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

Programare în limbaj de asamblare

24. Instrucțiuni de transfer, aritmetice, de prelucrare la nivel de bit.

Instrucțiuni de transfer date.

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

Instrucțiuni de transfer cu scop general

MOV <dest>,<sursa>

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;

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>

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

- CS nu poate fi destinație.

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

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

- instrucțiunea nu se poate executa cu registrele segment.

Instrucțiuni de conversie (extensie de semn)

realizează extensia de semn a acumulatorului (AL, AX, EAX) în extensia sa (AH, DX sau EAX, EDX). Nu se modifică nici un indicator.

CBW (Convert Byte to Word)
CWDE (Convert Word to Double word Extended)
CWD (Convert Word to Double word)
CDQ (Convert Double word to Quad word)

La 386/486 transfer cu extensia bitului de semn sau de zerouri:

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

Destinația poate fi un reg 16/32 biți, iar sursa reg/mem 8/16 biți.

MOVSX reg_16,reg/mem_8
MOVSX reg_32,reg/mem_8
MOVSX reg_32,reg/mem_16

Nu este afectat nici un indicator.

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

Instrucțiuni de interschimbare a datelor

XCHG <dest>,<sursa> (eXCHanGe)

(temp) ← (destinație), (destinație) ← (sursă), (sursă) ← (temp)

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

Dacă unul dintre operanzi este din memorie, atunci procesorul activează, în mod automat semnalul LOCK\, pentru a asigura accesul exclusiv la memorie pentru un singur procesor, într-o structură multiprocesor.

Exemplu: să inversăm, între ei, octeții dintr-un şir de cuvinte.

```
        .model small
        .data
         mesajdw     'Ex','em','pl','u ','pr','og','ra','m ','2$'
         lung_mesaj  equ     ($-mesaj)/2
         linie_noua  db      0dh,0ah,'$'
.code
        start:  mov     ax,@data        ; inițializare registru segment DS
                mov     ds,ax           ; cu adresa segmentului de date
                lea     dx,mesaj        ; tipărim mesajul sub forma inițială
                mov     ah,9            ; utilizând funcția 9, din DOS :
                int     21h             ; "xEmelp urpgora m"
                lea     dx,linie_noua   ; se trece pe o linie noua
                mov     ah,9
                int     21h
                ; pentru a aduce mesajul la forma dorită trebuie inversați
                ; octeții, fiecărui cuvânt între ei
                lea     si,mesaj
                mov     cx,lung_mesaj
iar:    mov     ax,[si]         ; se ia câte un cuvânt
                xchg    al,ah           ; şi se inversează octeții
                mov     [si],ax         ; se depune la aceeaşi adresă
                add     si,type mesaj   ; se actualizează adresa curentă
                loop    iar             ; şi se continuă pentru tot şirul
                                        ; se tipăreşte mesajul astfel obținut
                lea     dx,mesaj
                mov     ah,9            ; se va tipări mesajul:
                int     21h             ; 'Exemplu program 2'
                mov     ax,4c00h        ; revenire DOS
                int     21h
```

end start

BSWAP <dest> (Byte SWAP)

care schimbă între ei octeții operandului destinație, care nu poate fi decât un registru de 32 de biți. Octeții sunt interschimbați astfel: primul octet cu ultimul şi al doilea cu al treilea. Altfel spus instrucțiunea converteşte o valoare de 32 de biți de la formatul "micul indian" la "marele indian", care sunt formate de memorare a datelor reprezentate pe mai mulți octeți.

<u>*Instrucțiuni de transfer cu stiva*</u>

- salvarea / refacerea adresei de revenire pentru apel/întreruperi; administrarea stivei pentru astfel de operații este făcută, 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.

PUSH <sursa> - depune în vârful stivei <sursa>.

$(SP) \leftarrow (SP) - 2$

$((SP)+1:(SP)) \leftarrow (sursa)$

Operandul sursă poate fi registru, memorie de 16 biți, sau o dată imediată. Dacă data imediată este octet, ea este extinsă, cu semn, la o valoare de tip cuvânt.

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

POP <dest> - extrage din vârful stivei şi duce la dest.

$(dest) \leftarrow ((SP)+1:(SP))$

$(SP) \leftarrow (SP) + 2$

Exemple:
```
pop bx
pop ds; cs nu poate fi destinație
pop beta[bp][di]
```

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

```
; salvăm registrele: ax, bx, cx
        push ax
        push bx
        push cx
; refacem aceleaşi registre cu aceleaşi valori:
        pop  cx
        pop  bx
        pop  ax
```

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

```
; memorare informații în stivă:
        mov  bp,sp  ; memorare "vârful" stivei
        push ax     ; se va depune la adresa [bp-2]
        push bx     ; [bp-4]
        push cx     ; [bp-6]
; accesul la informații se poate face, cu BP, în orice ordine
        mov  ax,[bp-2]
```

```
            mov     bx,[bp-4]
            mov     cx,[bp-6]
; descărcarea stivei se poate face modificând valoarea lui SP
            add     sp,6
; depunerea de parametrii în stivă, înainte de apelul procedurii:
            push    ax
            push    bx
            push    cx
; preluarea parametrilor din stivă, în cadrul procedurii:
; (s-a făcut abstracţie, în acest exemplu, de salvarea, din stivă, a adresei de revenire
; şi de parametrii, eventual, transmişi prin stivă)
            push    bp
            mov     bp,sp
            mov     ax,[bp+6]
            mov     bx,[bp+4]
            mov     cx,[bp+2]
; descărcarea stivei şi refacerea registrului BP, salvat :
            pop     bp
            add     sp,6
```

PUSHA permite salvarea registrelor:

AX, CX, DX, BX, SP, BP, SI, DI

POPA

care reface aceleaşi registre, bineînţeles în ordinea inversă.

La procesorul 386/486 salvarea şi refacerea se face cu instrucţiunile:

PUSHAD, POPAD

pentru registrele de 32 biţi (EAX, ECX, ..., EDI), în aceeaşi ordine.

Pentru ambele tipuri de instrucţiuni de refacere în bloc a registrelor, registrul SP (respectiv ESP) nu va prelua din stivă valoarea salvată anterior execuţiei instrucţiunii POPA/POPAD. În final conţinutul registrului SP (ESP) va creşte cu 16 (32).

Instrucţiuni de transfer specifice acumulatorului

Aceste instrucţiuni permit transferul de date între registrul acumulator (AL, AX sau EAX) şi portul de I/O, din spaţiul de I/O, specificat ca operand. Specificarea portului se poate face direct în instrucţiune, pe 8 biţi, 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
```

În principiu fiecărui periferic îi sunt asociate porturi pentru:

- citirea stării dispozitivului;
- transmiterea comenzii către periferic;

- realizarea transferului propriu-zis.

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

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

Aceste instrucțiuni transferă date la/de la un port de I/O. 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 face în funcție de atributul de dimensiune de adresă.

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

XLAT - instrucțiune de transfer dintr-o tabelă de octeți, 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

```
putere db 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, … , 225
        mov   bx, offset putere
        mov   al, 9
        xlat  putere
        ; conversie din binar în hexa (ASCII)
        ; (AL) conține o valoare în intervalul 0-15
conv            proc          near
        lea   bx,tabela_conversie
        xlat  cs:tabela_conversie; sau xlat cs:[bx]
        ret
conv            endp
tabela_conversie      label   byte
        db    '0123456789ABCDEF'
```

Programul citeşte de la tastatură codul ASCII al unei taste şi afişează acest cod (sub forma a două cifre hexa).

```
seg_date      segment
              tab_conv    db    '0123456789ABCDEF'
              mesaj  db   '-are codul ASCII-'
              tasta  db   2 dup (?), 0dh,0ah,'$'
        seg_date      ends
        seg_cod       segment
        assume cs : seg_cod, ds : seg_date
        start:
              mov   ax,seg_date
              mov   ds,ax
```

```
                mov     ah,1            ; citire cu ecou a unei taste
                int     21h             ; fără ecou este funcția 8
                mov     ah,al           ; salvare cod tastă
                lea     bx,tab_conv     ; inițializare BX, pentru XLAT
                and     al,0fh          ; se reține doar a doua tetradă
                xlat    tab_conv        ; codul ASCII al acestei tetrade
                mov     tasta+1,al      ; este cea de-a doua cifră a codului
                mov     al,ah           ; codul inițial al tastei
                mov     cl,4    ; contor număr de deplasări la dreapta
                shr     al,cl   ; deplasare logică la dreapta cu 4 biți
                xlat    tab_conv        ; conversia primei tetrade
                mov     tasta,al        ; codul ASCII al primei tetrade
                lea     dx,mesaj
                mov     ah,9            ; se tipăreşte codul tastei
                int     21h
                mov     ax,4c00h        ; revenire DOS
                int     21h
        seg_cod         ends
end start
```

pentru caracterele 'obişnuite' din setul ASCII de bază (0÷127):

cod de scanare + codul ASCII (0÷127);

setul ASCII extins (128÷255), 'ALT' + taste numerice,:

cod scanare + codul ASCII (extins) ;

pentru tastele speciale (funcționale, de deplasare, 'CTRL', 'insert', 'delete' etc.), se transmite:

cod scanare + 0 ;

Pentru utilizarea funcțiilor tastaturii se poate apela INT 16h, cu următoarele funcții (această valoare se încarcă în AH):

- 0, citire caracter (AL=cod ASCII, AH=cod scanare);
- 1, test caracter disponibil, şi returnează în
  - ZF=0, dacă este caracter disponibil în AX,
  - ZF=1, dacă nu este caracter disponibil;
- 2, citirea stării tastelor de tip "shift", care este returnată în AL (acesta este memorat şi la adresa 40h:17h).

Instrucțiuni de transfer adrese

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

```
                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. Pot apare, însă situațiile:

LEA <reg_16>,<mem_32>
LEA <reg_32>,<mem_16>
LDS <dest_reg_16>,<sursa_mem_32> (Load pointer using DS)
LES <dest_reg_16>,<sursa_mem_32> (Load pointer using ES)

Exemple:

```
1)      sir      db      . . . . . . .
        adr_sir          dd      sir
        lds      si,adr_sir
2)      octeti   db      20 dup (?)
        ptr_oct  dw      offset  octeti
                 dw      seg     octeti
        les      di, dword ptr   ptr_oct
3)      ptr_1    dd      1a2b3d4ch
        adr_w    dw      1234h, 0abcdh
        ptr_2    equ     <dword ptr adr_w>
        les      di, ptr_1       ; (ES) = 1a2bh, (DI) = 3d4ch
        lds      si, ptr_2       ; (DS) = 1234h, (SI) = 0abcdh
```

La 386/486 mai pot fi executate instrucțiunile:

LFS / LGS / LSS < reg_16/32 >, < mem_16:16/32 >

Instrucțiuni de transfer indicatori

LAHF (Load AH from Flags)

(AH) ← (SF, ZF, *, AF, *, PF, *, CF)

SAHF (Store AH into Flags)

(SF, ZF, *, AF, *, PF, *, CF) ← (AH)

PUSHF (PUSH Flags)

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

POPF

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

Instrucțiunile PUSHF şi POPF permit unei proceduri să salveze şi să restaureze indicatorii programului ce o apelează. De asemenea numai utilizând aceste instrucțiuni se poate modifica starea lui TF, pentru care nu există o instrucțiune dedicată. Modificarea se poate realiza astfel:

```
pushf                    ; se pun indicatorii în stivă
push bp                  ; salvare registru (BP)
mov bp,sp                ; se încarcă vârful stivei în (BP)
or  word ptr [bp+2], 0100h ; se pune imaginea din stivă a lui
(tf) pe 1
pop     bp               ; refacere (BP)
popf    ; şi se extrage în indicatori valoarea 1 pentru (tf)
```

În stivă, la apariția întreruperii SS (INT 1), se salvează: indicatorii şi adresa de revenire (segment + offset). Presupunând că stiva a fost adusă la forma inițiala, ştergerea lui TF se poate face astfel:

```
push    bp      ; salvare (BP)
mov     bp,sp   ; se încarcă vârful stivei în (BP)
```

```
        and     word ptr [bp+6], 0feffh ; pune (tf) pe 0, în imaginea
din stivă
        pop     bp      ; refacere (BP)
        iret            ; revenire din procedura 'single-step'
```

Instrucțiuni Aritmetice

Formatul datelor aritmetice

Operațiile aritmetice pot fi efectuate pe patru tipuri de numere:

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

Hexa	Forma binară	Numere binare		Numere zecimale	
		fără semn	cu semn	neîmpachetat	împachetat
07 H	0000 0111 B	7	+ 7	7	7
89 H	1000 1001 B	137	- 119	invalid	89
C5 H	1100 0101 B	197	- 59	invalid	invalid

Numerele binare întregi pot fi de 8 sau 16 biți, iar la 386/486 şi de 32. Numerele întregi cu semn sunt reprezentate în complement față de doi.

Numerele zecimale (BCD) sunt memorate în octeți:

- două cifre pe un octet, la formatul zecimal împachetat;
- o cifră pe un octet, la formatul zecimal neîmpachetat.

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.

- Sunt definite operațiile de înmulțire şi împărțire, specifice tipului de date binar cu semn, 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 se pot utiliza instrucțiunile de salt condiționat, pentru CF sau OF, după operația respectivă.
- Operațiile aritmetice cu numere 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 în a doua etapă este ajustat la o valoare corectă, de număr 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.
- Operațiile de adunare şi scădere pt. nr. zec. împach. se realizează în două etape, la fel ca la numerele zecimal neîmpachetate:
- instr. respectivă (+, -), va furniza un rezultat în AL;
- corecția rezultatului din AL la formatul zec. împachetat.

- Nu există instrucțiuni de ajustare pentru operațiile de înmulțire sau împărțire cu astfel de numere (zec. împachetat).
- Indicatorii, după execuția unei instrucțiuni aritmetice 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.
- Aceste instrucțiuni pot opera pe date din memorie sau registre, dar nu pot fi din registre segment, dar nu pot avea ambii operanzi din memorie. Operandul sursă poate fi şi o dată imediată.

Instrucțiunile de adunare / scădere

ADD <dest>,<sursă> (ADDition)

(dest) ← (dest) + (sursa) Modifică: OF, SF, ZF, AF, PF, CF

ADC <dest>,<sursă> (ADdition with Carry flag)

(dest) ← (dest) + (sursa) + (CF) Modifică: OF, SF, ZF, AF, PF, CF

INC <dest> (INCrement)

(dest) ← (dest) + 1 Modifică: OF, SF, ZF, AF, PF; Nu modif. CF ;

AAA (ASCII Adjust for Addition)

modifică AL la un număr 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
Modifică: AF, CF;     Nedefiniți: OF, SF, ZF, PF.
```

DAA (Decimal Adjust for Addition)

modifică AL la 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;
                Modifică: SF, ZF, AF, PF, CF; Nedefinit: OF.
```

De la procesorul 486 se mai poate executa şi instrucțiunea:

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

SUB <dest>,<sursă> (SUBtraction)

(dest) ← (dest) - (sursa); Modif: OF, SF, ZF, AF, PF, CF

SBB <dest>,<sursă> (SuBtraction with Borrow)

(dest) ← (dest) - (sursa) - (CF); M: OF, SF, ZF, AF, PF, CF

DEC <dest> (DECrement)

(dest) ← (dest) - 1; Modifică: OF, SF, ZF, AF, PF; NU: CF ;

NEG <dest>

determină complementul față de doi al destinației:

(dest) ← compl'2 (dest)

{ (dest) ← 0 - (dest) }; Modifică: 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)

De obicei după o astfel de instrucțiune urmează un salt condiționat.

AAS (ASCII Adjust for Subtraction)

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
Modifică: AF, CF; Nedefiniți: OF, SF, ZF, PF.

DAS (Decimal Adjust for Subtraction)

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;
Modifică: SF, ZF, AF, PF, CF; Nedefinit: OF.

```
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  ; inițializarea reg. DS
        mov     ds, ax          ; cu adresa de segment
        mov     cx, lung_nr1    ; contor lungime număr
        mov     si, 0           ; inițializare index elemente
        clc                     ; (CF)=0, transportul inițial
bucla:
        mov     ax, numar_2[si]
        adc     numar_1[si], ax
        inc     si              ; actualizare index element
        inc     si
        loop    bucla
        mov     ax, 4c00h       ; revenire în DOS
        int     21h
multibyte_add           ends
end     start
```

Programul adună două numere, fără semn, reprezentate pe mai multe cuvinte (octeți), dar de lungimi diferite, iar rezultatul se va depune peste numărul mai lung.

```
add_data_2   segment
        primuldw      0a8adh, 7fe2h, 0a876h,0
        lung_1dw      ($ - primul)/2
        al_doilea     dw      0fedch, 0abcdh, 0cdefh, 52deh, 5678h, 0
        lung_2dw      ($ - al_doilea)/2
add_data_2   ends
multi_add_2  segment
        assume cs: multi_add_2, ds: add_data_2
start:
        mov     ax,add_data_2
        mov     ds,ax
; se determină cel mai mare număr, unde se pune rezultatul
; vom considera că    (BX)= adresa numărului mai lung
;                     (DX)= dimensiunea numărului mai lung
;                     (BP)= adresa numărului mai scurt
;                     (CX)= dimensiunea acestuia
; numărul de octeți ai numărului mai mic va controla bucla_1
; în care ambele numere au cuvinte ce trebuie adunate
; diferența dimensiunilor celor două numere va controla bucla_2
; necesară dacă apare transport, pentru propagarea acestuia
        mov     dx,lung_2       ; presupun, inițial numărul al
        lea     bx,al_doilea    ; doilea mai mare
        mov     cx,lung_1
        lea     bp,primul
        cmp     dx,cx           ; verific presupunerea făcută
        jge     num2_mai_mare       ; dacă e respectată se continuă
        xchg    bx,bp           ; dacă nu se schimbă între ele conținutul
```

```
        xchg   cx,dx           ; perechilor de registre, respective
num2_mai_mare:
        sub    dx,cx           ; determin diferența dintre lungimile lor
        clc                    ; (CF)=0
        mov    si,0            ; se inițializează indexul elementelor
bucla_1:
        mov    ax,ds:[bp][si]
        adc    [bx][si],ax
        inc    si              ; actualizare index
        inc    si
        loop   bucla_1         ; (CX)=contor pentru adunare
        mov    cx,dx           ; contor pentru propagarea transportului
bucla_2:
        jnc    gata            ; dacă nu mai este transport de propagat
        adc    word ptr [bx][si],0
        inc    si
        inc    si
        loop   bucla_2
; trebuie testat (CF), şi dacă avem transport, adică se
; depăşeşte dimensiunea inițială a numărului mai lung,
; transportul trebuie memorat la adresa următoare,
; dacă s-a rezervat spațiu, sau semnalată depăşirea: jc eroare . . .
gata:   mov    ax,4c00h
        int    21h
multi_add_2  ends
end     start
; (DI) = va conține extensia de semn a numărului mai scurt
num2_mai_mare:
        push   cx      ; salvare lungime număr mai scurt
        mov    di,0    ; extensie de semn pentru număr pozitiv
        mov    si,bp   ; adresa de început a numărului mai scurt
        shl    cx,1    ; lungimea * 2 = număr octeți ai numărului
        add    si,cx   ; se poziționează pe primul element
        sub    si,2    ; care conține bitul de semn
        mov    ax,[si] ; şi îi testăm bitul de semn
        cmp    ax,0    ; dacă este pozitiv se continuă cu valoarea
        jp     cont    ; inițială pentru (di)=0
        mov    di,0ffffh ; altfel (di)=ffffh, extensie semn număr negativ
cont:   pop    cx      ; refacerea contorului
        sub    dx,cx   ; determin diferența dintre lungimile lor
        clc            ; (CF)=0
mov     si,0    ; se inițializează indexul elementelor
bucla_1:
        mov    ax,ds:[bp][si]
        adc    [bx][si],ax
        inc    si      ; actualizare index
```

```
        inc     si
        loop    bucla_1         ; (CX)=contor pentru adunare
        mov     cx,dx   ; contor pentru propagarea transportului
bucla_2:
        adc     word ptr [bx][si],di    ; şi a semnului numărului mai scurt
        inc     si
        inc     si
        loop    bucla_2
; în acest punct se testează OF pentru a detecta o depăşire
; a dimensiunii inițiale a numărului mai lung
```

4) Să rezolvăm aceeaşi problemă (1) pentru două numere zecimal împachetate, de aceeaşi lungime, iar rezultatul să se depună la altă adresă.

```
        mov     si,0
        mov     cx,numar_octeți
        clc
reia_adunare:
        mov     al,primul[si]
        adc     al,al_doilea[si]
        daa
        mov     rezultat[si],al
        inc     si
        loop reia_adunare
```

5) Complementarea unui număr reprezentat pe mai multe cuvinte.

complement față de 2 pentru număr = 2^n *- număr*;

```
        mov     cx,lung_numar ; contor număr cuvinte
        clc                     ; inițializare transport cu 0
compl:
        mov     ax,0            ; echivalentul lui 2^n
        sbb     ax,numar [si]   ; se efectuează scăderea, cu propagarea împrumutului
        mov     numar[si],ax    ; se depune rezultatul
        inc     si              ; actualizare index cuvânt următor
        inc     si
        loop    compl
```

Dacă numărul inițial este valoarea minimă negativă (80 00 . . .00H), după complementare valoarea va fi aceeaşi.

```
.model small
.stack
.data
numar   db 23h, 45h, 80h
lung_numar      equ $-numar
mes_dep         db 'Depasire compl2 (numar minim negativ)', 0dh, 0ah, '$'
.code
        mov ax, @data
        mov ds, ax
```

```
        mov cx, lung_numar ; contor număr octeți
        mov si, 0                     ; index la octetul curent
compl2:
        neg numar[si]; complementare față de 2
        inc si          ; actualizare index pentru octetul următor
        jc gasit_bit_unu              ; până la prima unitate
        loop compl2   ; dacă nu s-a găsit primul 1 se reia compl'2
        jmp gata               ; dacă se execută 'jmp' operandul a fost 0
gasit_bit_unu:
dec cx          ; actualizare contor
        jcxz gata       ; dacă a fost ultimul octet, s-a terminat operația
compl1:
        not numar[si] ; se continuă cu complementarea față de 1
        inc si
        loop compl1
gata:
        cmp numar[si-1], 80h
        jne quit
        lea si, numar  ; test depăşire pentru 80.00. . . 00h
        mov cx, lung_numar
        mov al, [si]           ; se face OR între toți octeții rezultatului
        inc si          ; şi dacă este 80H -> depăşire compl'2
        dec cx          ; pentru valoarea minimă negativă
        jcxz gata_or
test_dep:
        or al, [si]
        inc si
        loop test_dep
gata_or:
        cmp al, 80h
        jne quit
        lea dx, mes_dep
        mov ah, 9
        int 21h
quit:
        mov ax, 4c00h
        int 21h
end
```

Instrucțiunile de înmulțire / împărțire

MUL <sursa> ; **MUL <acumulator>, <sursa>**

<sursa> * <AL, AX sau EAX>

rezultat: (AX), (DX, AX), (EDX, EAX).

Depăşire OF=CF=1, altfel OF=CF=0. Ceilalți indicatori sunt nedefiniți (SF, ZF, AF, PF).

```
        mov     al,op1_octet    ; înmulțitorul de tip octet
```

mov ah,0 ; se extinde înmulțitorul la cuvânt, fără semn
mul op2_cuv ; se realizează înmulțirea

IMUL <sursa> ; IMUL <acumulator>, <sursa>

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

AAM (Ascii Adjust for Multiply)

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

DIV < sursa > sau **DIV < acc >,< sursa >**

(temp) ← (numărător) / (numitor)
if (temp) > MAX then
(cât), (rest) - nedefiniți
(SP) ← (SP) - 2; ((SP)+1:(SP)) ← indicatori; (tf), (if) ← 0;
(SP) ← (SP) - 2; ((SP)+1:(SP)) ← (CS); (CS) ← (2)
(SP) ← (SP) - 2; ((SP)+1:(SP)) ← (IP); (IP) ← (0)
else (cât) ← (temp)
(rest) ← (numărător) mod (numitor)
MAX = FFH/octet, FFFFH/cuv., FFFFFFFFH/dublu cuv.

mov al,num_octet ; operand octet
mov ah,0 ; extensia deîmpărțitului/ cbw
div divizor_octet ; idiv
mov ax,num_cuvant ; operand cuvânt
div divizor_octet ; idiv
mov ax,num_cuvant ; operand cuvânt
mov dx,0 ; extensia deîmpărțitului/ cwd
div divizor_cuvant ; idiv
mov eax,cuvant_dublu ; operand dublu cuvânt
mov edx,0 ; extensie a deîmpărțitului în (EDX)
div divizor_dublu_cuvant ; idiv

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

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

Constanta MAX are valorile: 7FH, 7FFFH şi 7FFFFFFFH pentru câturi pozitive, şi 80H, 8000H, 80000000H cele negative.

AAD (ASCII Adjust for Division)

(AL) ← (AH) * 0AH + (AL)

(AH) ← 0

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

```
mov   al,7     ; înmulțire 7 * 7
mov   cl,7
mul   cl       ; se obține (ah,al) = (0h, 31h)
aam            ; corecție  (ah,al) = (04h, 09h)
mov   ah,8     ; deîmpărțit 89
mov   al,9     ;
aad            ; corecție deîmpărțit zecimal la binar
mov   bl,4     ; împărțitor 4
div   bl       ; rezultat (ah,al) = (01h, 16h)
mov   dh,ah    ; se salvează restul (dh) = 01h
aam            ; corecție (ah,al) = (02h, 02h)
```

- Afişare AL în zecimal (AL<100)

```
aam            ; corecție la format BCD: ah = cifra zeci
add   ax,3030h ; al = cifra unități. Conversie la ASCII
mov   dx,ax    ; se salvează codurile ASCII ale cifrelor
xchg  dh,dl    ; se depune în (dl) cod ASCII pt. cifra zeci
mov   ah,2     ; se apelează funcția 2 din DOS
int   21h      ; de tipărire a caracterului din (dl)
xchg  dh,dl    ;se depune în (dl) cod ASCII cifra unități

mov   ah, 2
int   21h
```

Afişarea în zecimal a conținutului registrului AX:

```
; (AX) conține o valoare care este în intervalul 0÷65535
; dacă se împarte numărul la 100 se obține ca prim rest
; o valoare ce conține ultimele două cifre (zeci, unități)
; dacă se împarte câtul anterior din nou la 100 se va obține
; o valoare ce conține următoarele două cifre (mii, sute)
; iar acest ultim rest va avea valoarea primei cifre
; a zecilor de mii, care este în intervalul: 0 - 6.
        mov   bx,100      ; împărțitor
        mov   dx,0        ; extensie semn deîmpărțit pozitiv
        div   bx          ; dx = ultimele 2 cifre (zeci, unități)
        mov   cx,dx       ; ax = primele 3 cifre, cx = ultimele 2 cifre
        mov   dx,0        ; extensie semn deîmpărțit pozitiv
div     bx                ; dx = cifrele: mii, sute; ax = zeci de mii
        push dx           ; se salvează dx deoarece va fi modificat
        add   al,30h      ; se tipăreşte cifra zecilor de mii(0÷6)
        mov   dl,al       ; caracterul de tipărit în DL
        mov   ah,2        ; se apelează funcția 2 din DOS
        int   21h         ; care tipăreşte caracterul din (DL)
```

```
pop     ax              ; se iau din stiva valoarea pt. mii, sute
aam                     ; conversie la format zecimal neîmpachetat
add     ax,3030h        ; conversie la cod ASCII
mov     dx,ax           ; se salvează cele două cifre
xchg    dh,dl           ; se tipăreşte cifra miilor
mov     ah,2
int     21h
xchg    dh,dl           ; se tipăreşte cifra sutelor
mov     ah,2
int     21h
mov     ax,cx           ; ultimele două cifre: zeci, unităţi
aam                     ; conversie la format zecimal neîmpachetat
add     ax,3030h        ; conversie la cod ASCII
mov     dx,ax           ; se salvează cele două cifre
xchg    dh,dl           ; se tipăreşte cifra zecilor
mov     ah,2
int     21h
xchg    dh,dl           ; se tipăreşte cifra unităţilor
mov     ah,2
int     21h
```

Pentru a tipări valoarea din (AX) cu semn se va testa primul bit din AX, care reprezintă semnul şi se tipăreşte. Dacă numărul este negativ se va complementa faţă de 2, după care programul este acelaşi ca în exemplul anterior.

```
        cmp     ax,0    ; test semn
        mov     bx,ax   ; salvare valoare
        jns     cont    ; dacă e pozitiv se continuă
        mov     dl,'-'  ; se tipăreşte semnul
        mov     ah,2
        int     21h
        mov     ax,bx   ; refacere număr
        neg     ax      ; complementare (transformare în pozitiv)
cont:   .........
```

Instrucţiuni de prelucrare la nivel de bit

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

Instrucţiuni logice

OF şi CF ← 0;
AF - nedefinit;
SF, ZF, PF - sunt poziţionaţi conform rezultatului instrucţiunii.

NOT <dest> 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.

Instrucțiuni de testare şi modificare de bit.

Bitul testat este specificat prin deplasamentul său (al doilea operand, dată de 8 biți sau reg_16/32) față de bitul cel mai puțin semnificativ al destinației (reg/mem_6/32). Ceilalți indicatori sunt nedefiniți.

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

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

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

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

Instrucțiuni de scanare pe bit.

Aceste instrucțiuni permit scanarea biților din cel de-al doilea operand direct, începând cu bitul mai puțin semnificativ (*forward*) sau invers (*reverse*), pentru a determina primul bit egal cu 1. Dacă toți biții sunt zero ZF=1, altfel ZF=0, şi registrul destinație va reține indexul primului bit 1 din operandul sursă (reg/mem de aceeaşi dimensiune cu reg. Dest.). Modif. ZF, iar toți ceilalți sunt nedefiniți.

- scanare directă:

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

- scanare inversă:

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

Instrucțiuni de setare condiționată.

Pun octetul pe zero sau unu, în funcție de una din cele 16 condiții definite de indicatori. Operandul destinație poate fi reg/mem. Aceste instrucțiuni sunt folosite pentru implementarea expresiilor booleene, din limbajele de nivel înalt.

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

SETE / SETZ ZF = 1
SETNE / SETNZ ZF = 0
SETL / SETNGE SF <> OF, valori cu semn
SETLE / SETNG SF <> OF sau ZF = 1, valori cu semn
SETNL / SETGE SF = OF, valori cu semn
SETNLE / SETG SF = OF şi ZF = 0, valori cu semn
SETB / SETNAE / SETC CF = 1, valori fără semn

SETBE / SETNA CF = 1 sau ZF = 1,fără semn
SETNB / SETAE / SETNC CF = 0, valori fără semn
SETNBE / SETA CF = 0 şi ZF = 0,fără semn
SETO / SETNO OF = 1 / respectiv OF = 0
SETP / SETPE PF = 1, adică paritate pară
SETNP / SETPO PF = 0, adică paritate impară
SETS / SETNS SF = 1 / respectiv SF = 0

Instrucţiuni de deplasare

Deplasările pot fi aritmetice sau logice. Se poate deplasa operandul destinaţie cu până la 31 de biţi, corespunzător operandului contor, codificat în instrucţiune (sunt luaţi în considerare numai ultimii 5 biţi ai acestuia).

OF = este nedefinit într-o deplasare pe mai mulţi biţi;
= este poziţionat pt. deplasare de un bit;
CF = ultimul bit deplasat în afara operandului destinaţie;
AF = nedefinit;
SF, ZF, PF = modificaţi conform rezultatului.

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

Poziţiile eliberate devin zero; între cele două mnemonici nu există nici o deosebire. Bitul cel mai semnificativ se deplasează în CF.

SHR <dest>, <contor> (SHift logical Right)
SAR <dest>, <contor> (Shift Arithmetic Right)

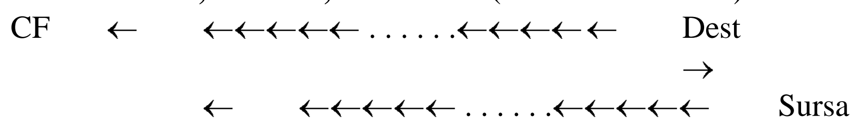
Bitul cel mai semnificativ îşi păstrează vechea valoare, dar este şi deplasat spre dreapta (extensie de semn), ceea ce înseamnă că nu poate să apară depăşire (schimbarea de semn a rezultatului) şi (OF)=0 la o deplasare de 1 bit.

Trebuie menţionat că SAR nu furnizează acelaşi rezultat ca instrucţiunea IDIV, pentru aceeaşi operanzi, dacă deîmpărţitul este negativ, şi sunt deplasaţi (SAR) în afara operandului biţi egali cu 1. De exemplu -5 deplasat la dreapta (SAR), cu un bit va furniza -3, în timp ce împărţirea întreagă (IDIV) va furniza valoarea -2: (-5) = (11111011) → (11111101) = (-3)

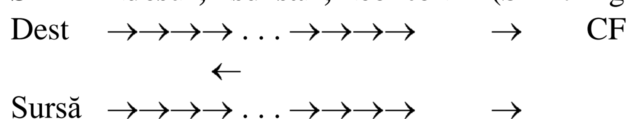
Diferenţa dintre cele două instrucţiuni este că IDIV trunchiază toate câturile către zero, în timp ce SAR trunchiază numerele pozitive către zero, iar pe cele negative către infinit negativ.

La 386/486 pot fi realizate instrucţiuni de deplasare dublă, pe cuvânt sau dublu cuvânt. Operandul destinaţie poate fi reg/ mem, iar sursa reg. general. Rezultatul înlocuieşte operandul destinaţie, iar contorul numărului de biţi deplasaţi este specificat de registrul CL, sau ca o valoare imediată de 8 biţi. Indicatorii sunt poziţionaţi la fel ca la deplasările anterioare.

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



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



Registrul sursă nu se modifică. 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 (circulați) înapoi către celălalt capăt al operandului. Începând de la procesoarele 286 contorul, chiar dacă este diferit de 1, el poate fi specificat în instrucțiune, ca dată imediată.

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)

CF ← Bn ←← . . . . . .←← B0 ← Bn

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

B0 → CF Bn →→ . . . . . . →→ B0 → Bn

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

B0 ← CF ← Bn ←← . . . . . . ←← B0 ← CF

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

B0 → CF → Bn →→ . . . . . . →→ B0 → CF

1) Înmulțirea cu 20 a uni număr din registrul AX:

```
; 20 = 16 + 4 = 2^4 + 2^2
; deci 2 deplasări la stânga, se salvează rezultatul,
mov   cl, 2   ; care reprezintă * 4, şi încă două deplasări
shl   ax, cl
mov   bx, ax  ; se salvează *4 în (BX)
      mov   cl, 2   ; contorul pentru încă două deplasări (*16)
shl   ax, cl  ; care realizează * 16, şi se adună cele două
add   ax, bx  ; rezultate parțiale
```

2) Împărțirea la 1024 a valorii din registrul AX:

```
mov   cl, 2   ; 1024 = 2^10 = 2^8 * 2^2
sar   ah, cl  ; după 8 deplasări la dreapta AL se pierde
xchg ah, al   ; (sau mov ah, al), e suficient să deplasăm AH la
cbw           ; dreapta cu 2, şi apoi să-i facem extensia de semn
```

3) Împărțirea unui număr de 32 biți (64) la o putere a lui 2, de exemplu la $64 = 2^6$:

```
      mem_32      dd      50000;
      . . . . .  ; trebuie deplasat numărul la dr. cu 6 biți
            mov   cx, 6         ; contorul numărului de deplasări
iar:  sar   word ptr mem_32[2], 1      ; bitul 17 → (CF)
```

```
        rcr     word   ptr mem_32[0], 1      ; (CF) → bitul 16
        loop iar
```

4) Determinarea numărului de biți egali cu 1 dintr-o variabilă.

```
.model small
.stack  100h
.data
        lung_var        equ     8
        variabila       dw      lung_var        dup (0acfh)
        nr_unitati      db      ?
.code
assume          cs: @code, ds: @data
start:  mov     ax, @data
        mov     ds, ax
        mov     si, 0           ; indexul curent al cuvintelor din variabila
        mov     dl, lung_var    ; contor număr de cuvinte
        mov     bl, 0           ; contor număr de unități găsite
bucla2: mov     cx, 16  ; contorul de biți pentru un cuvânt
        mov     ax, variabila[si]       ; se citeşte cuvântul curent
bucla1:
        rcl     ax, 1                   ; o rotație pentru a deplasa un bit
        adc     bl, 0                   ; se contorizează numărul de unități
        loop    bucla1                  ; pentru un cuvânt
        add     si, type variabila      ; se actualizează indexul
        dec     dl                      ; se testează dacă mai sunt cuvinte
        jnz     bucla2                  ; dacă da se reia citirea cuvintelor
        mov     nr_unitati, bl          ; dacă nu se depune rezultatul
        mov     ax, 4C00h               ; revenire DOS
        int     21h
end
```

4a). Aceeaşi problemă poate fi rezolvată utilizând instrucțiunile *BSF* şi *BTR*, astfel:

```
        mov     cx, lung_var    ; contor dimensiune variabilă
        mov     bl, 0           ; contor unități
        mov     si, 0           ; index cuvinte
bucla: mov      ax, variabila[si]
 scanare:
        bsf     dx, ax          ; determină primul bit 1 (indexul în DX)
        jz      gata_cuv        ; dacă ZF=0, toți biții sunt 0
        btr     ax, dx          ; transferă 1 din AX, din poziția DX în CF,
        adc     bl, 0           ; iar bitul =0, sau inc bl, numără unitățile
        jmp     scanare         ; se reia scanarea cuvântului curent
gata_cuv:
        add     si, 2           ; indexul cuvântului următor
        loop bucla      ; decrementare contor număr de cuv. variabilă
                        ; dacă nu s-a terminat variabila ia cuv. următor
```

5) Tipărirea conținutului registrului DX, în format octal:

```
tip_car         proc    far
; procedura tipăreşte caracterul al cărui cod ASCII
; îl primeşte în registrul AL
                push    dx          ; se salvează registrul de lucru DX
                mov     dl, al      ; se apelează funcția 2 din DOS care
                mov     ah, 2       ; tipăreşte caracterul al cărui cod
                int     21h         ; ASCII se transmite în registrul DL
                pop     dx          ; se reface registrul salvat
                ret
tip_car         endp
tip_octal       proc    far
        ; tipăreşte în octal valoarea, fără semn transmisă în DX
                push    cx      ; se salvează registrele de lucru
                push    ax
; prima cifră de tipărit este doar de 1 bit
        rol     dx, 1           ; care este rotit pe ultimul bit
        mov     al, dl          ; şi dusă în registrul AL
        and     al, 1           ; se şterg ceilalți biți
        add     al, 30h         ; este convertită la codul ASCII
        call    tip_car         ; şi se tipăreşte
                                ; următoarele 5 cifre sunt de câte 3 biți
        mov     cx, 5           ; contor număr de cifre de tipărit
octal1: push    cx              ; se salvează contorul în stivă
        mov     cl, 3           ; contor număr de rotiri la stânga
        rol     dx, cl          ; cifra este rotită pe ultimii 3 biți
        mov     al, dl          ; şi adusă în AL
        and     al, 7           ; sunt şterşi ceilalți biți
        add     al, 30h         ; şi convertită la codul ASCII
        call    tip_car                 ; tipărirea cifrei

        pop     cx              ; se citeşte valoarea contorului de cifre
        loop    octal1
        pop     ax              ; se refac registrele salvate în stivă
        pop     cx
        ret
tip_octal       endp
```