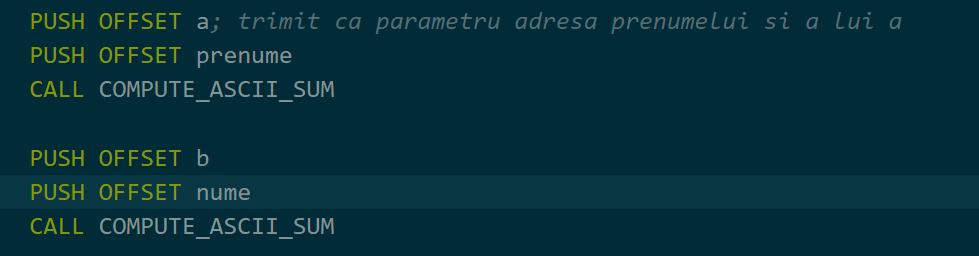
Tema semestrială

Std. Cap. Leancă Răzvan-Andrei

# Calcularea valorilor a și b:

Calcularea celor două valori se face prin apelul rutinei COMPUTE\_ASCII\_SUM. Inainte de apelul rutinei, sunt adăugate pe stivă adresa la care va fi încărcat rezultatul și valorile pe baza cărora se va face calculul. (vezi Figură 1)

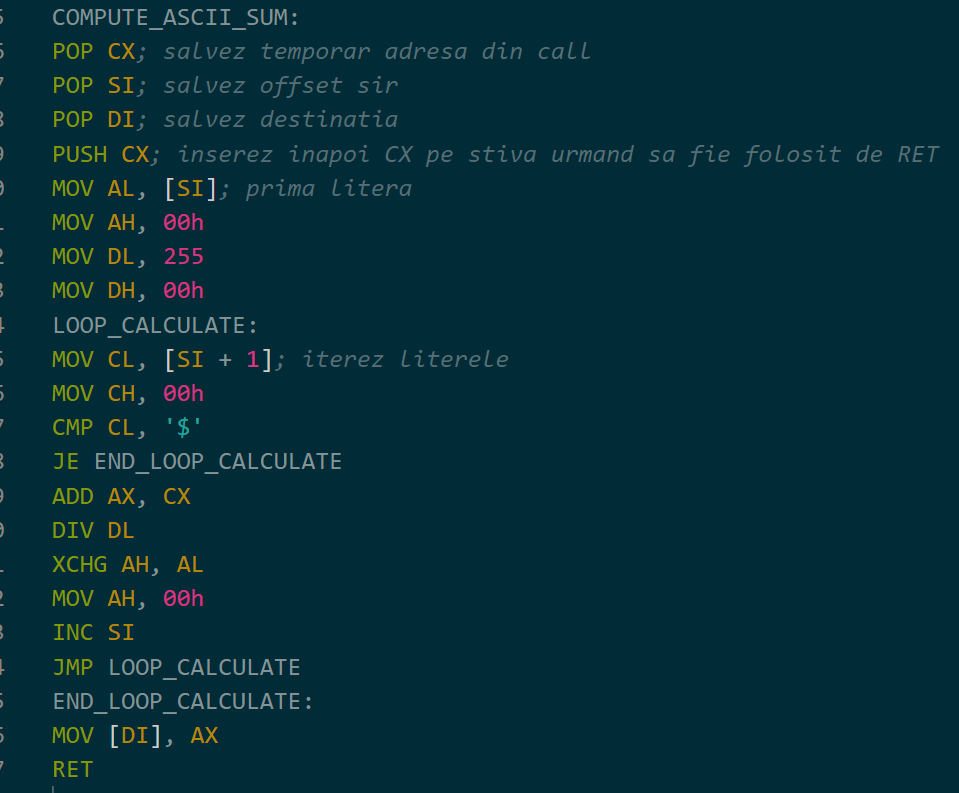


Figură 1

Având în vedere că la apel, pe stivă este adăugată și adresa la care se va executa operația RET pentru a ne întoarce la următoare instrucțiune de după apel, în cadrul procedurii am făcut un artificiu, astfel că am scos de pe stivă întâi valoarea menționată anterior pe un registru, urmând ca după scoaterea valorilor utile calcului, această adresă să fie introdusă înapoi pe stivă, astfel încât operația RET să se execute corespunzător (vezi Figură 2). Registrul utilizat pentru salvarea temporară a adresei este CX.

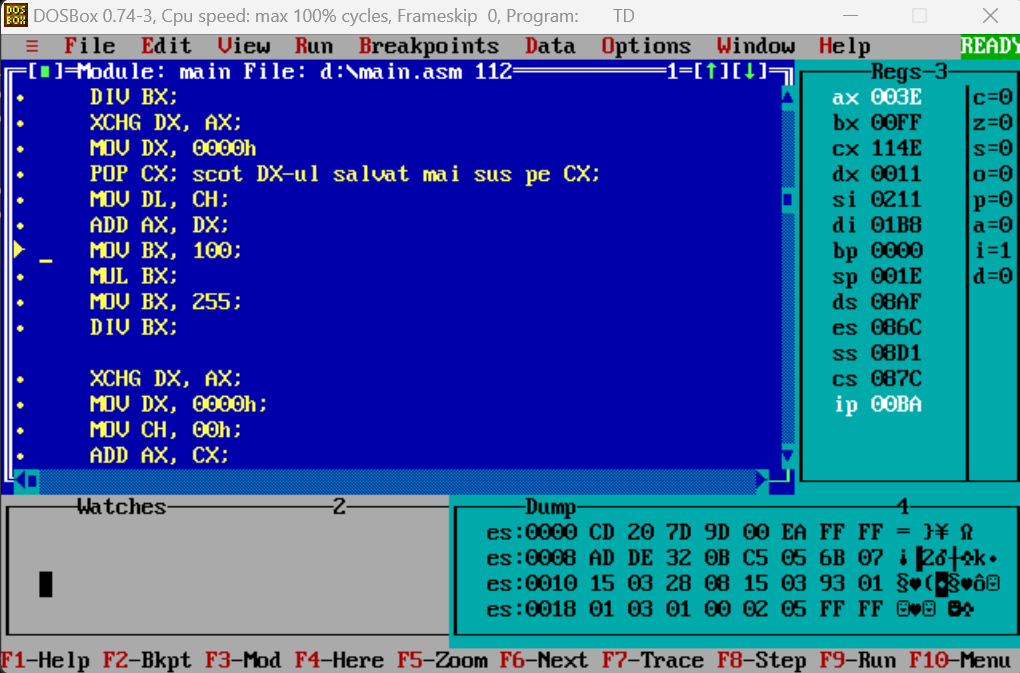
Procedura adaugă în AX valoarea primei litere, după care însumează fiecare literă din șirul de caractere cu acest registru și execută operația de împărțire cu valoarea 255 astfel încât rezultatul să rămână tot timpul pe un octet.

Această sumă se execută până când este întâlnit caracterul “$”

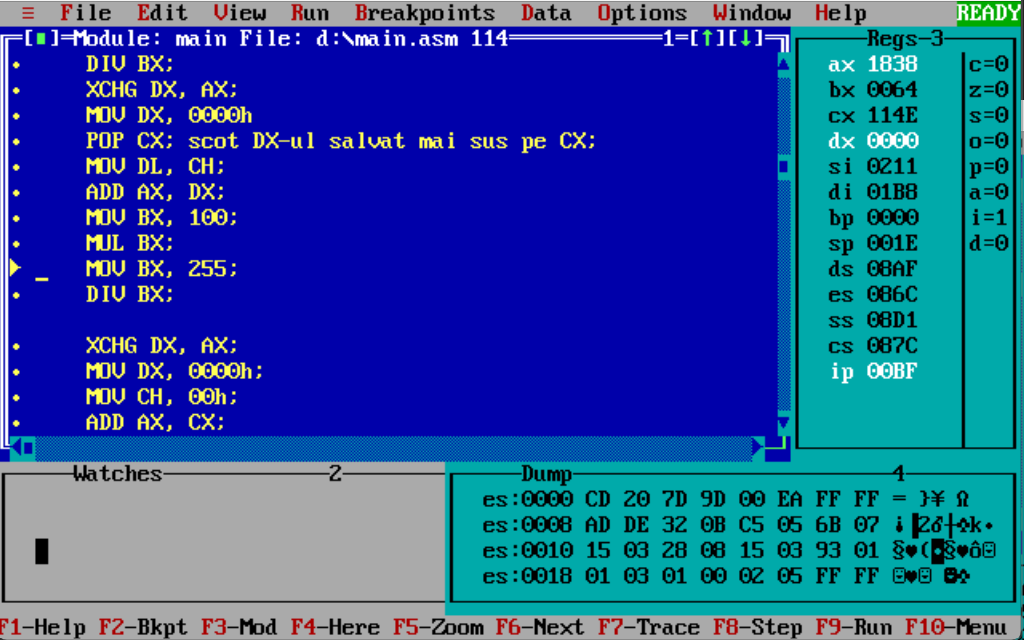


Figură 2

# Calcularea valorilor ș :

Pentru a mă asigura că în cadrul calculelor nu depășesc niciodată formatul de reprezentare, am distribuit operația de mod în cadrul întregii formule. În principiu, operația mod 255 este executată la fiecare operație de înmulțire, cu excepția cazurilor în care nu este neapărat necesar. Un astfel de exemplu este înmulțirea cu registrul BX în care se află valoarea 100; prin exploatarea faptului că înmulțirea dintre AX și un registru salvează rezultatul pe DX concatenat cu AX și că elementul din AX nu depășește un octet fiind rezultatul unei operații mod 255, formatul nu va fi depășit niciodată.   
În figura 3 și figura 4 se poate observa că după execuția înmulțirii, se modifică atât AX cât și DX.  
Rezultatul final este salvat și în x, cât și in x0.  
  


Figură 3



Figură 4

# Criptarea mesajului

Procedura de criptare folosește un loop în cadrul căruia se realizeză XOR între valoarea actuală a lui x și elementele din mesaj, rezultatul fiind salvat în „temp”, cu care voi realiza codificarea la următorul pas și in „encrypted”, șir pe care îl folosesc doar la afișare (am modificat procedura WRITE).

La fiecare pas, incrementez “iterations”, valoare pe care o voi folosi de asemenea la codificare.

# Codificare

În cadrul procedurii, asignez valoarea zero pentru “iterations” deoarece urmează să fie folosită in cadrul codificări pe post de iterator.

În primă fază, calculez dacă este nevoie să adaug octeți la final pentru padding; acest lucru este determinat prin faptul ca verific dacă lungimea mesajului și implicit, a mesajului criptat este divizibilă cu 3. Dacă este, nu va fi nevoie de padding, iar dacă nu este, lungimea mesajului este incrementată cu 1 asftel încât, la final, mesajul să fie completat în “main\_loop\_encode” cu numarul de caractere “B” necesare care vor fi convertite la final în “+” de catre procedura ADD\_PADDING.

Codificarea se realizează prin procesarea a cate 4 grupuri de cate șase biți, aducând in registre câte 3 octeți din mesajul criptat. Pentru fiecare grup am cate o procesare în care am realizat operațiile necesare codificarii. Selectarea caracterelor din alfabetul de codificare o fac prin folosirea celor 6 biți ca index în alfabetul reținut la adresa dată de eticheta “code\_table”.

“Iterations\_encoded” reține lungimea mesajului dupa codificare și este utilizat de către procedura ADD\_PADDING.

# Adaugarea paddingului

Padding-ul se adaugă prin calcularea numarului de caractere de pad care sunt necesare. Acest lucru se determină prin scaderea restului la împărțirea cu 3 a lungimii mesajului.

După, utilizând un loop, modific la finalul mesajului codat acele caractere “B” în caractere “+”.

# Observații

* Am modificat procedura WRITE, prin faptul că afișează în fișierul out.txt mesajul criptat de la adresa dată de eticheta “encrypted” adăugată de mine.