**LUCRAREA DE LABORATOR 3**

**PROGRAME LINIARE**

**1. SCOPUL LUCRARII**

Lucrarea prezinta instructiunile pentru transferuri de date, instructiuni in aritmetica binara si in aritmetica BCD.

**2. NOTIUNI TEORETICE**

Instructiunile de transfer permit copierea unui octet sau cuvant de la sursa la destinatie. Destinatia poate fi un registru, locatie de memorie sau un port de iesire, iar sursa poate fi un registru, o locatie de memorie, constante sau port de intrare. Ca regula generala destinatia si sursa nu pot fi ambele locatii de memorie. In specificarea sursei si destinatiei se vor folosi notatiile:

* segment: offset pentru adrese fizice;
* [x] paranteze patrate pentru a desemna “continutul lui x”.

OBS. Instructiunea de transfer nu modifica indicatori de conditie (flagurile) cu exceptia instructiunilor SAHF si POPF.

**2.1. Instructiuni de transfer.**

**2.1.1. Instrucţiunea MOV (Move Data).**

Forma generală a instrucţiunii MOV este:

*mov dest, sursa ;*  [dest] ¬ [sursa]

realizează transferul informaţiei de la adresa efectiva data de *sursa* la *dest.*

Restricţii:

* Este necesar ca ambii operanzi să fie de aceiași mărime;
* Ambii operanzi nu pot fi locaţii de memorie (este necesară utilizarea unui registru);
* Registrele IP, EIP, sau RIP nu pot fi ca operanzi destinaţie.

Structura instrucţiunii **MOV** poate fi următoarea:

MOV *reg,reg*

MOV *mem,reg*

MOV *reg,mem*

MOV *mem,imm*

MOV *reg,imm*

Exemple:

.data

var1 WORD ?

var2 WORD ?

.code

mov ax,var1

mov var2,ax

.data

oneByte BYTE 78h

oneWord WORD 1234h

oneDword DWORD 12345678h

.code

mov eax,0 ; EAX = 00000000h

mov al,oneByte ; EAX = 00000078h

mov ax,oneWord ; EAX = 00001234h

mov eax,oneDword ; EAX = 12345678h

mov ax,0 ; EAX = 12340000h

.data

alfa dw 1234h

beta db 56h

.code

mov ax, alfa; transfera conţinutul adresei alfa în ax

mov bx, offset beta; transfera adresa efectiva alfa în bx

mov al, 75h; transfera 75h în al

mov cx, [100]; transfera conţinutul adresei 100 în cx

mov [di], bx; transfera conţinutul lui bx la adresa conţinuta în di

mov byte ptr alfa , [bx]; pune conţinutul octetului de la adresa

;dată de bx la adresa alfa

Instrucţiunea **MOVZX** (*move with zero-extend*)

Copie conţinutul sursei în destinaţie cu extinderea valorii întroducând zerouri. Această instrucţiune este utilizată numai petru valori fără semn. Sunt trei variante:

MOVZX *reg32,reg/mem8*

MOVZX *reg32,reg/mem16*

MOVZX *reg16,reg/mem8*

Exemple:

.data

byteVal BYTE 10001111b

.code

movzx ax,byteVal ; AX = 0000000010001111b



.data

byte1 BYTE 9Bh

word1 WORD 0A69Bh

.code

movzx eax,word1 ; EAX = 0000A69Bh

movzx edx,byte1 ; EDX = 0000009Bh

movzx cx,byte1 ; CX = 009Bh

Instrucţiunea **MOVSX** (*move with sign-extend*)

Copie conţinutul sursei în destinaţie cu extinderea valorii întroducând unităţi. Această instrucţiune este utilizată numai petru valori cu semn. Sunt trei variante:

MOVZX *reg32,reg/mem8*

MOVZX *reg32,reg/mem16*

MOVZX *reg16,reg/mem8*

Exemplu:

mov bx,0A69Bh

movsx eax,bx ; EAX = FFFFA69Bh

movsx edx,bl ; EDX = FFFFFF9Bh

movsx cx,bl ; CX = FF9Bh

Exemplu de program:

.data

val1 WORD 1000h

val2 WORD 2000h

arrayB BYTE 10h,20h,30h,40h,50h

arrayW WORD 100h,200h,300h

arrayD DWORD 10000h,20000h

.code

main PROC

; Demonstrating MOVZX instruction:

mov bx,0A69Bh

movzx eax,bx ; EAX = 0000A69Bh

movzx edx,bl ; EDX = 0000009Bh

movzx cx,bl ; CX = 009Bh

; Demonstrating MOVSX instruction:

mov bx,0A69Bh

movsx eax,bx ; EAX = FFFFA69Bh

movsx edx,bl ; EDX = FFFFFF9Bh

mov bl,7Bh

movsx cx,bl ; CX = 007Bh

; Memory-to-memory exchange:

mov ax,val1 ; AX = 1000h

xchg ax,val2 ; AX=2000h, val2=1000h

mov val1,ax ; val1 = 2000h

; Direct-Offset Addressing (byte array):

mov al,arrayB ; AL = 10h

mov al,[arrayB+1] ; AL = 20h

mov al,[arrayB+2] ; AL = 30h

; Direct-Offset Addressing (word array):

mov ax,arrayW ; AX = 100h

mov ax,[arrayW+2] ; AX = 200h

; Direct-Offset Addressing (doubleword array):

mov eax,arrayD ; EAX = 10000h

mov eax,[arrayD+4] ; EAX = 20000h

mov eax,[arrayD+4] ; EAX = 20000h

**Instrucţiunea XCHG ( Exchange Data )**

Interschimbă sursa cu destinaţia. Forma generală:

*XCHG dest, sursa*

Restricţii:

* registrele de segment nu pot apărea ca operanzi;
* cel puţin un operand trebuie sa fie un registru general.

Sunt trei variante:

XCHG reg,reg

XCHG reg,mem

XCHG mem,reg

Exemple:

xchg al, ah

xchg alfa, ax

xchg sir [si], bx

xchg eax,ebx ; exchange 32-bit regs

Interschimbarea conţinutului a doi operanzi din memorie op1 și op2 se poate face prin secvenţa de instrucţiuni:

mov reg, op1

xchg reg, op2

mov op2, reg

**Instrucţiunea XLAT (Translate)**

Forma generală:

XLAT

Instrucţiunea nu are operanzi, semnificaţia fiind:

[al]¬ ds: [[bx]+[al]]

adică se transfera în *al* conţinutul octetului de la adresa efectiva : *[bx]+[al].*

Instrucţiunea se folosește la conversia unor tipuri de date, folosind tabele de conversie, adresa acestor tabele se introduce în *bx*, iar în *al* se introduce poziţia elementului din tabel. De exemplu: conversia unei valori numerice cuprinsă intre 0 și 15 în cifrele hexazecimale corespunzătoare, se poate face prin:

tabel BYTE '0123456789abcdef''

. . . . . . . . . .

lea bx, tabel

mov al,11

xlat

În **al** se va depune cifra hexazecimala b.

**2.1.2 Instrucţiunile IN (Input Data) şi OUT (Output Data)**

Instrucţiunea **IN** execută o citire de 8, 16, 32 biţi de la portul de intrare. Şi invers, instrucţiunea **OUT** execută o scriere de 8, 16, 32 biţi într-un port. Sintaxa este următoarea:

IN *accumulator,port*

OUT *port,accumulator*

***Port***poate fi o constantă cu plaja 0 - FFh, sau poate fi o valoare încărcată în registrul DX cu plaja 0 - FFFFh. Ca ***Accumulator*** va fi registrul AL pentru transferuri pe 8-biţi, AX pentru transferuri pe 16-biţi şi EAX pentru transferuri pe 32-biţi.

Exemple:

in al,3Ch ; input byte from port 3Ch

out 3Ch,al ; output byte to port 3Ch

mov dx, portNumber ; DX can contain a port number

in ax,dx ; input word from port named in DX

out dx,ax ; output word to the same port

in eax,dx ; input doubleword from port

out dx,eax ; output doubleword to same port

**Instrucțiunea LEA (Load Effective Address)**

Are ca efect încărcarea adresei efective (offsetul) într-un registru general.

Forma generală:

**LEA** reg, sursa

unde:

* *sursa* - este un operand aflat în memorie, specificat printr-un mod de adresare ;
* *reg* - este un registru general.

Exemplu:

lea bx, alfa

lea si, alfa [bx][si]

Acelaşi efect se obţine folosind operandul OFFSET în Instrucţiunea MOV:

mov bx, offset alfa

mov si, offset alfa [bx][si]

**Instrucţiunea LDS/ LES (Load Data Segment/ Load Extra Segment)**

Forma generală:

LDS reg, sursa

unde:

* *reg* -este un registru general de 16 biti;
* *sursa* -este un operand de tip double – word aflat în memorie, care conţine o adresă completă de 32 biţi.

Are ca efect transferul unei adrese complete în perechea de registre *ds* și *reg* specificat în instrucţiune, adică:

[reg] ¬ [[sursa]]

[ds] ¬ [[sursa] + 2]

Exemplu:

alfa byte 25

adr\_alfa dword alfa

. . . . . . .. . .. . . .

lds si, adr\_alfa ; în registru *si* se transferă offset-ul,

;iar în *ds* adresa de segment a celulei alfa

mov byte ptr [si], 75

. . . . .. . . .. . . . . . . .

**Instrucţiunea LAHF (Load AH with FLAGS)**

Instrucţiunea încarcă în registrul AH octetul dat de partea cea mai puţin semnificativa a registrului FLAGS, ce conţine indicatorii. Instrucţiunea nu are operanzi.

AH ¬ FLAGS 0¸7

Ex.

.data

saveflags BYTE ?

.code

lahf ; load flags into AH

mov saveflags,ah ; save them in a variable

**Instrucţiunea SAHF (Store AH into FLAGS)**

Instrucţiunea încarcă în registrul FLAGS (EFLAGS or RFLAGS), în octetul cel mai puţin semnificativ conţinutul registrului AH, adică:

FLAGS ¬ [AH] 0¸7

Instrucţiunea nu are operanzi.

Ex.

mov ah,saveflags ; load saved flags into AH

sahf ; copy into Flags register

**2.1.3 Instrucţiunea INC (Increment)**

Forma generală:

**INC**  dest ; [dest] ¬ [dest] + 1

unde *dest* este un registru general, un operand din memorie. Semnificația fiind operandul *dest* este incrementat cu unu. Indicatorii afectaţi sunt AF, PF, SF, ZF, OF.

Exemple:

inc alfa

inc bl

inc eax

inc rbx

inc word ptr [bx] [si]

**Instrucţiunea DEC (decrement)**

Forma generală:

**DEC** dest ; [dest] ¬ [dest] – 1

unde *dest* are aceeaşi semnificaţie ca *dest* de la Instrucţiunea INC. Aceeaşi indicatori ca la INC sunt afectaţi.

**Instrucţiunea NEG (Negate)**

Forma generală:

**NEG** dest : [dest] ¬ 0 – [dest] schimbare de semn

unde *dest* este un operand pe 8, 16, 32 sau 64 biţi ce poate fi un registru general sau o locaţie de memorie. Instrucţiunea afectează indicatorii AF, CF, PF, SF, OF și ZF.

Exemplu:

alfa byte 75

. . . . . . . . . . . .

mov al, alfa

neg al

mov alfa, al ; la adresa alfa avem - 75

sau

neg alfa

**Instrucţiunea CMP (Compare)**

Forma generală:

**CMP** dest, sursa ; [dest] – [sursa]

Instrucţiunea realizează o operaţie de scădere intre cei doi operanzi, fără a modifica operandul *dest* sau *sursa* cu poziţionarea indicatorilor de condiţie. Indicatorii afectaţi sunt: AF, CF, PF, SF, ZF și OF.

Aceasta instrucţiune se folosește împreuna cu instrucţiunea de salt condiţionat.

Când comparăm doi operanzi fără semn indicatoarele Zero şi Carry indică următoarea relaţie între operanzi:



Exemple:

cmp ax, alfa[bx][si]

cmp [si], 0

***2.2 Instrucţiuni de salt și ciclare***

***a)Instrucţiunea de salt necondiţionat JMP***

Forma generala :

***JMP operand***

unde, operand este adresa de salt necondiţionat. Există următoarele tipuri de instrucţiuni JMP:

* de tip SHORT - când operandul specifica o adresă în domeniul -128÷ +127 față de (IP) actualizat
* de tip NEAR - operandul specifică o adresă din acelaşi segment de cod;
* de tip FAR - operandul specifică o adresă din alt segment de cod.

***b)Instrucţiuni de salt condiţionat***

Aceste instrucţiuni implementează salturile condiţionate de indicatorii de condiţie.

Forma generala:

***Jcond operand***

unde:

- *cond* este condiţia de salt și este reprezentată de una sau două litere (vezi tabelul de mai jos);

- *operand* este un offset cuprins între -128 si 128.

Dacă condiţia este îndeplinită are loc saltul la adresa dată de operand, dacă nu - se continuă în secvenţă.

Se observă că există 2 categorii de instrucţiuni pentru ‘mai mic’ si ‘mai mare’, cele care conţin cuvintele ‘above’ sau ‘bellow’ şi cele care conţin cuvintele ‘less’ sau ‘greater’. Primele se folosesc în situaţia comparării a două valori fără semn, iar ultimele în situaţia comparării a două valori cu semn.

Fie secvenţele de program:

mov ax,0FFFEh

mov bx, 2

cmp ax, bx

ja alfa

și

mov ax, 0FFFEh

mov bx, 2

cmp ax, bx

jg alfa

în care se compară pe cuvânt 0FFFEh şi 2.

Se observă că (AX) > (BX) dacă cele două valori se consideră reprezentate fără semn şi că (AX) < (BX) dacă cele două valori se consideră cu semn. (-2 este mai mic decat 2). Ca atare în primul caz saltul la eticheta alfa are loc, pe cand în cel de-al doilea caz nu are loc.

Fiecare mnemonică din tabel se referă la iniţialele cuvintelor următoare, ce indică condiţia în limba engleză: Above (peste, mai mare), Below (sub, mai mic), Equal (egal), Not (nu), Greater (mai mare), Less (mai mic), Carry (transport), Zero, Overflow (depăşire de capacitate), Parity (PEven - paritate pară, POdd - paritate impară), Sign (semn).

|  |  |  |
| --- | --- | --- |
| Instrucţiune  (mnemonica) | Condiţie de salt | Interpretare |
| JE, JZ | ZF = 1 | Zero, Equal |
| JL, JNGE | SF ¹ OF | Less, Not Greater or Equal |
| JLE,JNG | SF ¹ OF sau ZF = 1 | Less or Equal, Not Greater |
| JB, JNAE, JC | CF = 1 | Below, Not Above or Equal, Carry |
| JBE, JNA | CF = 1 sau ZF = 1 | Below or Equal, Not Above |
| JP, JPE | PF = 1 | Parity, Parity Even |
| JO | OF = 1 | Overflow |
| JS | SF = 1 | Sign |
| JNE, JNZ | ZF = 0 | Not Zero, Not Equal |
| JNL, JGE | SF = OF | Not Less, Greater or Equal |
| JNLE, JG | SF = OF si ZF = 0 | Not Less or Equal, Greater |
| JNB, JAE, JNC | CF = 0 | Not Below, Above or Equal, Not Carry |
| JNBE, JA | CF = 0 si ZF = 0 | Not Below or Equal, Above |
| JNP, JPO | PF = 0 | Not Parity, Parity Odd |
| JNO | OF = 0 | Not Overflow |
| JNS | SF = 0 | Not Sign |

(*Exemplu de citire: JNBE = jump if not below or equal, salt (J) dacă nu (N) e mai mic (B) sau egal (E)*).

**c) Instrucțiunea JCXZ (JUMP if CX is Zero)**

Instrucțiunea realizează salt la eticheta specificată dacă conținutul registrului CX este zero. Forma generala:

***JCXZ eticheta***

***JECXZ eticheta***

***JRCXZ eticheta***

unde *eticheta* este o eticheta aflata in domeniul -128 si 127 fata de (IP).

*Exemple:*

mov edx,0A523h

cmp edx,0A523h

jne L5 ; jump not taken

je L1 ; jump is taken

mov bx,1234h

sub bx,1234h

jne L5 ; jump not taken

je L1 ; jump is taken

mov cx,0FFFFh

inc cx

jcxz L2 ; jump is taken

xor ecx,ecx

jecxz L2 ; jump is taken

*Exemple, comparări cu semn:*

mov edx,-1

cmp edx,0

jnl L5 ; jump not taken (-1 >= 0 is false)

jnle L5 ; jump not taken (-1 > 0 is false)

jl L1 ; jump is taken (-1 < 0 is true)

mov bx,+32

cmp bx,-35

jng L5 ; jump not taken (+32 <= -35 is false)

jnge L5 ; jump not taken (+32 < -35 is false)

jge L1 ; jump is taken (+32 >= -35 is true)

mov ecx,0

cmp ecx,0

jg L5 ; jump not taken (0 > 0 is false)

jnl L1 ; jump is taken (0 >= 0 is true)

mov ecx,0

cmp ecx,0

jl L5 ; jump not taken (0 < 0 is false)

jng L1 ; jump is taken (0 <= 0 is true)

Exemplu, salt la etichetă dacă toţi biţii (2, 3 şi 7) sunt setaţi în 1:

mov al,status

and al,10001100b ; mask bits 2,3,7

cmp al,10001100b ; all bits set?

je ResetMachine ; yes: jump to label

**2.3 Aritmetica BCD**

**2.3.1 Instrucțiunea AAA (ASCII Adjust for Addition)**

Instrucțiunea nu are operanzi și execută corecția acumulatorului AX, după operații de adunare cu numere în format BCD despachetat. Semnificația este:

daca [AL0:3] > 9 sau [AF] = 1, atunci {

[AL] ¬ [AL] + 6

[AH] ¬ [AH]+1

[AF] ¬1

[CF] ¬1

[AL] ¬ [AL] AND 0FH

}

Indicatorii afectați : AF, CF, restul nedefiniți.

Exemplu:

mov ax, 408h

mov dx, 209h

add ax, dx ; [ax]=0611h

AAA ; [ax]=0707h

**2.3.2 Instrucțiunea AAS (ASCII Adjust for Subtraction)**

Instrucțiunea nu are operanzi și execută corecția acumulatorului AX, după operații de scădere cu numere in format BCD despachetat. Semnificația este:

daca [AL0:3] > 9 sau [AF] = 1, atunci {

[AL] ¬ [AL] - 6

[AH] ¬ [AH] - 1

[AF] ¬1

[CF] ¬1

[AL] ¬ [AL] AND 0FH

}

Indicatorii afectați : AF, CF, restul nedefiniți.

Exemplu:

mov ax, 408h

mov dx, 209h

sub ax, dx ; [ax]=01ffh

AAS ; [ax]=0109h

**2.3.3 Instrucțiunea DAS (Decimal Adjust for Substraction)**

Instrucțiunea nu are operanzi și execută corecția zecimala a acumulatorului AL, după operații de scădere cu numere în format BCD împachetat. Semnificația este:

daca [AL0:3] > 9 sau [AF] = 1, atunci {

[AL] ¬ [AL] - 6

[AF] ¬ 1

}

daca acum [AL4:7] > 9 sau CF = 1, atunci {

[AL] ¬ [AL] - 60H

[CF] ¬ 1

}

Indicatorii afectați : AF, CF, PF, SF, ZF. Indicatorul OF este nedefinit.

De exemplu, în urma secvenței:

MOV AL, 52H

SUB AL, 24H ; AL = 2EH

DAS ; AL = 28H

se obține în AL rezultatul corect 28H.

**2.3.5 Instrucțiunea AAM (ASCII Adjunct for Multiply)**

Instrucțiunea nu are operanzi și efectuează o corecție a acumulatorului AX, după o înmulțire pe 8 biți cu operanzi în format BCD despachetat.

Semnificația este următoarea:

[AH] ¬ [AL] / 10

[AL] ¬ [AL] MOD 10

Indicatori afectați: PF, SF, ZF, restul nedefinite.

De exemplu

mov al, 7

mov bl, 9

mul bl ; 003Fh

AAM ; 0603h

**2.3.6 Instrucțiunea AAD (ASCII Adjunct for Division)**

Instrucțiunea nu are operanzi și efectuează o corecție a acumulatorului AX, înaintea unei împărțiri a doi operanzi în format BCD despachetat.

Semnificația este următoarea:

[AL] ¬ [AH] \* 10 + [AL]

[AH] ¬ 0

Indicatori afectați: PF, SF, ZF, restul nedefinite.

De exemplu

mov ax, 305h

mov bl, 2

AAD ; [ax]=35h

div bl ; [al]=12h [ah]=1

**3 EXEMPLE REZOLVATE SI COMENTATE**

Exemplul 1. Să se calculeze expresiile aritmetice:

Se cere obţinerea fișierului executabil si rularea apoi pas cu pas. Datele se introduce de la tastatură si rezultatele se afișează pe ecran

INCLUDE Irvine32.inc

.data

mes1 byte "Introduceti valoarea X:",0

mes2 byte "Introduceti valoarea Y:",0

mes3 byte "Rezutatul este:",0

vrx dword 0

vry dword 0

rez dd 0

.code

main PROC

mov edx,OFFSET mes1

call WriteString ; afisarea mes1

call ReadDec ; introducerea de la tastatura

mov vrx,eax ; salvarea valorii in variabila *vrx*

mov edx,OFFSET mes2

call WriteString ; afisarea mes2

call ReadDec ; introducerea de la tastatura

mov vry,eax ; salvarea valorii in variabila *vry*

;controlam conditiile

xor eax,eax

mov edx,0

mov eax,vry

mov bx,2

mul bx ; calcul Y\*2

cmp vrx,eax ;compararea X cu 2Y

jb con1 ; salt la con1, daca X<2Y

mov eax,vry ;realizam expresia 2Y-60

mov bx,2

mul bx

sub eax,60

mov rez,eax

jmp ex ; salt neconditionat la ex

con1: mov eax,vrx ; realizam expresia X/8+32-Y

mov bx,8

div bx

add eax,32

sub eax,vry

mov rez,eax

ex: mov edx,OFFSET mes3

call WriteString ; afisarea mesajului mes3

call WriteInt ; afisarea rezultatului cu semn

call Crlf ; din rand nou

exit

main ENDP

END main

Exemplul 2. Se considera un vector de trei componente, fiecare componentă având dimensiunea de un cuvânt. Sa se permute circular elementele vectorului cu o poziție la stânga:

.DATA

vect WORD 100h,200h,300h

.CODE

main PROC

mov dx,vect ; Salvare in *dx* a primei

; componente a vectorului

mov ax,vect+2; Transfera a doua

mov vect,ax ; componenta in prima

mov ax,vect+4; Transfera a treia

mov vect+2,ax; componenta in a doua

mov vect+4,dx; Transfera prima

; componenta in ultima

exit

main ENDP

END main

Exemplul 3. Se consideră un vector de trei componente cuvinte. Să se introducă valoarea *55h* în octeții superiori ai elementelor vectorului.

.DATA

val EQU 55h

vect DW 111h,222h,333h

.CODE

mov al,val ; Se incarca valoarea

; dorita in registrul *al*

mov BYTE PTR vect[1],al ; Se transfera

mov BYTE PTR vect[3],al ; valoarea in

mov BYTE PTR vect[5],al ; octetii

; superiori ai

; elementelor vectorului

Exemplul 4. Să se genereze *2* numere a câte *4* cifre zecimale reprezentate în format BCD împachetat (adică fiecare grup de doua cifre zecimale este memorat într-un octet) şi să se scrie o secvență de program pentru calculul sumei şi diferenței lor.

.data

bcd1 DB 34h,18h; 1834 in format BCD

; impachetat

bcd2 DB 89h,27h; 2789 in format BCD

; impachetat

sum DB 2 DUP(?)

dif DB 2 DUP(?)

.code

mov al,bcd1

add al,bcd2

daa

mov sum,al

mov al,bcd1+1

adc al,bcd2+1

daa

mov sum+1,al

mov al,bcd1

sub al,bcd2

das

mov dif,al

mov al,bcd1+1

sbb al,bcd2+1

das

mov sum+1,al

Exemplul 5. Să se scrie un program pentru adunarea şi scăderea a doua numere a câte două cifre zecimale reprezentate în format BCD neîmpachetat (adică fiecare cifra zecimală se memorează în tetrada inferioara a unui octet):

.data

upk1 DB 01h,02h; 21 in format BCD

; neimpachetat

upk2 DB 03h,06h; 63 in format BCD

; neimpachetat

sum DB 2 DUP(?)

dif DB 2 DUP(?)

.code

mov al,upk1

add al,upk2

aaa

mov sum,al

mov al,upk1+1

adc al,upk2+1

aaa

mov dif+1,al

mov al,upk1

sub al,upk2

aas

mov dif,al

mov al,upk1+1

sbb al,upk2+1

aas

mov dif+1,al

**4 DESFASURAREA LUCRARII DE LABORATOR**

Rulați exemplele pas cu pas, programele ce folosesc instrucțiuni aritmetice pentru a urmări evoluția rezultatelor parțiale.

**5 PROBLEME PROPUSE**

Conform variantei elaboraţi 2 variante de program :

1. cu introducerea datelor de la tastatură şi afişarea rezultatelor pe ecran.
2. cu generarea datelor de intrare, utilizănd procedurile Random32, RandomRange.

|  |  |  |  |
| --- | --- | --- | --- |
| Nr | Expresia | Nr | Expresia |
| 1 | ­­­­ | 9 |  |
| 2 |  | 10 |  |
| 3 |  | 11 |  |
| 4 |  | 12 |  |
| 5 |  | 13 |  |
| 6 |  | 14 |  |
| 7 |  | 15 |  |
| 8 |  |  |  |

**6 CONTINUTUL REFERATULUI**

Referatul va conține următoarele elemente:

* enunțul problemelor de rezolvat;
* exemplele de programe comentate;
* listingul programului comentat conform variantei.