Laborator 6 - Suport teoretic

Instructiuni de comparare, salt conditionat si de ciclare. Operatii pe siruri.

Comparaţii între operanzi

Instructiunea CMP

```
CMP <opd>, <ops>
```

- Instructiunea CMP realizeaza comparatia intre valorile numerice ale celor doi operanzi prin efectuarea unei scăderi fictive opd-ops.

CMP*d,s*	comparaţie valori operanzi (nu modifică operanzii) (execuţie fictivă *d - s*)	OF,SF,ZF,AF,PF şi CF

- **CMP** scade valoarea operandului sursă din operandul destinaţie, dar spre deosebire de instrucţiunea **SUB**, rezultatul nu este reţinut, el neafectând nici una din valorile iniţiale ale operanzilor. Efectul acestei instrucţiuni constă numai în modificarea valorii unor flaguri în conformitate cu efectuarea operaţiei opd-ops. Instrucţiunea **CMP** este cel mai des folosită în combinaţie cu instrucţiuni de salt condiţionat.
- **Deşi numele ei este CMP este important de subliniat ca în realitate aceasta instructiune NU COMPARA nimic, nestabilind nici un criteriu de comparaţie şi neluând de fapt nici o decizie, ci ea doar PREGATESTE decizia corespunzător cu flagurile setate, comparaţia efectivă si decizia corespunzatoare fiind luata concret de instrucţiunea de salt conditionat care va fi folosită ulterior instructiunii CMP ! Dacă nu folosim ulterior nici o instructiune decizională, CMP nu are nici un rol concret în vreo comparatie, ea reprezentând doar o simpla scadere fictiva cu rol de afectare a flagurilor şi nu îşi va merita în nici un caz numele de CMP (compare).**
- Operatorul destinatie poate sa fie registru sau variabila in memorie.
- Operandul sursa poate sa fie registru, variabila in memorie sau constanta.
- Ambii operanzi ai instructiunii CMP trebuie sa fie de aceasi dimensiune.

Exemplul 1:

```
cmp eax, ebx ; "compara" valorile stocate in cei doi registri (scadere fictiva eax-ebx)
jle done ;în funcţie de instrucţiunea de salt condiţionat utilizată (aici JLE) se stabileste criteriul de comparare.
```

```
;In acest caz: daca continutul din EAX in interpretarea cu semn este mai mic
sau egal cu continutul din EBX atunci JUMP la eticheta Done,
;Altfel continua cu urmatoarea  instructiune (flagul testat aici este ZF).
done:
;instructiuni care urmează etichetei done
```

Exemplul 2:

```
mov al,200 ; AL = C8h
mov bl,100
cmp al, bl ; se realizeaza scaderea fictiva al-bl si se seteaza
; flagurile in mod corespunzator acesteia (in acest caz vom avea SF=0, OF=1,
CF=0 şi ZF=0)
JB et2 ;instructiunea de salt conditionat stabileste criteriul de comparare,
in acest caz Jump if Below – comparatie pentru numere fara semn (este 200
BELOW 100 ?) si se testeaza continutul lui CF: dacă CF=1 saltul se va efectua,
dacă CF=0 saltul NU se va efectua.
;Cum CF=0 în cazul nostru, saltul NU se va efectua.
;.............. ;set de instructiuni care urmează
et2:
;.............. ;set de instructiuni care urmează etichetei
```

Exemplul 3:

```
mov al,-56 ; AL = C8h = 200 in interpretarea fara semn
mov bl,100
cmp al, bl ;se realizeaza scaderea fictiva al-bl si se seteaza flagurile in
mod corespunzator acesteia (pentru cazul nostru vom avea SF=0, OF=1, CF=0 şi
ZF=0)
JNGE et2 ;se verifica conditia JNGE - Jump if not greater or equal
;(comparaţie CU SEMN -56 fată de 100)
;concret se verifica daca este diferit continutul din SF si OF
; Avand in vedere ca în cazul nostru SF=0 şi OF=1, deci SF <> OF, condiţia
este îndeplinită (şi intr-adevăr -56 este „NOT GREATER OR ;EQUAL” fata de 100)
deci saltul la eticheta et2 se va efectua
mov dx,1
et2:
mov cx,1
```

Exemplul 4:

```
mov al,-56 ; AL = C8h = 200 in interpretarea fara semn
mov bl,100
cmp al, bl ;se realizeaza scaderea fictiva al-bl si se seteaza  flagurile in
mod corespunzator acesteia (pentru cazul nostru vom avea SF=0, OF=1, CF=0 şi
ZF=0)
JNBE et2 ;se verifica conditia JNBE - Jump if not below or equal
;(comparaţie FARA SEMN  200 fată de 100)
;concret se verifica daca CF=0 şi ZF=0
;Avand in vedere ca în cazul nostru CF=0 şi ZF=0, condiţia este îndeplinită
(şi intr-adevăr 200 este „NOT BELOW OR EQUAL” ;fata de 100) deci saltul la
eticheta et2 se va efectua
mov dx,1
et2:
mov cx,1
```

Instructiunea TEST

```
TEST <opd>, <ops>
```

- Instructiunea TEST realizeaza operatia logica SI intre cei doi operanzi (executie fictiva ops **AND** opd) fara a salva rezultatul operatiei in vreunul dintre cei doi operanzi.
- Ambii operanzi ai instructiunii TEST trebuie sa fie de aceasi dimensiune.
- Singurul efect al unei instructiuni TEST este modificarea continutului flagurilor specificate in tabelul de mai sus corespunzator cu rezultatul operatiei AND efectuate.

TEST *d,s*	execuţie fictivă *d* **AND** *s*	OF = 0, CF = 0 SF,ZF,PF - modificaţi AF - nedefinit

Exemplu 1:

```
ambiiOPeranziZero: ;set de instructiuni care compun eticheta....
test ECX, ECX ; set ZF to 1 if ECX == 0
je ambiiOPeranziZero ; în funcţie de instrucţiunea de salt condiţionat utilizată
; (aici JE) se stabileste criteriul de comparare.
;In acest caz: jump la eticheta if ZF = 1
; alternativ se putea folosi aici varianta jz AmbiiOPeranziZero, aceste doua instructiuni de salt conditionat (JE şi JZ) fiind similare in ceea ce priveste conditia testata (true if ZF=1)
```

Exemplu 2:

```
mov AH,[v]
test AH,0F2h
js et2 ; în funcţie de instrucţiunea de salt condiţionat utilizată (aici JS) se stabileste criteriul de comparare.
; In acest caz: daca rezultatul operatiei AH AND 0F2h este un numar strict negativ in interpretarea cu semn (adica daca bitul de semn al rezultatului este 1) atunci Jump ;la eticheta et2 (flagul testat este SF).
et2: ;set de instructiuni care compun eticheta....
```

- Similar cu situatia de la instructiunea CMP, avem de făcut şi aici următoarea observaţie:
- **Deşi numele ei este TEST este important de subliniat ca în realitate aceasta instructiune NU TESTEAZA nimic, nestabilind nici un criteriu de testare şi neluând de fapt nici o decizie, ci ea doar PREGATESTE decizia corespunzător cu flagurile setate, criteriul de testare, testarea efectivă si decizia corespunzatoare fiind luata concret de instrucţiunea de salt conditionat care va fi folosită ulterior instructiunii TEST ! Dacă nu folosim ulterior nici o instructiune decizională, TEST nu are nici un rol concret în vreo testare, ea reprezentând doar o simpla operatie SI bit cu bit cu rol de afectare a flagurilor şi nu îşi va merita în nici un caz numele de TEST (testare a unei conditii).**

Salturi condiţionate de flag-uri

- Cand se compara doua numere cu semn se folosesc termenii "less than" (mai mic decat) si "greater than" (mai mare decat), iar cand se compara doua numere fara semn se folosesc termenii "below" (inferior, sub) si respectiv "above" (superior, deasupra, peste).
- Redam mai jos o parte din documentatia INTEL, referitoare la aceste instructiuni de salt. (🔗 Combined Volume Set of Intel® 64 and IA-32 Architectures Software Developer's Manuals, incepand de la pag. 1058)

JMP – Unconditional Jump

Mnemonic	Description
JMP rel8	Jump short, relative, displacement relative to next instruction.
JMP rel16	Jump near, relative, displacement relative to next instruction.
JMP rel32	Jump near, relative, displacement relative to next instruction.
JMP r/m16	Jump near, absolute indirect, address given in r/m16.
JMP r/m32	Jump near, absolute indirect, address given in r/m32.
JMP ptr16:16	Jump far, absolute, address given in operand.
JMP ptr16:32	Jump far, absolute, address given in operand.
JMP m16:16	Jump far, absolute indirect, address given in m16:16.
JMP m16:32	Jump far, absolute indirect, address given in m16:32.

- Transfers program control to a different point in the instruction stream without recording return information.
- The destination (target) operand specifies the address of the instruction being jumped to.
- This operand can be an immediate value, a general-purpose register, or a memory location.

This instruction can be used to execute four different types of jumps:

- **Near jump:** A jump to an instruction within the current code segment (the segment currently pointed to by the CS register), sometimes referred to as an intrasegment jump.
- **Short jump:** A near jump where the jump range is limited to -128 to +127 from the current EIP value.
- **Far jump:** A jump to an instruction located in a different segment than the current code segment but at the same privilege level, sometimes referred to as an intersegment jump.

Jcc — Jump if Condition Is Met

- The conditions for each Jcc mnemonic are given in the "{description}" column of the table on the preceding page. The terms "less" and "greater" are used for comparisons of signed integers and the terms "above" and "below" are used for unsigned integers.

Mnemonic	Description
JA rel8	Jump short if above (CF=0 and ZF=0).
JAE rel8	Jump short if above or equal (CF=0).
JB rel8	Jump short if below (CF=1).
JBE rel8	Jump short if below or equal (CF=1 or ZF=1).
JC rel8	Jump short if carry (CF=1).
JCXZ rel8	Jump short if CX register is 0.
JECXZ rel8	Jump short if ECX register is 0.
JE rel8	Jump short if equal (ZF=1).
JG rel8	Jump short if greater (ZF=0 and SF=OF).
JGE rel8	Jump short if greater or equal (SF=OF).
JL rel8	Jump short if less (SF<>OF).
JLE rel8	Jump short if less or equal (ZF=1 or SF<>OF).
JNA rel8	Jump short if not above (CF=1 or ZF=1).
JNAE rel8	Jump short if not above or equal (CF=1).
JNB rel8	Jump short if not below (CF=0).

JNBE rel8	Jump short if not below or equal (CF=0 and ZF=0).
JNC rel8	Jump short if not carry (CF=0).
JNE rel8	Jump short if not equal (ZF=0).
JNG rel8	Jump short if not greater (ZF=1 or SF<>OF).
JNGE rel8	Jump short if not greater or equal (SF<>OF).
JNL rel8	Jump short if not less (SF=OF).
JNLE rel8	Jump short if not less or equal (ZF=0 and SF=OF).
JNO rel8	Jump short if not overflow (OF=0).
JNP rel8	Jump short if not parity (PF=0).
JNS rel8	Jump short if not sign (SF=0).
JNZ rel8	Jump short if not zero (ZF=0).
JO rel8	Jump short if overflow (OF=1).
JP rel8	Jump short if parity (PF=1).
JPE rel8	Jump short if parity even (PF=1).
JPO rel8	Jump short if parity odd (PF=0).
JS rel8	Jump short if sign (SF=1).
JZ rel8	Jump short if zero (ZF = 1).
JA rel16/32	Jump near if above (CF=0 and ZF=0).

JAE rel16/32	Jump near if above or equal (CF=0).
JB rel16/32	Jump near if below (CF=1).
JBE rel16/32	Jump near if below or equal (CF=1 or ZF=1).
JC rel16/32	Jump near if carry (CF=1).
JE rel16/32	Jump near if equal (ZF=1).
JZ rel16/32	Jump near if 0 (ZF=1).
JG rel16/32	Jump near if greater (ZF=0 and SF=OF).
JGE rel16/32	Jump near if greater or equal (SF=OF).
JL rel16/32	Jump near if less (SF<>OF).
JLE rel16/32	Jump near if less or equal (ZF=1 or SF<>OF).
JNA rel16/32	Jump near if not above (CF=1 or ZF=1).
JNAE rel16/32	Jump near if not above or equal (CF=1).
JNB rel16/32	Jump near if not below (CF=0).
JNBE rel16/32	Jump near if not below or equal (CF=0 and ZF=0).
JNC rel16/32	Jump near if not carry (CF=0).
JNE rel16/32	Jump near if not equal (ZF=0).
JNG rel16/32	Jump near if not greater (ZF=1 or SF<>OF).
JNGE rel16/32	Jump near if not greater or equal (SF<>OF).

JNL rel16/32	Jump near if not less (SF=OF).
JNLE rel16/32	Jump near if not less or equal (ZF=0 and SF=OF).
JNO rel16/32	Jump near if not overflow (OF=0).
JNP rel16/32	Jump near if not parity (PF=0).
JNS rel16/32	Jump near if not sign (SF=0).
JNZ rel16/32	Jump near if not zero (ZF=0).
JO rel16/32	Jump near if overflow (OF=1).
JP rel16/32	Jump near if parity (PF=1).
JPE rel16/32	Jump near if parity even (PF=1).
JPO rel16/32	Jump near if parity odd (PF=0).
JS rel16/32	Jump near if sign (SF=1).
JZ rel16/32	Jump near if 0 (ZF=1).

- These jumps checks the state of one or more of the status flags in the EFLAGS register (CF, OF, PF, SF, and ZF) and, if the flags are in the specified state (condition), performs a jump to the target instruction specified by the destination operand.
- A condition code (cc) is associated with each instruction to indicate the condition being tested for. If the condition is not satisfied, the jump is not performed and execution continues with the instruction following the Jcc instruction.
- The target instruction is specified with a relative offset (a signed offset relative to the current value of the instruction pointer in the EIP register).
- **A relative offset (rel8, rel16, or rel32) is generally specified as a label in assembly code, but at the machine code level, it is encoded as a signed, 8-bit or 32-bit immediate value, which is added to the instruction pointer.**
- Instruction coding is most efficient for offsets of -128 to +127. If the operand-size attribute is 16, the upper two bytes of the EIP register are cleared, resulting in a maximum instruction pointer size of 16 bits.
- The Jcc instruction does not support far jumps (jumps to other code segments). When the target for the conditional jump is in a different segment, use the opposite condition from the condition being

tested for the Jcc instruction, and then access the target with an unconditional far jump (JMP instruction) to the other segment.

- The **JECXZ** and **JCXZ** instructions differ from the other Jcc instructions because they do not check the status flags. Instead they check the contents of the ECX and CX registers, respectively, for 0. Either the CX or ECX register is chosen according to the address-size attribute.
- These instructions are useful at the beginning of a conditional loop that terminates with a conditional loop instruction (such as LOOPNE). They prevent entering the loop when the ECX or CX register is equal to 0, which would cause the loop to execute 232 or 64K times, respectively, instead of zero times.

- Sumarizand, prin analiza tabelului de mai sus, se observă că unele instrucţiuni testează exact aceeaşi condiţie, ele fiind astfel similare ca efect. Faptul că o aceeaşi instrucţiune apare sub mai multe forme sintactice echivalente provine din posibilitatea de a exprima o aceeaşi situaţie sub mai multe formulări echivalente. De exemplu condiţia op1 „mai mic sau egal" (JLE) decât op2 se poate exprima şi sub forma op1 „NU este mai mare decât" (JNG) op2. Similar, condiţia JB (Jump if below – tradus prin „este inferior") se poate exprima şi sub forma „NU este superior sau egal" (JNAE) etc.

In tabelul următor aveţi grupate toate instructiunile echivalente ca efect, echivalenţa lor fiind dată tocmai pe baza condiţiei testate. Ca urmare, este totuna de exemplu dacă folosiţi într-o comparaţie JAE, JNB sau JNC, efectul lor fiind identic: „salt dacă CF=0".

MNEMONICA	SEMNIFICAŢIE (salt dacă..<<relaţie>>)	Condiţia verificată
JB **JNAE** **JC**	este inferior nu este superior sau egal există transport	CF=1
JAE **JNB** **JNC**	este superior sau egal nu este inferior nu există transport	CF=0
JBE **JNA**	este inferior sau egal nu este superior	CF=1 sau ZF=1
JA **JNBE**	este superior nu este inferior sau egal	CF=0 şi ZF=0
JE **JZ**	este egal este zero	ZF=1
JNE **JNZ**	nu este egal nu este zero	ZF=0
JL **JNGE**	este mai mic decât nu este mai mare sau egal	SF≠OF
JGE	este mai mare sau egal	SF=OF

JNL	nu este mai mic decât	
JLE **JNG**	este mai mic sau egal nu este mai mare decât	ZF=1 sau SF≠OF
JG **JNLE**	este mai mare decât nu este mai mic sau egal	ZF=0 şi SF=OF
JP **JPE**	are paritate paritatea este pară	PF=1
JNP **JPO**	nu are paritate paritatea este impară	PF=0
JS	are semn negativ	SF=1
JNS	nu are semn negativ	SF=0
JO	există depăşire	OF=1
JNO	nu există depăşire	OF=0

- Pentru a facilita alegerea corectă de către programator a variantelor de salt condiţionat în raport cu rezultatul unei comparaţii (adică, dacă programatorul doreşte interpretarea rezultatului comparaţiei cu semn sau fără semn) dăm următorul tabel:

Relaţia între operanzi ce se doreşte a fi testată	Comparaţie cu semn	Comparaţie fără semn
d = s	JE	JE
d ≠ s	JNE	JNE
d > s	JG	JA
d < s	JL	JB
d ≥ s	JGE	JAE
d ≤ s	JLE	JBE

- Studiul acestor tabele reiterează afirmaţia noastră anterioară: nu instrucţiunea CMP este cea care face distincţie între o comparaţie cu semn şi una fără semn! Rolul de a interpreta în mod diferit (cu semn sau fără semn) rezultatul final al comparaţiei revine NUMAI instrucţiunilor de salt condiţionat specificate ULTERIOR comparaţiei efectuate.

- Tabelul de mai sus îl considerăm foarte util pentru interpretarea rezultatelor instrucţiunilor aritmetice în general. Fără a mai efectua o instrucţiune CMP, considerând d rezultatul ultimei instrucţiuni aritmetice executate şi punând s=0, tabelul rămâne valabil.

Instructiuni de ciclare

- Procesoarele x86 sunt prevăzute cu instrucţiuni speciale pentru realizarea ciclării. Ele sunt: LOOP, LOOPE, LOOPNE şi JECXZ.
- Sintaxa lor este:

```
[instructiune] eticheta
```

- Instrucţiunea LOOP comandă reluarea execuţiei blocului de instrucţiuni ce începe la etichetă, atâta timp cât valoarea din registrul ECX este diferită de 0. Se efectuează întâi decrementarea registrului ECX şi apoi se face testul şi eventual saltul. Saltul este de această dată în mod obligatoriu "scurt" (max. 127 octeți - atenție deci la "distanța" dintre LOOP şi etichetă!).

Exemplu:

```
mov ecx, 5
start_loop:
; the code here would be executed 5 times
loop start_loop
```

- În cazul în care condiţiile de terminare a ciclului sunt mai complexe se pot folosi instrucţiunile LOOPE şi LOOPNE.
- Instrucţiunea LOOPE (LOOP while Equal) diferă faţă de LOOP prin condiţia de terminare, ciclul terminându-se fie dacă ECX=0, fie dacă ZF=0. În cazul instrucţiunii LOOPNE (LOOP while Not Equal) ciclul se va termina fie dacă ECX=0, fie dacă ZF=1. Chiar dacă ieşirea din ciclu se face pe baza valorii din ZF, decrementarea lui ECX are oricum loc.
- LOOPE mai este cunoscută şi sub numele de LOOPZ iar LOOPNE mai este cunoscută şi sub numele de LOOPNZ. Aceste instrucţiuni se folosesc de obicei precedate de o instrucţiune CMP sau SUB
- Să presupunem de exemplu că dorim să reţinem într-un vector întregii pe 32 biti cititi de la tastatură atâta timp cât numărul acestora nu depăşeşte 128 şi valorile introduse sunt valide. Acest lucru se poate obţine prin secvenţa:

```
segment data use32 class=data
        vector resd 128 ; spatiu de stocare pentru 128 întregi
        fmt db "%d", 0 ; vom citi cu scanf ("%d", ...)
segment code use32  class=data
start:
        mov ebx, vector ; ebx indică elementul curent (primul)
        mov ecx, 128 ; permitem maximum 128 elemente
.bucla:
        push ecx ; salvam ECX (funcţiile externe au
        push ebx ; drept să-l altereze)
        push fmt ; cei doi parametri sunt în stivă
        call [scanf] ; apel scanf cu "%d" si ebx
        add esp, 2 * 4 ; eliberarea parametrilor din stivă
```

```
        pop ecx ; restaurare valoarea lui ECX
        add ebx, 4 ; avansăm cu un DWORD
        cmp eax, 0 ; eax (rezultatul lui scanf) este zero?
loopnz .bucla ; dacă nu, predă controlul instructiunii
; de după  .bucla
```

- Instructiunile JECXZ si JCXZ sunt instructiuni de jump conditional dar difera de setul standard al acestor instructiuni prin faptul ca acestea NU verifica statusul flagurilor. JECXZ si JCXZ verifica numai daca continutul din ECX sau CX are valoarea 0. Aceste instructiuni pot fi utilizate la inceputul unei bucle care se termina cu un loop conditional pentru a preveni intrarea in bucla cand ECX sau CX au valoarea 0. Daca se va intra in bucla cu CX=0, avand in vedere faptul ca "se efectuează întâi decrementarea registrului ECX şi apoi se face testul şi eventual saltul" acest aspect va determina executia unei bucle de 2^32 ori sau 2^16 ori, in loc de 0 ori cum ar fi normal pe baza valorii CX=0..

Exemplu:

```
mov ecx, numar ; numar de iteratii
JECXZ endFor   ;skip loop if numar=0
forIndex
; instructiuni
Loop forIndex  ; repeat
endFor
```

Echivalent cu:

```
mov ECX, numar
cmp ECX,0
JZ endFor
forIndex
; instructiuni
Loop forIndex  ; repeat
endFor
```

- Este important să precizăm aici faptul că nici una dintre instrucţiunile de ciclare prezentate nu afectează flag-urile şi de asemenea că

`loop Bucla`	`dec ecx` `jnz Bucla`

- deşi semantic echivalente, nu au exact acelaşi efect, deoarece spre deosebire de LOOP, instrucţiunea DEC afectează indicatorii OF, ZF, SF şi PF.

Laborator 6 - Exemple

Instructiuni de comparare, salt conditionat si de ciclare. Operatii pe siruri.

Exemplu

```
;Se da un sir de caractere format din litere mici.
;Sa se transforme acest sir in sirul literelor mari corespunzator.
bits 32
global start
extern exit,printf ; tell nasm that exit exists even if we won't be defining it
import exit msvcrt.dll ; exit is a function that ends the calling process. It is defined in msvcrt.dll
import printf msvcrt.dll
; msvcrt.dll contains exit, printf and all the other important C-runtime specific functions
; our data is declared here (the variables needed by our program)
segment data use32 class=data
        s db 'a', 'b', 'c', 'm','n' ; declararea sirului initial s
        l equ $-s ; stabilirea lungimea sirului initial l
        d times l db 0 ; rezervarea unui spatiu de dimensiune l pentru sirul destinatie d si initializarea acestuia
; format  db "%s", 0 ;definire format pentru afisare, daca vrem sa afisam sirul rezultat
segment code use32 class=code
start:
        mov ecx, l ;punem lungimea in ECX pentru a putea realiza bucla loop de ecx ori
        mov esi, 0
        jecxz Sfarsit
        Repeta:
                mov al, [s+esi]
                mov bl, 'a'-'A' ; pentru a obtine litera mare corespunzatoare literei mici, vom scadea din codul ASCII
                ; al literei mici diferenta dintre 'a'-'A'
                sub al, bl
                mov [d+esi], al
                inc esi
        loop Repeta
        Sfarsit:;terminarea programului
        ;Daca dorim si afisarea sirului d, avem urmatoarele:
        ;push dword d ; punem parametrii pe stiva de la dreapta la stanga
        ;push dword format
        ;call [printf] ;apelam functia printf
        ;add esp, 4 * 2 ; eliberam parametrii de pe stiva
        ; exit(0)
        push dword 0 ; push the parameter for exit onto the stack
        call [exit] ; call exit to terminate the program
```

Laborator 6 - Probleme propuse

Instructiuni de comparare, salt conditionat si de ciclare. Operatii pe siruri.

Exerciții

1. Se dau doua siruri de octeti S1 si S2. Sa se construiasca sirul D prin concatenarea elementelor din sirul S1 1uate de la stanga spre dreapta si a elementelor din sirul S2 luate de la dreapta spre stanga.
   Exemplu:

```
S1: 1, 2, 3, 4
S2: 5, 6, 7
```

```
D: 1, 2, 3, 4, 7, 6, 5
```

4. Se dau doua siruri de octeti S1 si S2 de aceeasi lungime. Sa se construiasca sirul D astfel: fiecare element de pe pozitiile pare din D este suma elementelor de pe pozitiile corespunzatoare din S1 si S2, iar fiecare element de pe pozitiile impare are ca si valoare diferenta elementelor de pe pozitiile corespunzatoare din S1 si S2.
   Exemplu:

```
S1: 1, 2, 3, 4
S2: 5, 6, 7, 8
```

```
D: 6, -4, 10, -4
```

7. Se da un sir de octeti S. Sa se construiasca sirul D astfel: sa se puna mai intai elementele de pe pozitiile pare din S iar apoi elementele de pe pozitiile impare din S.
   Exemplu:

```
S: 1, 2, 3, 4, 5, 6, 7, 8
```

```
D: 1, 3, 5, 7, 2, 4, 6, 8
```

9. Se da un sir de octeti S de lungime l. Sa se construiasca sirul D de lungime l-1 astfel incat elementele din D sa reprezinte diferenta dintre fiecare 2 elemente consecutive din S.
   Exemplu:

```
S: 1, 2, 4, 6, 10, 20, 25
```

```
D: 1, 2, 2, 4, 10, 5
```

11. Se dau doua siruri de octeti S1 si S2 de aceeasi lungime. Sa se obtina sirul D prin intercalarea elementelor celor doua siruri.

Exemplu:

```
S1: 1, 3, 5, 7
S2: 2, 6, 9, 4
```

```
D: 1, 2, 3, 6, 5, 9, 7, 4
```

14. Se da un sir de octeti S. Sa se obtina sirul D1 ce contine toate numerele pare din S si sirul D2 ce contine toate numerele impare din S.
Exemplu:

```
S: 1, 5, 3, 8, 2, 9
D1: 8, 2
```

```
D2: 1, 5, 3, 9
```

17. Se da un sir de octeti S. Sa se construiasca sirul D1 ce contine elementele de pe pozitiile pare din S si sirul D2 ce contine elementele de pe pozitiile impare din S.
Exemplu:

```
S: 1, 5, 3, 8, 2, 9
D1: 1, 3, 2
```

```
D2: 5, 8, 9
```

20. Se dau 2 siruri de octeti S1 si S2 de aceeasi lungime. Sa se construiasca sirul D astfel incat fiecare element din D sa reprezinte maximul dintre elementele de pe pozitiile corespunzatoare din S1 si S2.
Exemplu:

```
S1: 1, 3, 6, 2, 3, 7
S2: 6, 3, 8, 1, 2, 5
```

```
D: 6, 3, 8, 2, 3, 7
```

23. Se da un sir de octeti S. Sa se construiasca un sir D1 care sa contina toate numerele pozitive si un sir D2 care sa contina toate numerele negative din S.
Exemplu:

```
S: 1, 3, -2, -5, 3, -8, 5, 0
D1: 1, 3, 3, 5, 0
```

```
D2: -2, -5, -8
```

26. Se da un sir de octeti S. Sa se obtina in sirul D multimea elementelor din S.
Exemplu:

```
S: 1, 4, 2, 4, 8, 2, 1, 1
```

```
D: 1, 4, 2, 8
```

28. Se dau doua siruri de caractere S1 si S2. Sa se construiasca sirul D ce contine toate elementele din S1 care nu apar in S2.

Exemplu:

```
S1: '+', '4', '2', 'a', '8', '4', 'X', '5'
S2: 'a', '4', '5'
```

```
D: '+', '2', '8', 'X'
```

31. Se da un sir de octeti S. Sa se determine maximul elementelor de pe pozitiile pare si minimul elementelor de pe pozitiile impare din S.
Exemplu:

```
S: 1, 4, 2, 3, 8, 4, 9, 5
max_poz_pare: 9
```

```
min_poz_impare: 3
```

34. Se da un sir de caractere S. Sa se construiasca sirul D care sa contina toate caracterele cifre din sirul S.
Exemplu:

```
S: '+', '4', '2', 'a', '8', '4', 'X', '5'
```

```
D: '4', '2', '8', '4', '5'
```

36. Se dau doua siruri de caractere S1 si S2. Sa se construiasca sirul D prin concatenarea elementelor de pe pozitiile multiplu de 3 din sirul S1 cu elementele sirului S2 in ordine inversa.
Exemplu:

```
S1: '+', '4', '2', 'a', '8', '4', 'X', '5'
S2: 'a', '4', '5'
```

```
D: '+', 'a', 'X', '5', '4', 'a'
```

39. Se da un sir de octeti S. Sa se construiasca sirul D ale carui elemente reprezinta suma fiecaror doi octeti consecutivi din sirul S.

```
S: 1, 2, 3, 4, 5, 6
```

```
D: 3, 5, 7, 9, 11
```