
        (AF) ← 0
        Modifica: AF, CF;
        Nedefiniti: OF, SF, ZF, PF.
```

Simbolul &, utilizat în descrierea instrucțiunilor, reprezintă operația ŞI logic la nivel de bit.

Corecția aceasta este necesară deoarece procesorul efectuează calculele în binar, iar un transport de la bitul 3 la bitul 4 înseamnă de fapt modificarea valorii de la 8 la 16, valoare interpretată de utilizator ca fiind 10, deoarece numărul este citit în zecimal; din acest motiv, trebuie să se adune valoarea 6, pentru a obține valoarea zecimală corectă.

DAA (Decimal Adjust for Addition)

înlocuieşte conținutul registrului AL, obținut prin adunarea a două numere în format zecimal împachetat, cu un număr zecimal împachetat, astfel:

```
if ((AL) & 0FH) > 9 or (AF)=1 then
        (AL) ← (AL) + 6;
        (AF) ← 1
else
        (AF) ← 0;
if ((AL) > 9FH) or (CF)=1 then
        (AL) ← (AL) + 60H;
        (CF) ← 1
else
        (CF) ← 0;
        Modifica: SF, ZF, AF, PF, CF
        Nedefinit: OF.
```

Se realizează o corecție asemănătoare ca la instrucțiunea precedentă, cu deosebirea că această corecție se face pentru ambele tetrade, întrucât numărul este zecimal împachetat.

Începând de la procesorul 486 şi următoarele, se mai poate executa şi instrucțiunea:

XADD <dest>,<sursa> (eXchange and ADD)

care schimbă sursa cu destinația, asemănător instrucțiunii XCHG, dar realizează şi suma celor doi operanzi, ca la instrucțiunea ADD; deci, practic, combină cele două instrucțiuni într-

una singură astfel: adună sursa cu destinația şi depune suma la destinație, dar copiază şi valoarea inițială a destinației, pe care o depune la sursă. Operandul destinație poate fi reg/mem, iar cel sursă reg.

SUB <dest>,<sursa> (SUBtraction)
(dest) ← (dest) – (sursa)
Modifica: OF, SF, ZF, AF, PF, CF
SBB <dest>,<sursa> (SuBtraction with Borrow)
(dest) ← (dest) – (sursa) – (CF)
Modifica: OF, SF, ZF, AF, PF, CF
DEC <dest> (DECrement)
(dest) ← (dest) – 1
Modifica: OF, SF, ZF, AF, PF
Nu afecteaza: CF ;
NEG <dest>

determină complementul față de doi al operandului destinație şi-l returnează destinației:

(dest) ← compl'2 (dest)
{ (dest) ← 0 – (dest) }
Modifica: OF, SF, ZF, AF, PF, CF

Dacă operandul este zero, el rămâne neschimbat şi CF=0. Pentru toate celelalte numere CF=1. Dacă se complementează cel mai mic număr negativ (de ex. -128 sau -32268), operandul nu se va modifica, dar OF=1.

CMP <dest>,<sursa>

poziționează toți indicatorii pentru operația:

(dest) – (sursa)

fără însă a modifica operandul destinație şi nici pe cel sursă. De obicei, după o astfel de instrucțiune urmează o instrucțiune de salt condiționat.

CMPXCHG <dest>,<sursa> (CoMPare and eXCHanGe)

Operandul destinație poate fi reg/mem, iar cel sursă reg. Instrucțiunea este disponibilă începând de la procesorul 486. Spre deosebire de instrucțiunea *cmp*, instrucțiunea *cmpxchg* modifică doar indicatorul ZF. Instrucțiunea utilizează, de asemenea, registrul acumulator, alegând în mod automat, în funcție de dimensiunea operanzilor, unul din registrele *al*, *ax* sau *eax*. Instrucțiunea compară acumulatorul cu primul operand şi actualizează ZF. Dacă acumulatorul este egal cu primul operand, ZF=1 şi copiază cel de-al doilea operand în primul. În caz contrar, ZF=0 şi se copiază primul operand în acumulator.

Procesoarele Pentium permit compararea şi interschimbarea pe 64 de biți cu ajutorul instrucțiunii *cmpxchg8b*, care are sintaxa:

cmpbxchg8b ax, mem_64

Instrucțiunea compară valoarea din memorie de 64 de biți la care face referire *mem_64* cu valoarea de 64 de biți din *edx:eax*. În cazul în care cele două valori sunt egale, se memorează valoarea din *edx:eax* în memorie la *mem_64* şi se setează ZF=1, altfel se încarcă perechea de registre (*edx:eax*) cu valoarea din memorie (*mem_64*) şi ZF=0.

AAS (ASCII Adjust for Subtraction)

înlocuieşte conținutul registrului AL, obținut prin scăderea a două numere în format zecimal neîmpachetat, cu un număr corect în format zecimal neîmpachetat, astfel:

```
if ((AL) & 0FH) > 9 or (AF)=1 then
        (AL) ← (AL) – 6;
        (AH) ← (AH) – 1;
        (AF) ← 1
        (CF) ← 1
        (AL) ←  (AL) & 0FH
else
        (CF) ← 0
        (AF) ← 0
Modifica: AF, CF;
Nedefiniti: OF, SF, ZF, PF.
```

Corecția aceasta este necesară deoarece procesorul efectuează calculele în binar şi un împrumut de la bitul 4 la bitul 3 înseamnă de fapt un împrumut de valoare 16, dar această valoare este considerată de utilizator ca fiind 10, deoarece numărul este interpretat în zecimal; din acest motiv trebuie să se scadă valoarea 6, pentru a corespunde valorii zecimale reale.

DAS (Decimal Adjust for Subtraction)

înlocuieşte conținutul registrului AL, obținut prin scăderea a două numere în format zecimal împachetat, cu un număr în format zecimal împachetat, astfel:

```
if ((AL) & 0FH) > 9 or (AF)=1 then
        (AL) ← (AL) – 6;
        (AF) ← 1
else
        (AF) ← 0;
if ((AL) > 9FH) or (CF)=1 then
        (AL) ← (AL) – 60H;
        (CF) ← 1
else
        (CF) ← 0;
Modifica: SF, ZF, AF, PF, CF
```

Nedefinit: OF.

Exemple de utilizare a acestor instrucțiuni:

1) Programul adună două numere fără semn, reprezentate pe mai multe cuvinte (octeți), iar rezultatul se va depune peste primul număr.

```
add_data                segment
        numar_1     dw          0a8adh, 7fe2h, 0a876h,0
        lung_nr1    equ         ($ – numar_1)/2
        numar_2     dw          0cdefh, 52deh, 378ah, 0
add_data                ends
multibyte_add           segment
                        assume      cs: multibyte_add, ds: add_data
        start:  mov     ax,add_data ; initializarea reg. DS
                mov     ds,ax       ; cu adresa de segment
                mov     cx,lung_nr1 ; contor lungime numar
                mov     si,0        ; initializare index elemente
                clc                 ; (CF)=0, transportul initial
        bucla:  mov     ax,numar_2[si]
                adc     numar_1[si],ax
                inc     si          ; actualizare index element
                inc     si
                loop    bucla
                mov     ax,4c00h    ; revenire in DOS
                int     21h
multibyte_add ends
end     start
```

2) Complementarea unui număr reprezentat pe mai multe cuvinte.

Se pornește chiar de la definiția complementului față de 2: complement față de 2 pentru număr = 2^n – număr; unde numar este reprezentat pe *n-1* biți.

```
        mov     cx,lung_numar       ; contor numar cuvinte
        clc                         ; initializare transport cu 0
        mov     si, -2          ; initializare index
compl:  inc     si              ; actualizare index cuvant prim/ urmator
        inc     si
        mov     ax,0            ; echivalentul lui 2n, se face scaderea
        sbb     ax,numar [si]   ;       cu propagarea imprumutului si
                                ; pozitionare OF/ depasire, la ultimul
        mov     numar[si],ax    ; cuvant, se depune rezultatul, si se
        loop    compl           ; pastreaza ultimul OF
                                ; testare depasire (OF), ce are loc doar pentru
                                ; valoarea minima negativa (80.00... 00h)
```

Dacă numărul inițial are valoarea minimă negativă (80 00 ...00H), după complementare valoarea va fi aceeaşi, deci rămâne nemodificată, ca şi în cazul instrucțiunii de complementare (NEG).

Instrucțiunile de înmulțire/împărțire

MUL <sursa> ; **MUL <acumulator>,<sursa>** – la 386/486

Realizează înmulțirea fără semn a operandului sursă cu acumulatorul (AL, AX sau EAX). Dacă sursa este de tip octet atunci este înmulțită cu registrul AL şi rezultatul de lungime dublă este returnat în AX. Dacă operandul sursă este de tip cuvânt, atunci este înmulțit cu registrul AX, iar rezultatul este returnat în perechea de registre (DX, AX). Dacă sursa este de 32 biți, atunci: (sursa_32) × (EAX) (EDX, EAX). Operanzii sunt consideraţi fără semn. Dacă jumătatea superioară a rezultatului (AH, DX sau EDX – în funcție de tipul operanzilor) este diferită de zero, ceea ce înseamnă depăşirea formatului inițial al numerelor, atunci OF=CF=1; în caz contrar, OF=CF=0. Deci, dacă după înmulțirea numerelor rezultă OF=CF=1, atunci jumătatea superioară a rezultatului (AH, DX sau EDX) conține cifre semnificative ale rezultatului (depăşeşte dimensiunea inițială a operanzilor). Ceilalți indicatori sunt nedefiniți (SF, ZF, AF, PF).

Exemplu de înmulțire a unui octet cu un cuvânt:

```
mov   al,op1_octet      ; inmultitorul de tip octet
mov   ah,0              ; se extinde inmultitorul la cuvant, fara semn
mul   op2_cuv           ; se realizeaza inmultirea
```

IMUL <sursa> ; **IMUL <acumulator>,<sursa>** – la 386/486

Realizează înmulțirea cu semn a operandului sursă cu acumulatorul (AL, AX sau EAX) şi depune rezultatul de lungime dublă în AX (DX, AX) sau (EDX, EAX), în funcție de tipul sursei. Operația se realizează în mod asemănător ca la înmulțirea fără semn. Dacă jumătatea superioară a rezultatului (AH, DX, EDX) nu reprezintă extensia de semn a jumătății inferioare (AL, AX, EAX), atunci OF=CF=1; în caz contrar, OF=CF=0. Deci, dacă după înmulțirea numerelor se obține OF=CF=1, atunci jumătatea superioară a rezultatului conține cifre semnificative ale rezultatului (depăşeşte dimensiunea inițială a operanzilor). La 386/486, pe lângă aceste instrucțiuni se pot realiza înmulțiri care pot avea ca destinație şi alte registre, nu numai cele prestabilite (AX, DX:AX sau EDX:EAX). Acestea pot avea doi sau trei operanzi:

IMUL r_16, r/m_16
IMUL r_32, r/m_32
IMUL r_16, data_8
IMUL r_32, data_8
IMUL r_16, data_16
IMUL r_32, data_32
IMUL r_16, r/m_16, data_8
IMUL r_32, r/m_32, data_8
IMUL r_16, r/m_16, data_16
IMUL r_32, r/m_32, data_32

Instrucțiunile cu doi operanzi au ca operand destinație un registru general de 16/32 biți, iar operandul sursă poate fi un registru, un operand din memorie, o dată imediată de 8 biți sau o dată imediată, de aceeaşi dimensiune cu registrul destinație.

Instrucțiunile cu trei operanzi au ca operand destinație un registru general de 16/32 biți şi doi operanzi sursă: unul este un registru sau operand memorie, de aceeaşi dimensiune cu destinația, iar cel de-al doilea poate fi o dată imediată de 8 biți sau de aceeaşi dimensiune cu ceilalți doi operanzi.

Pentru aceste instrucțiuni, care au ca destinație un alt registru general decât acumulatorul, rezultatul este trunchiat la dimensiunea registrului destinație. Dacă rezultatul nu depăşeşte dimensiunea destinației, deci nu este necesară trunchierea lui la dimensiunea destinației, atunci OF=CF=0; în caz contrar, OF=CF=1. Deci, practic, indiferent de tipul înmulțirii şi de numărul de operanzi, dacă apare o depăşire a dimensiunii inițiale a destinației cei doi indicatori vor fi poziționați pe 1.

Exemplu de înmulțire cu semn a unui octet cu un cuvânt:

```
mov   al,op1_octet          ; inmultitorul de tip octet
cbw                                                    ;  se
extinde semnul inmultitorului la cuvant
imul op2_cuv                ; se realizeaza inmultirea
```

AAM (Ascii Adjust for Multiply)

(AH) ← (AL)/0AH
(AL) ← (AL) mod 0AH
Modifica: SF, ZF, PF;
Nedefiniti: OF, AF, CF

Corectează rezultatul unei înmulțiri anterioare a doi operanzi zecimali neîmpachetați. În urma înmulțirii, utilizând instrucțiunea MUL, a doi operanzi de tip zecimal împachetat, se obține produsul în (AH,AL), unde (AH) = 0, iar (AL) <> 0. Pentru ca rezultatul returnat de această instrucțiune să fie corect trebuie ca (AL) < 100, condiție care este îndeplinită dacă valoarea sa se obține prin înmulțirea a două numere de tip zecimal neîmpachetat (valoarea maximă este 9, deci 9×9 =81< 100).

DIV **< sursa >** sau **DIV < acc >,< sursa >** la 386.

```
temp) ← (numarator)
if          (temp) / (numitor) > MAX then
            (cat), (rest) – nedefiniti
            (SP) ← (SP) – 2;              intrerupere
            ((SP)+1:(SP)) ← indicatori; pe
            (tf), (if) ← 0;               nivelul
            (SP) ← (SP) – 2;              0
            ((SP)+1:(SP)) ← (CS);
            (CS) ← (2)
            (SP) ← (SP) – 2;
            ((SP)+1:(SP)) ← (IP);
            (IP) ← (0)
else
            (cat) ← (temp) / (numitor)
            (rest) ← (temp) mod (numitor)
```

MAX = FFH/octet, FFFFH/cuvant, FFFFFFFFH/dublu cuvant.

Indicatori: toți sunt nedefiniți.

Această instrucțiune realizează împărțirea întreagă, fără semn, a acumulatorului (AL, AX, EAX) şi a extensiei sale (AH, DX, EDX) prin operandul sursă. Operandul deîmpărțit este de lungime dublă (AX, DX:AX, EDX:EAX) față de operandul împărțitor (octet, cuvânt sau dublu cuvânt). În cazul în care câtul depăşeşte capacitatea destinației (>MAX) sau la împărțirea prin zero se generează întrerupere pe nivelul 0.

În urma împărțirii întregi, restul şi câtul se obțin în funcție de tipul operanzilor, astfel:

rest	cât	tip împărțire
AH	AL	pentru împărțire la octet
DX	AX	pentru împărțire la cuvânt
EDX	EAX	pentru împărțire la dublu cuvânt

IDIV < sursa > sau **IDIV < acc >,< sursa >** la 386/486.

Această instrucțiune realizează împărțirea întreagă, cu semn, a acumulatorului (AL, AX, EAX) şi a extensiei sale (AH, DX, EDX) prin operandul sursă. Operandul deîmpărțit este de lungime dublă (AX, DX:AX, EDX:EAX) față de operandul împărțitor (octet, cuvânt sau dublu cuvânt). În cazul în care câtul depăşeşte capacitatea destinației (este mai mare decât MAX sau mai mic decât -MAX-1) sau la împărțirea prin zero se generează întrerupere pe nivelul 0, întocmai ca la împărțirea fără semn. Constanta MAX are valorile 7FH, 7FFFH şi 7FFFFFFFH pentru câturi pozitive de tip octet, cuvânt şi dublu cuvânt, respectiv 80H, 8000H, 80000000H pentru câturi negative de tip octet, cuvânt şi dublu cuvânt. La fel ca la împărțirea fără semn, la împărțirea întreagă toți indicatorii sunt nedefiniți, iar restul şi câtul împărțirii se obțin în aceleaşi perechi de registre (AH,AL ; DX,AX ; EDX,EAX). În rest, descrierea instrucțiunii IDIV este aceeaşi ca la împărțirea fără semn (DIV), doar cu modificarea corespunzătoare a liniei:

if (temp) / (numitor) > MAX or (temp) / (numitor) > -MAX –1 then
.....

AAD (ASCII Adjust for Division)

(AL) ← (AH) * 0AH + (AL)
(AH) ← 0

Indicatori: modifică SF, ZF, PF, iar ceilalți sunt nedefiniți: OF, CF, AF.

Modifică numărătorul din AL înaintea împărțirii a doi operanzi corecți de tip zecimal neîmpachetat, astfel încât deîmpărțitul să reprezinte corect valoarea respectivă în binar, deoarece instrucțiunea realizează împărțirea în binar. AH trebuie să fie zero pentru următoarea instrucțiune DIV, în vederea furnizării unui rezultat corect. Câtul, în urma operației de împărțire, este returnat în AL, iar restul în AH. Pentru a transforma rezultatul într-un număr corect de tip zecimal neîmpachetat (ASCII), în cazul în care câtul este mai mare de 9, trebuie utilizată, după împărțire, instrucțiunea de corecție AAM, bineînțeles după ce s-a salvat restul din AH.

Instrucțiuni de prelucrare la nivel de bit

În setul de instrucțiuni există trei grupuri de instrucțiuni pentru manipularea biților, pentru date de tip octet, cuvânt sau dublu cuvânt, şi anume:

- instrucțiuni logice (NOT, AND, OR, XOR, TEST etc.);
- instrucțiuni de deplasare (SHL, SAL, SHR, SAR etc.);
- instrucțiuni de rotire (ROL, ROR, RCL, RCR);

Toate aceste operații se efectuează bit cu bit.

Instrucțiuni logice

Aceste instrucțiuni modifică indicatorii astfel:

OF şi CF ← 0;
AF – nedefinit;
SF, ZF, PF – sunt poziționați conform rezultatului instrucțiunii respective şi pot fi testați de instrucțiunile de salt condiționat.

Excepție: instrucțiunea NOT nu modifică nici un indicator.

NOT <dest>

Inversează toți biții operandului sursă (complement față de 1). Nu afectează nici un indicator.

AND <dest>, <sursa>
(dest) ← (dest) and (sursa)
OR <dest>, <sursa>
(dest) ← (dest) or (sursa)
XOR <dest>, <sursa>
(dest) ← (dest) xor (sursa)
TEST <dest>, <sursa>

Poziționează indicatorii pentru operația (dest) and (sursa), fără a modifica nici un operand.

Dacă instrucțiunea TEST este urmată de o instrucțiune de salt JNZ, saltul va avea loc dacă există cel puțin doi biți egali cu 1 pe aceleaşi poziții în cei doi operanzi ai instrucțiunii.

Operanzii pot fi întocmai ca la instrucțiunile de transfer, adică reg/mem pentru destinație şi reg/mem sau dată pentru sursă.

Tot în această categorie pot fi incluse şi instrucțiunile, specifice procesoarelor 386/486, de testare / modificare de bit, cele de scanare pe bit, precum şi instrucțiunile de setare condiționată.

Instrucțiuni de testare şi modificare a unui bit

Aceste instrucțiuni operează pe un singur bit, specificat în instrucțiune prin deplasamentul său (în cel de-al doilea operand, care poate fi o dată de 8 biți sau un registru de 16/32 biți) față de bitul cel mai puțin semnificativ al operandului destinație (registru sau locație de memorie de 16/32 biți). Bitul respectiv este transferat în CF şi apoi modificat conform instrucțiunii respective (ceilalți indicatori rămân nemodificați).

BT **<dest>, <pozitie>** (Bit Test)
(CF) ← Bit (dest, pozitie)

BTS **<dest>, <pozitie>** (Bit Test and Set)
(CF) ← Bit (dest, pozitie),
Bit (dest, pozitie) ← 1

BTR **<dest>, <pozitie>** (Bit Test and Reset)
(CF) ← Bit (dest, pozitie),
Bit (dest, pozitie) ← 0

BTC **<dest>, <pozitie>** (Bit Test and Complement)
(CF) ← Bit (dest, pozitie),
Bit (dest, pozitie) ← Not (Bit (dest, pozitie))

Instrucțiuni de scanare pe bit

Aceste instrucțiuni permit scanarea directă a biților din cel de-al doilea operand (de tip cuvânt sau dublu cuvânt), începând cu bitul mai puțin semnificativ (forward) sau invers, începând cu bitul cel mai semnificativ (reverse), pentru a determina primul bit egal cu 1. Dacă toți biții sunt zero, atunci ZF=1; în caz contrar ZF=0, şi registrul destinație va reține indexul primului bit 1 din operandul sursă (registru sau memorie de aceeaşi dimensiune cu registrul destinație). Se modifică numai indicatorul ZF.

- scanare directă:

BSF **<dest>, <sursa>** (Bit Scan Forward)

- scanare inversă:

BSR **<dest>, <sursa>** (Bit Scan Reverse)

Instrucțiuni de setare condiționată

Instrucțiunile din această categorie permit poziționarea unui octet pe zero sau unu, în funcție de una din cele 16 condiții definite de indicatori. Operandul destinație, de tip octet, poate fi un registru sau o locație de memorie. Aceste instrucțiuni sunt folosite pentru implementarea expresiilor booleene din limbajele de nivel înalt. Forma generală a instrucțiunii este:

SETcond **<dest>** (SET byte on condition)
if cond then
(dest) ← 1
else
(dest) ← 0

Aceste instrucțiuni nu modifică nici un indicator.

Cele 16 instrucțiuni referitoare la condițiile respective sunt următoarele:

SETE / SETZ	Condiția ZF = 1
SETNE / SETNZ	Condiția ZF = 0
SETL / SETNGE	Condiția SF <> OF, valori cu semn
SETLE / SETNG	Cond. SF <> OF sau ZF = 1, valori cu semn
SETNL / SETGE	Condiția SF = OF, valori cu semn
SETNLE / SETG	Cond. SF = OF şi ZF = 0, valori cu semn
SETB / SETNAE / SETC	Cond. CF = 1, valori fără semn
SETBE / SETNA	Cond. CF = 1 sau ZF = 1,fără semn
SETNB / SETAE / SETNC	Cond. CF = 0, valori fără semn
SETNBE / SETA	Cond. CF = 0 şi ZF = 0,fără semn
SETO / SETNO	Cond. OF = 1 / respectiv OF = 0
SETP / SETPE	Cond. PF = 1, adică paritate pară
SETNP / SETPO	Cond. PF = 0, adică paritate impară
SETS / SETNS	Cond. SF = 1 / respectiv SF = 0

Instrucțiuni de deplasare

Deplasările pot fi aritmetice sau logice. Se poate deplasa operandul destinație cu până la 31 de biți, corespunzător operandului contor codificat în instrucțiune (sunt luați în considerare numai ultimii 5 biți ai acestuia). Contorul poate fi specificat ca o constantă imediată în instrucțiune, dacă are valoarea 1, sau ca registrul CL, dacă este diferit de 1; în acest fel, contorul de deplasări poate fi o variabilă furnizată în momentul execuției. Începând de la procesoarele 286 (386/486/Pentium) contorul, chiar dacă este diferit de 1, poate fi specificat în instrucțiune ca dată imediată.

Deplasările aritmetice pot fi utilizate pentru a înmulți sau împărți numere binare prin puteri ale lui 2. Deplasările logice pot fi utilizate pentru a izola biți în octeți sau cuvinte.

Indicatorii sunt modificați astfel:

OF = este nedefinit într-o deplasare pe mai mulți biți;
= este poziționat corespunzător, dacă se efectuează o deplasare de un bit
(1 – dacă se modifică bitul de semn în urma deplasării, altfel este 0);
CF = ultimul bit deplasat în afara operandului destinație;
AF = nedefinit;
SF, ZF, PF = modificați conform rezultatului.

SHL / SAL <dest>, <contor> (SHift logical Left / Shift Arithmetic Left)

Pozițiile eliberate din dreapta operandului se completează cu zerouri, indiferent de tipul deplasării; deci, între cele două mnemonici nu există nici o deosebire. Bitul cel mai semnificativ se deplasează în CF. În cazul în care contorul de deplasări este 1 şi dacă după deplasarea aritmetică (SAL), (CF) <> primul bit al rezultatului, deci s-a schimbat semnul rezultatului, atunci (OF) = 1 (depăşire domeniu de reprezentare); în caz contrar, (OF) = 0.

Exemple:

```
shl     ah, 1           ; deplasari de 1 bit, care va ajunge
sal     beta[bx][di], 1 ; in CF
mov     cl, 5           ; deplasari de mai multi biti, de ex. 5
sal     dx, cl          ; ultimul bit deplasat este in CF
shl     beta[bp][si], cl
```

SHR **<dest>, <contor>** (SHift logical Right)

Fiind o deplasare logică, pozițiile eliberate prin deplasare din stânga se vor completa cu zerouri. Bitul cel mai puțin semnificativ se va deplasa în CF. La o deplasare de 1 bit, dacă se modifică bitul de semn atunci (OF)=1; în caz contrar (OF)=0.

SAR **<dest>, <contor>** (Shift Arithmetic Right)

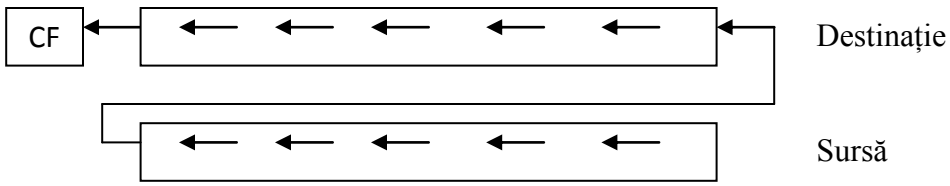
Bitul cel mai semnificativ îşi păstrează vechea valoare, dar este şi deplasat spre dreapta (extensie de semn), ceea ce înseamnă că nu poate să apară depăşire (schimbarea de semn a rezultatului) şi (OF)=0 la o deplasare de 1 bit. Bitul cel mai puțin semnificativ se va deplasa în CF. Trebuie menționat că SAR nu furnizează acelaşi rezultat ca instrucțiunea IDIV pentru aceeaşi operanzi, dacă deîmpărțitul este negativ, şi biți egali cu 1 sunt deplasați (SAR) în afara operandului. De exemplu, -5 deplasat la dreapta (SAR) cu un bit va furniza rezultatul -3, în timp ce împărțirea întreagă (IDIV) va furniza valoarea -2:

(-5) = (11111011) (11111101) = (-3)

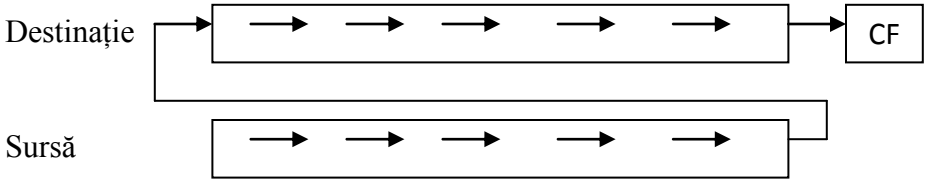
Diferența dintre cele două instrucțiuni este că IDIV trunchiază toate câturile către zero, în timp ce SAR trunchiază numerele pozitive către zero, iar pe cele negative către infinit negativ.

La 386/486 pot fi realizate instrucțiuni de deplasare dublă, pe cuvânt sau dublu cuvânt, furnizând din doi operanzi de intrare (cuvânt sau dublu cuvânt) un rezultat de ieşire de aceeaşi lungime (cuvânt sau dublu cuvânt). Operandul destinație poate fi registru sau memorie, iar sursa poate fi un registru general. Rezultatul înlocuieşte operandul destinație, iar contorul numărului de biți deplasați este specificat de registrul CL sau ca o valoare imediată de 8 biți; el poate avea valoarea maximă 31 (deci este considerat modulo 32). Indicatorii sunt poziționați la fel ca la deplasările multibit. Aceste instrucțiuni furnizează operațiile de bază necesare pentru implementarea operațiilor de deplasare pe şiruri lungi de biți (64 sau mai mult) nealiniați.

SHLD **<dest>, <sursa>, <contor>** (SHift Left Double)



SHRD **<dest>, <sursa>, <contor>** (SHift Right Double)



De notat că registrul sursă nu se modifică, deci el rămâne la valoarea avută anterior deplasării; pozițiile eliberate, pe durata deplasării, prin deplasarea biților se vor pune pe zero. Instrucțiunea este utilă atunci când se împachetează date din mai multe surse diferite.

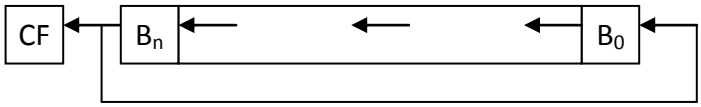
Instrucțiuni de rotire

În cazul rotațiilor, biții deplasați în afara operandului nu sunt pierduți, ca în cazul deplasărilor, ci sunt rotiți înapoi către celălalt capăt al operandului. Numărul de rotiri este dat de un contor care poate fi constanta 1 sau registrul CL, dacă numărul de rotiri este diferit de 1. Ca şi la deplasări contorul din CL este interpretat modulo 32, adică sunt luați în considerare ultimii 5 biți ai acestuia. Începând de la procesoarele 286 (deci şi la următoarele: contorul, chiar dacă este diferit de 1, poate fi specificat în instrucțiune ca dată imediată. Indicatorul CF poate acționa ca o extensie a operandului, la două dintre instrucțiunile de rotire, permițând unui bit să fie izolat în CF şi apoi testat de instrucțiuni JC sau JNC.

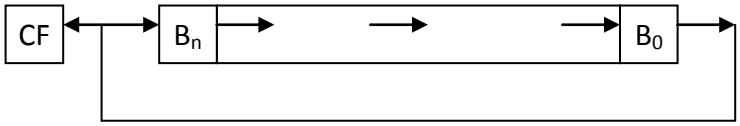
Rotațiile modifică numai CF şi OF:

CF = ultimul bit rotit în afara operandului;
OF = nedefinit pentru rotații multibit;
= 1 sau 0, pentru rotație doar de 1 bit, după cum s-a modificat sau nu bitul de semn al operandului.

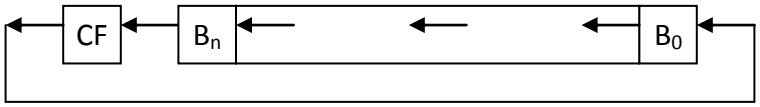
ROL **<dest>, <contor>** (ROtate Left)



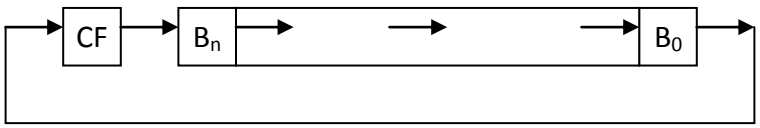
ROR **<dest>, <contor>** (ROtate Right)



RCL **<dest>, <contor>** (Rotate through Carry Left)



RCR **<dest>, <contor>** (ROtate through Carry Right)



Exemple utilizând aceste instrucțiuni:

5) Tipărirea conținutului registrului DX, în format octal:

```
                tip_car         proc    far
        ; procedura afiseaza caracterul al carui cod ASCII
        ; il primeste in registrul AL
                push    dx      ; se salveaza registrul de lucru DX
                mov     dl, al  ; se apeleaza functia 2 din DOS care
                mov     ah, 2   ; afiseaza caracterul al carui cod
                int     21h     ; ASCII se transmite in registrul DL
                pop     dx      ; se reface registrul salvat
                ret
        tip_car                 endp
        tip_octal       proc    far
        ; tipareste in octal valoarea fara semn transmisa in DX
                push    cx      ; se salveaza registrele de lucru
                push            ax
        ; prima cifra de afisat este doar de 1 bit
                rol     dx, 1   ; care este rotit pe ultimul bit
                mov     al, dl  ; si dusa in registrul AL
                and     al, 1   ; se sterg ceilalti biti
                add     al, 30h ; este convertita la codul ASCII
                call    tip_car ; si se afiseaza
        ; urmatoarele 5 cifre sunt de cate 3 biti
                mov     cx, 5   ; contor numar de cifre de afisat
        octal1:
                push    cx      ; se salveaza contorul in stiva
                mov     cl, 3   ; contor numar de rotiri la stanga
                rol     dx, cl  ; cifra este rotita pe ultimii 3 biti
                mov     al, dl  ; si adusa in AL
                and     al, 7   ; sunt stersi ceilalti biti
                add     al, 30h ; si convertita la codul ASCII
                call    tip_car ; afisarea cifrei
                pop     cx      ; se citeste valoarea contorului de cifre
                loop    octal1
                pop     ax                                              ;       se
refac registrele salvate in stiva
                pop     cx
                ret
        tip_octal       endp
```

6) Împachetarea unor date din mai multe surse diferite utilizând instrucțiunea de deplasare pe dublu cuvânt *shld*. De exemplu, să asamblăm într-un singur cuvânt 4 tetrade diferite, aflate

în memorie la locații succesive (începând de la adresa *tetrada*) aliniate la stânga în cuvintele respective şi memorate începând cu tetrada cea mai semnificativă.

```
        mov     si, 0           ; index tetrada
        mov     cx, 4           ; contor numar de tetrade, de asamblat in cuvant
reia_depl:
        mov     ax[si]          ; tetrada ‘si’
        shld    bx, ax, 4       ; deplasare tetrada in BX
        inc     si              ; actualizare index
        loop    reia_depl
                                ; in final in BX se va afla cuvantul asamblat
                                ; (cele 4 tetrade)
```