


```

bits 32 ; assembling for the 32 bits architecture
; declare the EntryPoint (a label defining the very first instruction of the program)
global start
; declare external functions needed by our program
extern exit ; 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
; our data is declared here (the variables needed by our program)
segment data use32 class=data
    A db 2,1,3,-3 ;definim sirul a
    B db 4,5,-5,7 ;definim sirul b
    l equ 4 ;lungimea sirului a
    r equ 4 ;lungimea sirului b
    R times 10 db 0 ;rezervam spatiu pentru sirul destinatie
; our code starts here
segment code use32 class=code
start:
    mov ECX,l ;pregatim registrul ECX pentru a parcurge sirul
    mov ESI,0 ;Initializam registrul indice cu 0
    mov EDX,0
    jecxz sf ;tratam cazul in care ECX e 0
start_loop:
    mov AL,[A+ESI] ;punem in AL, elementul curent
    test AL,00000001b ;daca elemntul este impar ZF=0
    jz par ;daca elemntul este par, adica ZF=1 atunci nu executam instructiunea
    cmp AL,0
    jnge negat ;daca elemntul este negativ, nu executam
    mov [R+EDX],AL ;daca elemntul indeplineste conditiile, il punem in sir
    inc EDX ; incrementam lungimea sirului de facut
    negat:
    par:
    inc ESI ;incrementam indicele la care ne aflam
loop start_loop
sf:
    mov ECX,r ;pregatim registrul ECX pentru a parcurge sirul
    mov ESI,0 ;Initializam registrul indice cu 0
    jecxz sf2 ;tratam cazul in care ECX e 0
start_loop2:
    mov AL,[B+ESI] ;punem in AL, elementul curent
    test AL,01h ;daca elemntul este impar ZF=0
    jz par2 ;daca elemntul este par, adica ZF=1 atunci nu executam instructiunea
    cmp AL,0 ;seteaza OF si SF
    jnge negat2 ;daca elemntul este negativ, nu executam
    mov [R+EDX],AL ;daca elemntul indeplineste conditiile, il punem in sir
    inc EDX ; incrementam lungimea sirului de facut
    negat2:
    par2:
    inc ESI ;incrementam indicele la care ne aflam
loop start_loop2
sf2:    push    dword 0 ; push the parameter for exit onto the stack
call [exit] ; call exit to terminate the program

```

25. 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'

The screenshot displays a debugger interface with two main panels. The left panel shows assembly code with instructions like `MOV ECX, 8`, `MOV ESI, 0`, `JEQZ SHORT 00402002`, and a loop structure using `DEC BYTE PTR DS:[EDI+401000]` and `JECXZ sf`. The right panel shows the state of CPU registers, including `EAX` (00000000), `ECX` (00000008), `EDX` (00000000), `EDI` (00402000), `ESI` (00000000), `EIP` (00402002), and `EFL` (00000000). The `Registers (FPU)` window is also visible, showing floating-point registers.

```

12 segment data use32 class=data
13     S1 db '+','4','2','a','8','4','X','5' ; definim sirul S1
14     l equ $-S1
15     S2 db 'a','4','5'
16     r equ $-S2
17     D times 20 db 0 ; D: '+', '2', '8', 'X'
18     ap resb 1
19     cont resb 1
20
21 ; our code starts here
22 segment code use32 class=code
23     start:
24         mov ECX, 1 ; punem in registrul contor dimensiunea primului sir
25         mov ESI, 0 ; indicele de la care incepem iterarea
26         mov EDI, 0 ; indicele din sirul destinatie (D)
27         jecxz sf ; tratam cazul in care ECX e 0
28         loop_start:
29
30         mov AL, [S1+ESI] ;punem in AL elementul curent 0
31         mov byte[ap], 0 ;initializam [ap] cu 0, intrucat ea va contoriza numarul de aparitii a lui AL in S2
32         mov byte[cont], r ;ne luam o variabila contor, intrucat ECX este ocupat, cu ea implementand o varianta proprie de loop
33         mov EBX, 0 ;initializam EBX, pentru al folosi pe post de contor in noul sir
34         strt:
35
36         mov DL, [S2+EBX] ;punem in DL elemntul curent
37
38         cmp AL, DL ; verificam egalitatea cu DL
39         jne et
40         inc byte[ap] ; daca cele doua sunt egale, incrementam variabila ce retine numarul de aparitii
41         et:
42
43         inc EBX ;incrementam variabila contor pentru sirul S2
44         dec byte[cont] ;decrementam contorul loop-ului
45         cmp byte[cont], 0;comparam contorul loop-ului cu 0 pentru a vedea daca mai avem de executat pasi
46         jne strt
47
48         cmp byte[ap], 0 ;comparam numarul de aparitii cu 0 pentru a vedea daca exista caracterul cautat in S2
49         jne eth
50         mov [D+EDI], AL ;in cazul in care nu exista punem caracterul in noul sir, cel destinatie, adica D
51         inc EDI ;incrementam registrul index pentru sirul D
52         eth:
53         inc ESI ;incrementam registrul index pentru sirul S1
54
55     loop loop_start
56     sf:
57
58     push dword 0 ; push the parameter for exit onto the stack
59     call [exit] ; call exit to terminate the program

```

```

bits 32 ; assembling for the 32 bits architecture
; declare the EntryPoint (a label defining the very first instruction of the program)
global start
; declare external functions needed by our program
extern exit ; 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
; our data is declared here (the variables needed by our program)
segment data use32 class=data
    S1 db '+','4','2','a','8','4','X','5' ; definim sirul S1
    l equ $-S1 ; determinam lungimea sirului S1
    S2 db 'a','4','5'
    r equ $-S2 ; determinam lungimea sirului S2
    D times 20 db 0 ;D: '+', '2', '8', 'X'
    ap resb 1
    cont resb 1
; our code starts here
segment code use32 class=code
start:
    mov ECX, l ; punem in registrul contor dimensiunea primului sir
    mov ESI, 0 ; indicele de la care incepem iterarea
    mov EDI, 0 ; indicele din sirul destinatie (D)
    jecxz sf ; tratam cazul in care ECX e 0
loop_start:
    mov AL,[S1+ESI] ;punem in AL elementul curent 0
    mov byte[ap],0 ;initializam [ap] cu 0, intrucat ea va contoriza numarul de aparitii a lui AL in S2
    mov byte[cont],r ;ne luam o variabila contor, intrucat ECX este ocupat, cu ea implementand o varianta proprie de loop
    mov EBX,0 ;intializam EBX, pentru al folosi pe post de contor in noul sir
strt:
    mov DL,[S2+EBX] ;punem in DL elemntul curent

    cmp AL,DL ; verificam egalitatea cu DL
    jne et
    inc byte[ap] ; daca cele doua sunt egale, incrementam variabila ce retine numarul de aparitii
et:

    inc EBX ;incrementam variabila contor pentru sirul S2
    dec byte[cont] ;decrementam contorul loop-ului
    cmp byte[cont],0;comparam contorul loop-ului cu 0 pentru a vedea daca mai avem de executat pasi
    jne strt

    cmp byte[ap],0 ;comparam numarul de aparitii cu 0 pentru a vedea daca exista caracterul cautat in S2
    jne eth
    mov [D+EDI],AL ;in cazul in care nu exista punem caracterul in noul sir, cel destinatie, adica D
    inc EDI ;incrementam registrul index pentru sirul D
eth:
    inc ESI ;incrementam registrul index pentru sirul S1
loop loop_start
sf:

push dword 0 ; push the parameter for exit onto the stack
call [exit] ; call exit to terminate the program

```