**Laborator 5 - Suport teoretic**

**Instrucţiuni de comparare, salt condiționat şi de ciclare. Operații pe şiruri.**

**Comparații între operanzi**

**Instrucţiunea CMP**

CMP <opd>, <ops>

* Instrucţiunea CMP realizează comparația între 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 instrucţiune NU COMPARĂ nimic, nestabilind nici un criteriu de comparaţie şi neluând de fapt nici o decizie, ci ea doar PREGĂTEŞTE decizia corespunzător cu flagurile setate, comparaţia efectivă şi decizia corespunzătoare fiind luată concret de instrucţiunea de salt condiționat care va fi folosită ulterior instrucţiunii CMP! Dacă nu folosim ulterior nici o instrucţiune decizională, CMP nu are nici un rol concret în vreo comparație, ea reprezentând doar o simplă scădere fictivă cu rol de afectare a flagurilor şi nu îşi va merita în nici un caz numele de CMP (compare).**
* Operatorul destinație poate să fie registru sau variabilă în memorie.
* Operandul sursa poate să fie registru, variabilă în memorie sau constantă.
* Ambii operanzi ai instrucţiunii CMP trebuie să fie de aceeași dimensiune.

***Exemplul 1:***

cmp eax, ebx ; ”compara” valorile stocate in cei doi regiştri (scădere fictiva eax-ebx)

jle done ;în funcţie de instrucţiunea de salt condiţionat utilizată (aici JLE) se stabilește criteriul de comparare.

;In acest caz: daca conţinutul din EAX in interpretarea cu semn este mai mic sau egal cu conţinutul din EBX atunci JUMP la eticheta Done,

;Altfel continuă cu următoarea instrucţiune (flagul testat aici este ZF).

done:

;instrucţiuni care urmează etichetei done

***Exemplul 2:***

mov al,200 ; AL = C8h

mov bl,100

cmp al, bl ; se realizează scăderea fictiva al-bl şi se setează

; flagurile in mod corespunzător acesteia (in acest caz vom avea SF=0, OF=1, CF=0 şi ZF=0)

JB et2 ;instrucţiunea de salt condiționat stabilește criteriul de comparare, in acest caz Jump if Below – comparație pentru numere fără semn (este 200 BELOW 100 ?) şi se testează conţinutul 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 instrucţiuni care urmează

et2:

;............. ;set de instrucţiuni care urmează etichetei

***Exemplul 3:***

mov al,-56 ; AL = C8h = 200 in interpretarea fără semn

mov bl,100

cmp al, bl ;se realizează scăderea fictiva al-bl şi se setează flagurile in mod corespunzător acesteia (pentru cazul nostru vom avea SF=0, OF=1, CF=0 şi ZF=0)

JNGE et2 ;se verifica condiția JNGE - Jump if not greater or equal

;(comparaţie CU SEMN -56 fată de 100)

;concret se verifica daca este diferit conţinutul din SF şi 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 fără semn

mov bl,100

cmp al, bl ;se realizează scăderea fictivă al-bl şi se setează flagurile în mod corespunzător acesteia (pentru cazul nostru vom avea SF=0, OF=1, CF=0 şi ZF=0)

JNBE et2 ;se verifica condiția JNBE - Jump if not below or equal

;(comparaţie fără SEMN  200 fată de 100)

;concret se verifica daca CF=0 şi ZF=0

;Având în 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

**Instrucţiunea TEST**

TEST <opd>, <ops>

* Instrucţiunea TEST realizează operația logică şi între cei doi operanzi (execuție fictivă ops **AND** opd) fără a salva rezultatul operației în vreunul dintre cei doi operanzi.
* Ambii operanzi ai instrucţiunii TEST trebuie să fie de aceeaşi dimensiune.
* Singurul efect al unei instrucţiuni TEST este modificarea conţinutului flagurilor specificate în tabelul de mai sus corespunzător cu rezultatul operației 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 instrucţiuni 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 stabilește criteriul de comparare.

;In acest caz: jump la eticheta if ZF = 1

; alternativ se putea folosi aici varianta jz AmbiiOPeranziZero, aceste doua instrucţiuni de salt condiționat (JE şi JZ) fiind similare in ceea ce privește condiția 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 stabilește criteriul de comparare.

; În acest caz: dacă rezultatul operației AH AND 0F2h este un număr strict negativ în interpretarea cu semn (adică dacă bitul de semn al rezultatului este 1) atunci Jump ;la eticheta et2 (flagul testat este SF).

et2: ;set de instrucţiuni care compun eticheta....

* Similar cu situația de la instrucţiunea CMP, avem de făcut şi aici următoarea observaţie:
* **Deşi numele ei este TEST este important de subliniat ca în realitate aceasta instrucţiune 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 şi decizia corespunzătoare fiind luată concret de instrucţiunea de salt condiționat care va fi folosită ulterior instrucţiunii TEST! Dacă nu folosim ulterior nici o instrucţiune decizională, TEST nu are nici un rol concret în vreo testare, ea reprezentând doar o simplă operație ŞI bit cu bit cu rol de afectare a flag-urilor şi nu îşi va merita în nici un caz numele de TEST (testare a unei condiții).**

**Salturi condiționate de flag-uri**

* Când se compară două numere cu semn se folosesc termenii "less than" (mai mic decât) şi "greater than" (mai mare decât), iar când se compară doua numere fără semn se folosesc termenii "below" (inferior, sub) şi respectiv "above" (superior, deasupra, peste).
* Redăm mai jos o parte din documentația INTEL, referitoare la aceste instrucţiuni de salt. ([🔗 Combined Volume Set of Intel® 64 and IA-32 Architectures Software Developer’s Manuals, începând de la pag. 1058](https://software.intel.com/en-us/articles/intel-sdm))

**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 conţinues 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.
* Sumarizând, 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.

În tabelul următor aveţi grupate toate instrucţiunile 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”.

|  |  |  |
| --- | --- | --- |
| **MNEMONICĂ** | **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**  **JNL** | este mai mare sau egal  nu este mai mic decât | SF=OF |
| **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.

**Instrucţiuni de ciclare**

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

[instrucţiune] 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 octeţi - atenţie deci la "distanţa" 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 biți citiți 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 ; spațiu 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 (funcțiile externe au

push ebx ; drept să-l altereze)

push fmt ; cei doi parametri sunt în stivă

call [scanf] ; apel scanf cu "%d" şi 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 instrucţiunii

; de după .bucla

* Instrucţiunile **JECXZ** şi **JCXZ** sunt instrucţiuni de **jump** condițional dar diferă de setul standard al acestor instrucţiuni prin faptul ca acestea NU verifica statusul flag-urilor. **JECXZ** şi **JCXZ** verifica numai daca conţinutul din ECX sau CX are valoarea 0. Aceste instrucţiuni pot fi utilizate la începutul unei bucle care se termina cu un **loop** condițional pentru a preveni intrarea în buclă când ECX sau CX au valoarea 0. Dacă se va intra în bucla cu CX=0, având în vedere faptul ca “se efectuează întâi decrementarea registrului ECX şi apoi se face testul şi eventual saltul” acest aspect va determina execuția 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

; instrucţiuni

Loop forIndex ; repeat

endFor

Echivalent cu:

mov ECX, numar

cmp ECX,0

JZ endFor

forIndex

; instrucţiuni

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 5 - Exemple

## Instrucţiuni de comparare, salt condiționat şi de ciclare. Operații pe şiruri.

## Exemplu

;Se da un sir de caractere format din litere mici.

;Sa se transforme acest sir in şirul literelor mari corespunzător.

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 şirului iniţial s

l equ $-s ; stabilirea lungimea şirului iniţial l

d times l db 0 ; rezervarea unui spațiu de dimensiune l pentru şirul destinație d şi iniţializarea acestuia

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 Sfârșit

Repeta:

mov al, [s+esi]

mov bl, 'a'-'A' ; pentru a obține litera mare corespunzătoare literei mici, vom scădea din codul ASCII

; al literei mici diferența dintre 'a'-'A'

sub al, bl

mov [d+esi], al

inc esi

loop Repeta

Sfârșit:;terminarea programului

; exit(0)

push dword 0 ; push the parameter for exit onto the stack

call [exit] ; call exit to terminate the program

**Laborator 5 - Probleme propuse**

**Instrucţiuni de comparare, salt condiționat şi de ciclare. Operații pe şiruri.**

**Exerciții**

1. Se da un sir de octeţi S de lungime l. Sa se construiască şirul D de lungime l-1 astfel încât elementele din D sa reprezinte produsul dintre fiecare 2 elemente consecutive S(i) şi S(i+1) din S.

**Exemplu:**

S: 1, 2, 3, 4

D: 2, 6, 12

1. Se da un sir de caractere S. Sa se construiască şirul D care sa conţină toate caracterele speciale (!@#$%^&\*) din şirul S.

**Exemplu:**

S: '+', '4', '2', 'a', '@', '3', '$', '\*'

D: '@','$','\*'

1. Se dau doua şiruri de octeţi S1 şi S2. Sa se construiască şirul D prin concatenarea elementelor din şirul S1 1uate de la stânga spre dreapta şi a elementelor din şirul S2 luate de la dreapta spre stânga.

**Exemplu:**

S1: 1, 2, 3, 4

S2: 5, 6, 7

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

1. Se dau doua şiruri de octeţi S1 şi S2 de aceeași lungime. Sa se construiască şirul D astfel: fiecare element de pe poziţiile pare din D este suma elementelor de pe poziţiile corespunzătoare din S1 şi S2, iar fiecare element de pe poziţiile impare are ca şi valoare diferenţa elementelor de pe poziţiile corespunzătoare din S1 şi S2.

**Exemplu:**

S1: 1, 2, 3, 4

S2: 5, 6, 7, 8

D: 6, -4, 10, -4

1. Se da un sir de caractere S. Sa se construiască şirul D care sa conţină toate literele mici din şirul S.

**Exemplu:**

S: 'a', 'A', 'b', 'B', '2', '%', 'x'

D: 'a', 'b', 'x'

1. Se da un sir de octeţi S. Sa se construiască şirul D astfel: sa se pună mai întâi elementele de pe poziţiile pare din S iar apoi elementele de pe poziţiile impare din S.

**Exemplu:**

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

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

1. Se dau doua şiruri de octeţi S1 şi S2 de aceeași lungime. Sa se obţină şirul D prin intercalarea elementelor celor două şiruri.

**Exemplu:**

S1: 1, 3, 5, 7

S2: 2, 6, 9, 4

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

1. Se dă un sir de caractere S. Sa se construiască şirul D care să conţină toate literele mari din şirul S.

**Exemplu:**

S: 'a', 'A', 'b', 'B', '2', '%', 'x', 'M'

D: 'A', 'B', 'M'

1. Se dă un sir de octeţi S de lungime l. Sa se construiască şirul D de lungime l-1 astfel încât elementele din D sa reprezinte diferența dintre fiecare 2 elemente consecutive din S.

**Exemplu:**

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

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

1. Se dau două şiruri de caractere S1 şi S2. Sa se construiască şirul D prin concatenarea elementelor şirului S2 in ordine inversa cu elementele de pe poziţiile pare din şirul S1.

**Exemplu:**

S1: '+', '2', '2', 'b', '8', '6', 'X', '8'

S2: 'a', '4', '5'

D: '5', '4', 'a', '2','b', '6', '8'

1. Se da un sir de octeţi S. Să se obţină şirul D1 ce conţine toate numerele pare din S şi şirul D2 ce conţine toate numerele impare din S.

**Exemplu:**

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

D1: 8, 2

D2: 1, 5, 3, 9

1. Se dau doua şiruri de caractere S1 şi S2. Sa se construiască şirul D prin concatenarea elementelor de pe poziţiile pare din S2 cu elementele de pe poziţiile impare din S1.

**Exemplu:**

S1: 'a', 'b', 'c', 'd', 'e', 'f'

S2: '1', '2', '3', '4', '5'

D: '2', '4','a','c','e'

1. Se da un sir de octeţi S. Sa se construiască şirul D1 ce conţine elementele de pe poziţiile pare din S şi şirul D2 ce conţine elementele de pe poziţiile impare din S.

**Exemplu:**

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

D1: 1, 3, 2

D2: 5, 8, 9

1. Se dă un sir de octeţi S. Sa se construiască un sir D1 care sa conţină toate numerele pozitive şi un sir D2 care sa conţină 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

1. Se dau 2 şiruri de octeţi A şi B. Sa se construiască şirul R care să conţină elementele lui B în ordine inversa urmate de elementele impare ale lui A.

**Exemplu:**

A: 2, 1, 3, 3, 4, 2, 6

B: 4, 5, 7, 6, 2, 1

R: 1, 2, 6, 7, 5, 4, 1, 3, 3

1. Se dau două şiruri de caractere S1 şi S2. Sa se construiască şirul D prin concatenarea elementelor de pe poziţiile impare din S2 cu elementele de pe poziţiile pare din S1.

**Exemplu:**

S1: 'a', 'b', 'c', 'b', 'e', 'f'

S2: '1', '2', '3', '4', '5'

D: '1', '3', '5', 'b', 'b', 'f'

1. Se dau 2 şiruri de octeţi S1 şi S2 de aceeași lungime. Sa se construiască şirul D astfel încât fiecare element din D sa reprezinte maximul dintre elementele de pe poziţiile corespunzătoare din S1 şi S2.

**Exemplu:**

S1: 1, 3, 6, 2, 3, 7

S2: 6, 3, 8, 1, 2, 5

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

1. Se dau 2 şiruri de octeţi A şi B. Sa se construiască şirul R care sa conţină doar elementele impare şi pozitive din cele 2 şiruri.

**Exemplu:**

A: 2, 1, 3, -3

B: 4, 5, -5, 7

R: 1, 3, 5, 7

1. Se dau 2 şiruri de octeţi A şi B. Sa se construiască şirul R care sa conţină doar elementele pare şi negative din cele 2 şiruri.

**Exemplu:**

A: 2, 1, 3, -3, -4, 2, -6

B: 4, 5, -5, 7, -6, -2, 1

R: -4, -6, -6, -2

1. Se dau 2 şiruri de octeţi A şi B. Sa se construiască şirul R care sa conţină elementele lui B în ordine inversă urmate de elementele pare ale lui A.

**Exemplu:**

A: 2, 1, 3, 3, 4, 2, 6

B: 4, 5, 7, 6, 2, 1

R: 1, 2, 6, 7, 5, 4, 2, 4, 2, 6

1. Se dau 2 şiruri de octeţi A şi B. Sa se construiască şirul R care să conţină elementele lui B în ordine inversă urmate de elementele negative ale lui A.

**Exemplu:**

A: 2, 1, -3, 3, -4, 2, 6

B: 4, 5, 7, 6, 2, 1

R: 1, 2, 6, 7, 5, 4, -3, -4

1. Se dau 2 şiruri de octeţi S1 şi S2 de aceeași lungime. Sa se construiască şirul D astfel încât fiecare element din D sa reprezinte minimumul dintre elementele de pe poziţiile corespunzătoare din S1 şi S2.

**Exemplu:**

S1: 1, 3, 6, 2, 3, 7

S2: 6, 3, 8, 1, 2, 5

D: 1, 3, 6, 1, 2, 5

1. Se da un sir de octeţi S. Sa se obţină în şirul D mulţimea elementelor din S.

**Exemplu:**

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

D: 1, 4, 2, 8

1. Se dau 2 şiruri de octeţi A şi B. Sa se construiască şirul R care sa conţină elementele lui B în ordine inversa urmate de elementele in ordine inversa ale lui A.

**Exemplu:**

A: 2, 1, -3, 0

B: 4, 5, 7, 6, 2, 1

R: 1, 2, 6, 7, 5, 4, 0, -3, 1, 2

1. Se dau doua şiruri de caractere S1 şi S2. Sa se construiască şirul D ce conţine 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'

1. Se dă un sir de octeţi S. Sa se determine maximul elementelor de pe poziţiile pare şi minimul elementelor de pe poziţiile impare din S.

**Exemplu:**

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

max\_poz\_pare: 9

min\_poz\_impare: 3

1. Se dau 2 şiruri de octeţi S1 şi S2 de aceeaşi lungime. Să se construiască şirul D astfel încât fiecare element din D să reprezinte diferenţa dintre elementele de pe poziţiile corespunzătoare din S1 şi S2.

**Exemplu:**

S1: 1, 3, 6, 2, 3, 2

S2: 6, 3, 8, 1, 2, 5

D: -5, 0, -2, 1, 1, -3

1. Se dau doua şiruri de caractere S1 şi S2. Să se construiască şirul D prin concatenarea elementelor de pe poziţiile multiplu de 3 din şirul S1 cu elementele şirului S2 în ordine inversă.

**Exemplu:**

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

S2: 'a', '4', '5'

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

1. Se dă un şir de octeţi S. Sa se construiască şirul D ale cărui elemente reprezintă suma fiecăror doi octeţi consecutivi din şirul S.

**Exemplu:**

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

D: 3, 5, 7, 9, 11

1. Se dau 2 şiruri de octeţi S1 şi S2 de aceeași lungime. Să se construiască şirul D astfel încât fiecare element din D sa reprezinte suma dintre elementele de pe poziţiile corespunzătoare din S1 şi S2.

**Exemplu:**

S1: 1, 3, 6, 2, 3, 2

S2: 6, 3, 8, 1, 2, 5

D: 7, 6, 14, 3, 5, 7

1. Se dă un şir de caractere S. Să se construiască şirul D care să conţină toate caracterele cifre din şirul S.

**Exemplu:**

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

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

1. Se dă un şir de octeţi S de lungime l. Să se construiască şirul D de lungime l-1 astfel încât elementele din D să reprezinte câtul dintre fiecare 2 elemente consecutive S(i) şi S(i+1) din S.

**Exemplu:**

S: 1, 6, 3, 1

D: 0, 2, 3