Programarea microprocesorului Intel 8086.
Instructiuni de transfer al comenzii

Scopul lucrării

a) Instructiuni de salt.
b) Instructiuni de test.
c) Instructiuni iterative.

1. Instructiuni de salt

Instructiunile de salt pot fi de salt neconditionat si de salt conditionat.

1.1. Instructiunea de salt neconditionat

Instructiunea de salt neconditionat are forma:

JMP *pozitie*

unde *pozitie* poate fi *o eticheta, un registru, o variabila*, etc.

Efectul ei este incarcarea in registrul IP si eventual in registrul CS a unei adrese noi (obtinuta din *pozitie*), a urmatoarei instructiuni care va fi executata.

1.2. Instructiuni de salt conditionat

Instructiunile de salt conditionat au forma:

J*conditie pozitie*

unde dupa ce *conditie* este testata, se executa saltul doar daca aceasta este indeplinita.

Formele particulare ale instructiunii de salt conditionat sunt:

1. Pentru orice tip de valori:

a) Salt conditionat de *CX=0* (registrul *contor* are continut nul):

J**CXZ** *pozitie*

b) Salt conditionat de *CF=1* (*transport* la operatia anterioara):

J**C** *pozitie*

c) Salt conditionat de *CF=0* (*fara transport* la operatia anterioara):

J**NC** *pozitie*

d) Salt conditionat de *ZF=1* (rezultat *zero* la operatia anterioara):

J**E** *pozitie*
J**Z** *pozitie*

e) Salt conditionat de *ZF=0* (rezultat *nenul* la operatia anterioara):

J**NE** *pozitie*
J**NZ** *pozitie*

f) Salt conditionat de *PF=1* (rezultat *par* la operatia anterioara):

J**P** *pozitie*
J**PE** *pozitie*

g) Salt conditionat de *PF=0* (rezultat *impar* la operatia anterioara):

J**NP** *pozitie*
J**PO** *pozitie*

2. Pentru valori fara semn (naturale)

a) Salt conditionat de *(CF=0 AND ZF=0)*, adica rezultat "*mai mare*" (> 0) la operatia anterioara:

J**A** *pozitie*
J**NBE** *pozitie*

b) Salt conditionat de *CF=0*, adica rezultat "*mai mare sau egal*" (>= 0) la operatia anterioara:

J**AE** *pozitie*
J**NB** *pozitie*

c) Salt conditionat de *CF=1*, adica rezultat "*mai mic*" (< 0) la operatia anterioara:

J**B** *pozitie*
J**NAE** *pozitie*

d) Salt conditionat de *(CF=1 AND ZF=1)*, adica rezultat "*mai mic sau egal*" (<= 0) la operatia anterioara:

JBE *pozitie*
JNA *pozitie*

3. Pentru valori cu semn (intregi):

a) Salt conditionat de (*SF=OF AND ZF=0)*, adica rezultat "*mai mare*" (> 0) la operatia anterioara:

JG *pozitie*
JNLE *pozitie*

b) Salt conditionat de *SF=OF*, adica rezultat "*mai mare sau egal*" (>= 0) la operatia anterioara:

JGE *pozitie*
JNL *pozitie*

c) Salt conditionat de *SF<>OF*, adica rezultat "*mai mic*" (< 0) la operatia anterioara:

JL *pozitie*
JNGE *pozitie*

d) Salt conditionat de *(SF<>OF AND ZF=1)*, adica rezultat "*mai mic sau egal*" (<= 0) la operatia anterioara:

JLE *pozitie*
JNG *pozitie*

e) Salt conditionat de *SF=1* (rezultat *negativ* la operatia anterioara):

JS *pozitie*

f) Salt conditionat de *SF=0* (rezultat *pozitiv* la operatia anterioara):

JNS *pozitie*

g) Salt conditionat de O*F=1* (*depasire de gama* la operatia anterioara):

JO *pozitie*

h) Salt conditionat de *OF=0* (*fara depasire de gama* la operatia anterioara):

JNO *pozitie*

Cu ajutorul acestor instructiuni se pot realiza *structuri de decizie* in program, asemanatoare instructiunilor *if* din limbajele de nivel inalt (C, Pascal, etc.).

Exemplul 1 (calculul modulului unei sume):

```
        ADD  AX, DX      ; suma
        JGE  et1         ; daca AX >= 0, ramane neschimbat
        NEG  AX          ; altfel AX = - AX
et1:    MOV  rez, AX     ; variabila rez ia valoarea AX
```

Exemplul 2 (indicarea prin 'O" rezultatul nul si prin 'N' rezultatul nenul al unei sume):

```
        MOV  rez, 'O' ; rezultat presupus nul
        ADD  AX, DX      ; suma
        JZ   et1         ; daca AX = 0 salt
        MOV  rez, 'N' ; altfel rezultat nenul
et1:                     ; instructiunea urmatoare
```

2. Instructiuni de test

In cazul in care se doreste testarea unor conditii pentru a fi utilizate intr-o instructiune de salt conditionat fara ca rezultatul testului sa fie incarcat intr-un registru sau stocat in memorie se pot utiliza instructiunile de test CMP si TEST.

Aceste instructiuni au un efect asemanator cu instructiunile SUB respectiv AND, dar fara ca rezultatul sa fie incarcat intr-un registru sau stocat in memorie.

1. Instructiunea CMP are forma:

CMP *operand1, operand2*

si efectul: *operand1 - operand2*

Exemplu (initial *AX = 1000 H* si *BX = 800 H*):

```
        CMP  AX, BX
Efect:
        0001 0000 0000 0000 B     =  1000 H -
        0000 1000 0000 0000 B     =  0800 H
        ---------------------------------------------
        0000 1000 0000 0000 B     =  0800 H
```

In urma executiei instructiunii: *AX = 1000 H, BX = 800 H, OF=0, SF=0, ZF=0* si *CF* = 0.

2. Instructiunea TEST are forma:

TEST *operand1, operand2*

si efectul: *operand1* AND *operand2*

Exemplu (initial *AX = 5555 H* si *BX = 6666 H*):

TEST AX, BX

Efect:

```
0101 0101 0101 0101 B      = 5555 H x
0110 0110 0110 0110 B      = 6666 H
-----------------------------------------------
0100 0100 0100 0100 B      = 4444 H
```

In urma executiei instructiunii: *AX = 5555 H, BX = 6666 H, SF=0* si *ZF=0.*

3. Instructiuni iterative

3.1. Instructiunea iterativa neconditionata

Instructiunea iterativa simpla are forma:

LOOP *pozitie*

unde *pozitie* poate fi o eticheta, un registru, o variabila, etc.

Prima operatie efectuata in acest caz este decrementarea registrului CX (*contor* sau *numarator*). Daca continutul acestuia nu devine 0 dupa decrementare, efectul este identic instructiunii JMP. Daca in urma decrementarii CX devine 0, instructiunea LOOP nu are nici un alt efect.

Astfel, efectul instructiunii:

LOOP *etich*

este echivalent cu al secventei:

DEC CX
JNZ *etich*

Instructiunea LOOP se utilizeaza intr-o *structura de program tipica*:

```
            MOV  CX, nr
etich:  ...                     ; secventa de
            ...                         ; instructiuni
            LOOP etich
```

care permite executia repetata de *nr* ori a secventei de instructiuni.

Aceasta instructiune permite realizarea *structurilor iterative* de program.

Exemplu (calculul in DX a numarului de biti 1 din cuvintul *cuv*, varianta prin bitul Carry (CF), ciclu cu test final, numar fix = 16 iteratii):

```
            MOV   AX, cuv
            XOR  DX, DX
            MOV   CX, 16        ; pentru CX de la 16 la 1
et1:    SHL AX, 1           ; CF = MSb
            JNC    et2                  ; daca CF = 0 salt
            INC  DX                     ; altfel (CF=0), numara un 1
et2:    LOOP et1            ; repeta
```

3.2. Instructiuni iterative conditionate

Instructiunile iterative conditionate au formele:

a) Bucla conditionata de rezultat nul (ZF=1):

```
            LOOPZ  etich
sau         LOOPE  etich
```

b) Bucla conditionata de rezultat nenul (ZF=0):

```
            LOOPNZ  etich
sau         LOOPNE  etich
```

Structura tipica in care se utilizeaza instructiunea LOOP*conditie*:

```
            MOV  CX, nr
etich:  ...                     ; secventa de
            ...                         ; instructiuni
            LOOPconditie  etich
```

permite executia repetata de **cel mult** *nr* ori a secventei de instructiuni, deoarece repetarea este intrerupta in cazul in care *conditie* este indeplinita.

Astfel, efectul instructiunii:

LOOP*conditie etich*

este echivalent cu al secventei:

J*conditie etich*
DEC CX
JNZ *etich*

Exemplu (cautarea bitului 0 cel mai semnificativ in *cuv,* ciclu cu test final cu dubla conditie):

```
        MOV  CX, 15 ; CX = pozitia bitului testat
        MOV  BX, 1  ; BX = masca
et:     ROR  BX, 1  ; roteste BX cu o pozitie catre dr.
        TEST  cuv, BX       ; AND bit cu bit fara stocarea rez.
        LOOPNZ  et          ; pana cand ((bit testat=0)OR(CX=0))
```

4. Exemple de programe

1. Program de calcul al produsului a doua cifre hexazecimale citite de la tastatura si de afisare a rezultatului pe ecran.

```
data  segment
numeprog        db 25 dup(0ah),'Calculul produsului a doua valori $'
cifra1          db 2 dup(0ah),0dh,' Prima valoare: $'
cifra2          db 2 dup(0ah),0dh,' A doua valoare: $'
eroarecifra     db 2 dup(0ah),0dh,' Valorile nu sunt cifre hexa ! $'
rezultat        db 2 dup(0ah),0dh,' Rezultatul: $'
data  ends

assume cs:cod, ds:data
cod   segment
start:
        mov     ax, data
        mov     ds, ax
        mov     dx, offset numeprog   ; afisare sir caractere
        mov     ah, 9                 ; (nume program)
        int     21h
        mov     dx, offset cifra1     ; afisare sir caractere
        mov     ah, 9                 ; (mesaj cifra 1)
        int     21h

        mov     ah, 1                 ; citire caracter cu ecou pe ecran
```

```
        int     21h                     ; (citire prima cifra in al)

; conversie caracter -> cifra
        cmp     al, 30h         ; caracter >= '0' ?
        jl      er1             ; nu => eroare
        cmp     al, 3ah         ; da => caracter <= '9' ?
        jnl     urm11           ; nu => continuare cu urmatorul test
        sub     al, 30h  ; da => ajustare (caracterul e cifra 0..9)
        jmp     sf1             ; salt la sfirsit conversie
urm11:cmp   al, 41h             ; caracter >= 'A' ?
        jl      er1             ; nu => eroare
        cmp     al, 47h         ; da => litera <= 'F' ?
        jnl     urm12           ; nu => continuare cu urmatorul test
        sub     al, 37h  ; da => ajustare (caracterul e cifra A..F)
        jmp     sf1             ; salt la sfirsit conversie
urm12:cmp   al, 61h             ; caracter >= 'a' ?
        jl      er1             ; nu => eroare
        cmp     al, 67h         ; da => litera <= 'f' ?
        jnl     er1             ; nu => continuare cu urmatorul test
        sub     al, 57h  ; da => ajustare (caracterul e cifra a..f)
        jmp     sf1
er1:mov     dx, offset eroarecifra
        mov     ah, 9                   ; afisare sir caractere
        int     21h                     ; (mesaj eroare)
sf1:
        mov     bl, al                  ; cifra 1 in bl

        mov     dx, offset cifra2       ; afisare sir caractere
        mov     ah, 9                           ; (mesaj cifra 2)
        int     21h
        mov     ah, 1                   ; citire caracter cu ecou pe ecran
        int     21h                     ; (citire a doua cifra in al)

; conversie caracter -> cifra
        cmp     al, 30h         ; caracter >= '0' ?
        jl      er2             ; nu => eroare
        cmp     al, 3ah         ; da => caracter <= '9' ?
        jnl     urm21           ; nu => continuare cu urmatorul test
        sub     al, 30h  ; da => ajustare (caracterul e cifra 0..9)
        jmp     sf2             ; salt la sfirsit conversie
urm21:cmp   al, 41h             ; caracter >= 'A' ?
        jl      er2             ; nu => eroare
        cmp     al, 47h         ; da => litera <= 'F' ?
        jnl     urm22           ; nu => continuare cu urmatorul test
        sub     al, 37h  ; da => ajustare (caracterul e cifra A..F)
        jmp     sf2             ; salt la sfirsit conversie
```

```
urm22:cmp   al, 61h             ; caracter >= 'a' ?
        jl      er2             ; nu => eroare
        cmp     al, 67h         ; da => litera <= 'f' ?
        jnl     er2             ; nu => continuare cu urmatorul test
        sub     al, 57h  ; da => ajustare (caracterul e cifra a..f)
        jmp     sf2
er2:mov     dx, offset eroarecifra
        mov     ah, 9                           ; afisare sir caractere
        int     21h                             ; (mesaj eroare)
sf2:

; calcul propriu-zis
        mov     dl, bl                          ; cifra 1
        mul     dl                              ; (cifra 2 in al)
        mov     bx, ax                          ; rezultat in bx

        mov     dx, offset rezultat
        mov     ah, 9                           ; afisare sir caractere
        int     21h                             ; (mesaj rezultat)

    mov     cl,  4                              ; separare cifra 1
        mov       al, bl
    shr     al, cl

; conversie cifra 1 -> caracter
        cmp     al, 0ah                 ; cifra = 0..9 ?
        jl      urm3            ; da => ajustare (urm4)
        cmp     al, 10h                 ; nu => cifra = A..F ?
        jnl     er3
        add     al, 37h                 ; da => ajustare
        jmp     sf3
urm3:add    al, 30h
        jmp     sf3
er3:
sf3:
    mov     dl, al
        mov     ah, 2                           ; afisare cifra 1
        int     21h

        mov       al, bl                        ; separare cifra 2
    shl     al, cl
    shr     al, cl

; conversie cifra 2 -> caracter
        cmp     al, 0ah                 ; cifra = 0..9 ?
        jl      urm4            ; da => ajustare (urm4)
```

```
        cmp     al, 10h                 ; nu => cifra = A..F ?
        jnl     er4
        add     al, 37h                 ; da => ajustare
        jmp     sf4
urm4:add    al, 30h
        jmp     sf4
er4:
sf4:
   mov     dl, al
        mov     ah, 2                           ; afisare cifra 2
        int     21h

        mov     ah, 8                   ; citire caracter fara ecou pe ecran
        int     21h                     ; (Enter)
        mov     ah, 4ch                 ; exit
        int     21h
        cod     ends
        end     start
```

2. Program de ordonare crescatoare a 2 valori dintr-un sir de numere naturale.

```
DATA SEGMENT
        vect    DW      8766 H, 5678 H, 0ABC3 H, 0B44 H
DATA ENDS
ASSUME CS: COD, DS: DATA
COD   SEGMENT
START:
        MOV  AX, DATA
        MOV  DS, AX
        MOV  BX, OFFSET VECT
        MOV  AX, [BX]                   ; AX ia valoarea vect[0]
        CMP  AX, [BX+2]         ; AX (vect[0]) e comparat cu vect[2]
        JB   inord              ; daca vect[0] < vect[1] atunci sunt
                                        ; in ordine crescatoare (salt)
        XCHG AX, [BX+2]         ; altfel se permuta vect[0] cu
        MOV  [BX], AX                   ; vect[2]
inord:MOV  AH, 4CH
        INT  21H
COD   ENDS
        END  START
```

3. Program de ordonare crescatoare a 2 valori dintr-un sir de intregi cu semn.

```
DATA SEGMENT
```

```
        vect    DW      8766 H, 5678 H, 0ABC3 H, 0B44 H
DATA ENDS
ASSUME CS: COD, DS: DATA
COD     SEGMENT
START:
        MOV  AX, DATA
        MOV  DS, AX
        MOV  BX, OFFSET VECT
        MOV  AX, [BX]                   ; AX ia valoarea vect[0]
        CMP  AX, [BX+2]         ; AX (vect[0]) e comparat cu vect[2]
        JL   inord              ; daca vect[0] < vect[1] atunci sunt
                                        ; in ordine crescatoare (salt)
        XCHG AX, [BX+2]         ; altfel se permuta vect[0] cu
        MOV  [BX], AX                   ; vect[2]
inord:MOV AH, 4CH
        INT  21H
COD     ENDS
        END  START
```

4. Program de calcul al numarului de biti egali cu 1 din cuvintul cuv in registrul DX.

a) Varianta prin bitul Carry, ciclu cu test final, numar fix = 16 iteratii:

```
DATA SEGMENT
        cuv     DW      0e360 H
DATA ENDS
ASSUME CS: COD, DS: DATA
COD     SEGMENT
START:
        MOV  AX, DATA
        MOV  DS, AX
        MOV   AX, cuv           ; incarca cuv in registrul acumulator
        XOR  DX, DX             ; resetare contor (DX = 0)
        MOV   CX, 16        ; pentru CX de la 16 la 1
et1:    SHL    AX, 1        ;   deplasare la stanga cu o pozitie
        JNC    et2                  ;   salt daca nu s-a obtinut transport
        INC  DX                     ;   incrementare contor daca a fost CF
et2:    LOOP et1            ; repeta
        MOV  AH, 4CH
        INT  21H
COD     ENDS
        END  START
```

b) Varianta prin bitul Carry, ciclu cu test initial, numar *fix* de iteratii:

```
DATA SEGMENT
        cuv     DW      0e360 H
DATA ENDS
ASSUME CS: COD, DS: DATA
COD   SEGMENT
START:
        MOV  AX, DATA
        MOV  DS, AX
        MOV   AX, cuv                   ; incarca cuv in registrul acumulator
        XOR  DX, DX                     ; resetare contor (DX = 0)
        MOV  CX, 17             ; pentru CX de la 17 la 1
et1:    DEC  CX                 ; CX = CX - 1
        JCXZ et2                ; daca CX = 0 salt la et2 (gata)
        SHL  AX, 1              ; altfel deplasare la stanga cu o pozitie
        JNC  et1                        ; salt daca nu s-a obtinut transport
        INC  DX                         ; incrementare contor daca a fost CF
        JMP  et1                ; repeta
et2: MOV  AH, 4CH
        INT  21H
COD   ENDS
        END  START
```

c) Varianta prin bitul Carry, ciclu cu test final, numar *variabil* de iteratii:

```
DATA SEGMENT
        cuv     DW      0e360 H
DATA ENDS
ASSUME CS: COD, DS: DATA
COD   SEGMENT
START:
        MOV  AX, DATA
        MOV  DS, AX
        MOV   AX, cuv                   ; incarca cuv in registrul acumulator
        XOR  DX, DX                     ; resetare contor (DX = 0)
et1:    SHL  AX, 1              ; repeta deplasare la stanga cu o pozitie
        JNC  et2                        ;   salt daca nu s-a obtinut transport
        INC  DX                         ;   incrementare contor daca a fost CF
et2: OR   AX, AX                        ;   testeaza AX
        JNZ  et1                        ; pana cand AX=0
        MOV  AH, 4CH
        INT  21H
COD   ENDS
        END  START
```

d) Varianta prin bitul Carry, ciclu cu test initial, numar *variabil* de iteratii:

```
DATA SEGMENT
        cuv     DW      0e360 H
DATA ENDS
ASSUME CS: COD, DS: DATA
COD     SEGMENT
START:
        MOV  AX, DATA
        MOV  DS, AX
        MOV   AX, cuv                   ; incarca cuv in registrul acumulator
        XOR  DX, DX                     ; resetare contor (DX = 0)
et1:    JCXZ et2                ; cat timp CX <> 0
        SHL  CX, 1              ;   deplasare la stanga cu o pozitie
        JNC  et1                        ;   salt daca nu s-a obtinut transport
        INC  DX                         ;   incrementare contor daca a fost CF
        JMP  et1                ; repeta
et2:    MOV  AH, 4CH
        INT  21H
COD     ENDS
        END  START
```

e) Varianta prin mascare si numar *fix* de iteratii:

```
DATA SEGMENT
        cuv     DW      0e360 H
DATA ENDS
ASSUME CS: COD, DS: DATA
COD     SEGMENT
START:
        MOV  AX, DATA
        MOV  DS, AX
        MOV   AX, cuv                   ; incarca cuv in registrul acumulator
        XOR  DX, DX                     ; resetare contor (DX = 0)
        MOV  BX, 1              ; masca = BX = 1
et1: TEST cuv, BX               ; repeta test (cuv AND BX)
        JZ   et2
        INC  DX                         ;  daca rezultatul testului e nenul, s-a
        ;  gasit un 1 si se incrementeaza contorul
et2: SHL  BX, 1                 ;  deplasarea mastii cu o pozitie la stg.
        JNC  et1                        ; salt daca a fost transport
        MOV  AH, 4CH
        INT  21H
COD     ENDS
        END  START
```

5. Desfasurarea lucrarii

1. Se editeaza *programul de ordonare crescatoare a 2 valori dintr-un sir de numere naturale* (vezi $4.) intr-un fisier cu numele *AP41.ASM.*

Se parcurg etapele 1.a) ... 1.f) de la lucrarea nr. 2 pentru acest program.

2. Se concepe si editeaza un *program de ordonare crescatoare a N (=4) valori dintr-un sir de numere naturale* intr-un fisier cu numele *AP42.ASM.*

Se parcurg etapele 1.a) ... 1.f) de la lucrarea nr. 2 pentru acest program.

3. Se concepe si editeaza un *program de ordonare crescatoare a N (=4) valori dintr-un sir de numere intregi* intr-un fisier cu numele *AP43.ASM.*

4. Se concepe si editeaza un *program de deplasare in dubla precizie a continutului unei variabile v cu N (=6) pozitii catre dreapta* intr-un fisier cu numele *AP44.ASM (vezi si lucrarea nr. 3, $4.1.).*

6. Teme si exercitii

1. Sa se scrie un *program care sa calculeze suma in dubla precizie a doua siruri de dublu-cuvinte din memorie.*

2. Sa se scrie un *program care sa citeasca o cifra N de la tastatura, sa calculeze N! (N factorial) si sa afiseze rezultatul pe ecran.*

3. Sa se scrie un *program care sa citeasca de la tastatura valorile x, y si z, sa calculeze expresia: E = x! + 0,25y + 5z si sa afiseze rezultatul pe ecran.*

7. Intrebari

1. Care sunt instructiunile de salt conditionat pentru valori de orice tip ?
2. Care sunt instructiunile de salt conditionat pentru valori naturale ?
3. Care sunt instructiunile de salt conditionat pentru valori intregi ?
4. Cum se poate realiza utilizand instructiuni de salt o structura de program de forma:

daca *conditie* atunci
  secventa1
altfel
  secventa2