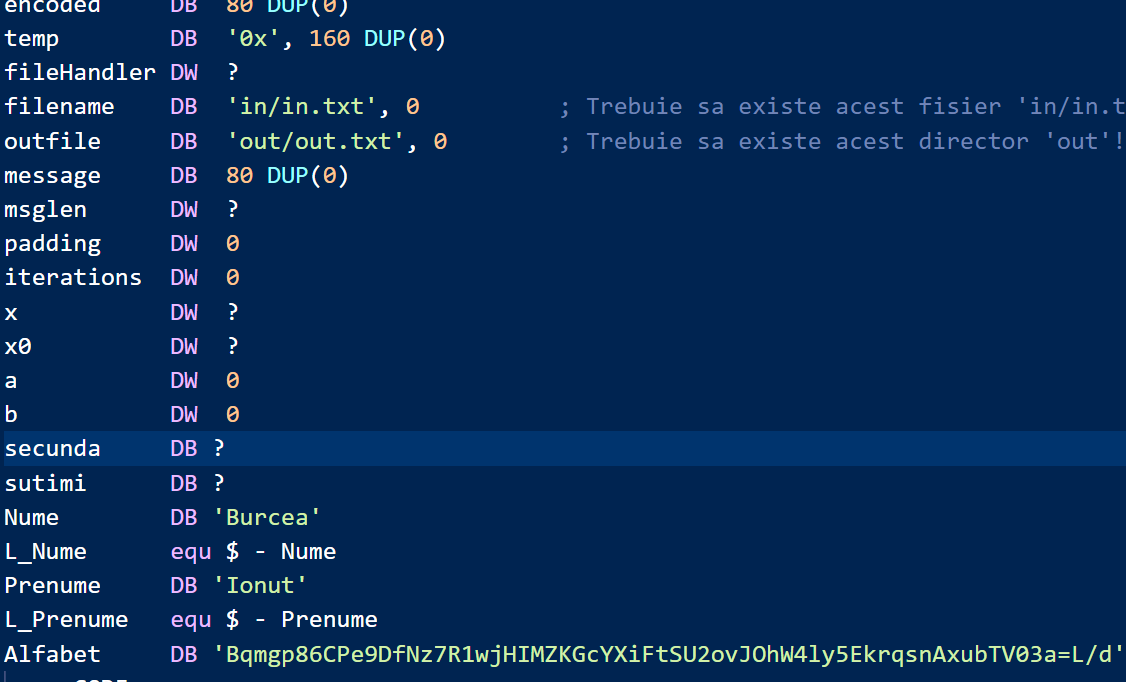
Document explicativ pentru tema de semestru la disciplina

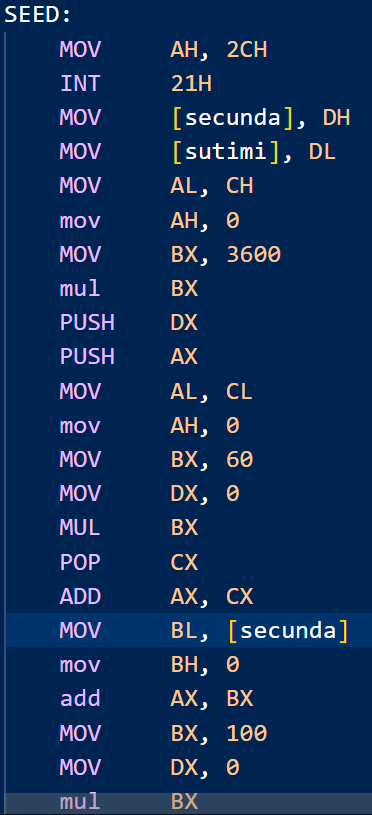
Arhitectura sistemelor de calcul

Burcea Ionut

C112E



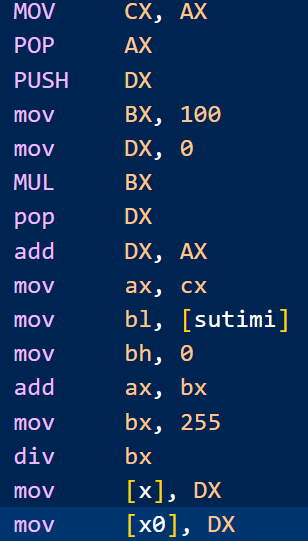
Secțiunea de declarare de variabile are în plus față de scheletul temei câteva elemente noi precum două variabile, “secunda” care va reține numărul de secunde generate de funcția 2Ch și “sutimi” care reține sutimile de secundă generate de această funcție. Am hardcodat pentru generarea coeficiențior a și b și numele și prenumele meu, iar pentru a prelua lungimea acestor șiruri am folosit o constantă și simboul $ care mă ajută să rețin dimensiunea variabilei “nume”, făcând scăderea dintre offset-ul acestei constante și offset-ul variabilei “nume”, prin offset referindu-mă la adresa de început a variabilei, respectiv a constantei în segmentul de date. În mod similar am procedat și la prenume. În final am hardocat și variabila “Alfabet”, unde am salvat alfabetul de codare pentru codificarea COD64.



În subrutina SEED, după tratarea întreruperii pentru furnizarea timpului sistemului, am decis să salvez valorille secundelor și sutimilor de secunde în variabilele aferente deoarece în continuare în cod pierd aceste valori în urma altor operații care modifică registrul DX.

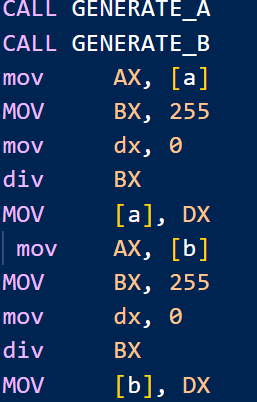
Pentru a genera primul coeficient, x0, am folosit prima formulă pusă la dispoziție, adică

x0=((CH\*3600 + CL\*60 + DH)\*100 + DL) mod 255. Pentru a forma corect x0, am luat în considerare și cazurile în care înmulțirile cu 3600, respective 100 ar putea depăși formatul de reprezentare de 2 octeți, maximul unui registru din procesorul 8086. Așadar, am decis să salvez depășirea acestor operații pe stivă și să gestionez această problemă la finalul generării lui x0, unde concatenez cele 2 rezultate din registrul DX.

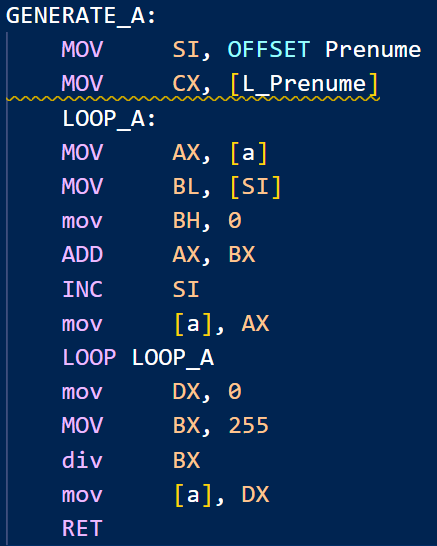


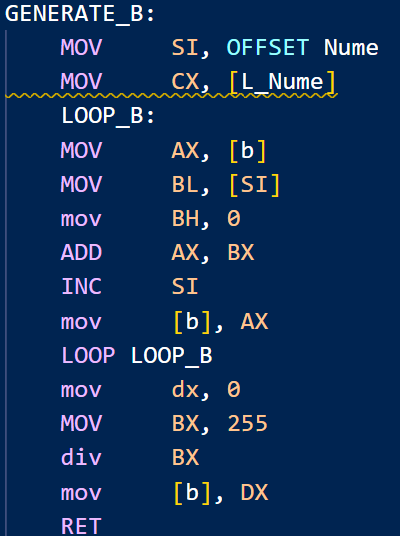
Pentru a concatena posibilele depășiri apărute, așa cum am menționat mai devreme, depășirea de la înmulțirea lui CH cu 3600 am salvat-o pe stivă, iar după înmulțirea cu 100, care poate genera și ea o depășire, am scos depășirea veche de pe stivă și am înmulțit-o cu 100 și am adunat-o apoi la noua depășire tocmai pentru a realiza o operație mod 255 corectă la final.

Acest lucru a fost practic lucrul mai dificil în ceea ce privește realizarea operației de encrypt a mesajului dat ca parametru de intrare.

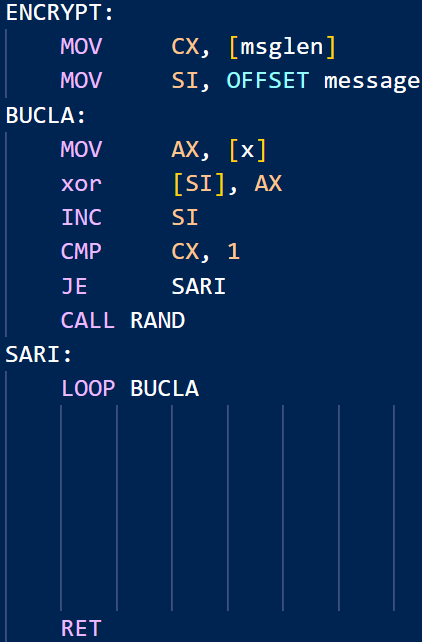


După generarea lui x0 și implicit a lui x(termenul curent generat), generez și coeficienții a și b pe baza prenumelui, respective numelui meu care sunt salvate în variabilele pe care le-am menționat mai sus. După generarea acestor coeficienți efectuez și operația mod 255 pe fiecare așa cum este menționat în cerință, iar rezultatele le salvez în variabilele a și b.

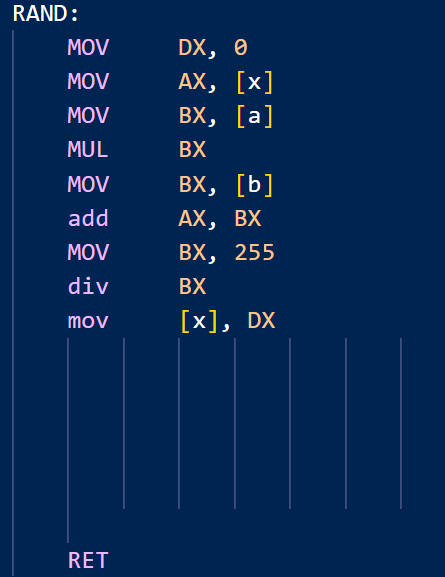




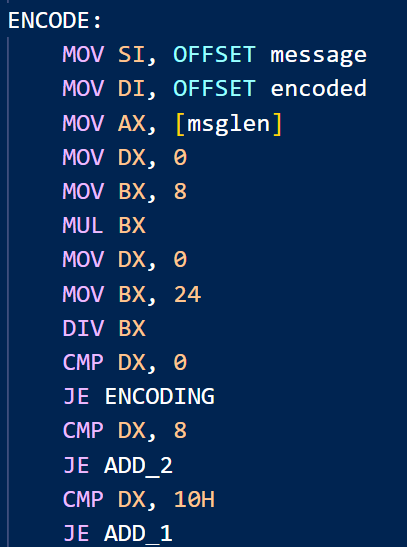
Atât în generarea lui a, cât și în generarea lui b, mă folosesc de registrul SI(source index) pentru a itera variabilele Nume și Prenume. În funcțiile de generare, adun fiecare caracter din nume și prenume(adică valorile în hexazecimal aferente) în registrul AX, apoi rezultatul în salvez în variabila aferentă.



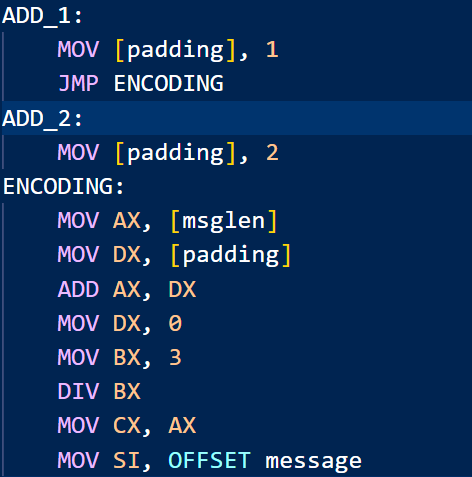
După generarea lui x0 și a coeficienților a și b, execut subrutina ENCRYPT, subrutină care ia fiecare caracter al șirului de intrare și în encryptează folosind operația XOR pe baza termenului curent generat xn cu n de la 0 la lungimea șirului – 1(acest lucru e garantat prin comparația iterației curente cu 1). Dacă nu s-a ajung la această iterație, se generează în continuu pe baza termenului anterior și folosind formula xn=(xn-1\*a + b) mod 255 x-ul curent.



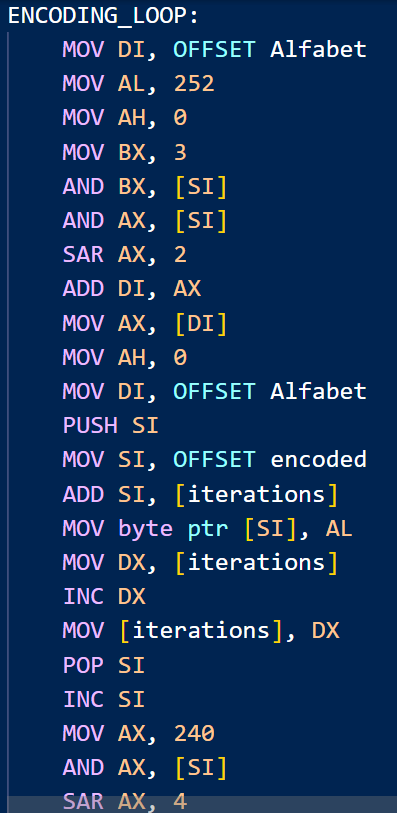
Subrutina RAND realizează operația pe care am precizat-o mai sus(de generare a termenului curent pe baza celui anterior).



Ultima subrutină, “ENCODE” este subrutina cu cele mai multe și complexe operații din întreaga temă. În primă fază, subrutina verifică dacă numărul de biți din șirul encryptat este divizibilă cu 24, iar dacă restul operației DIV este 8 va adăuga un padding de 2 octeți la finalul mesajului codificat, iar în caz contrar, dacă restul este 16(10H), se va da un jump la eticheta care are rolul de a adăuga un padding de 1 octet la finalul codificării COD64.

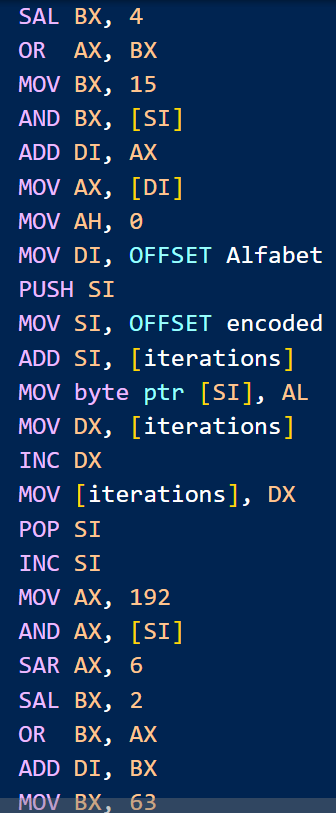


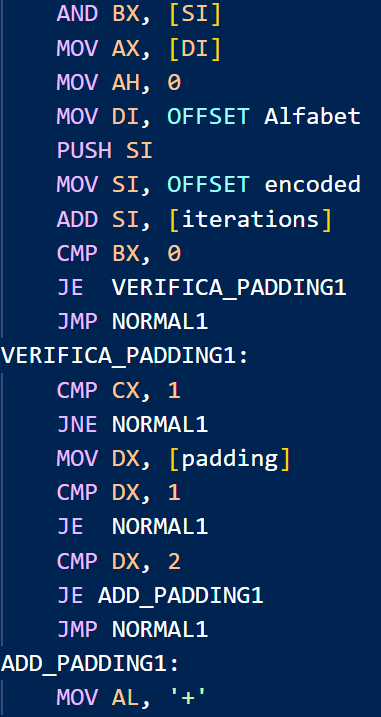
“ADD\_1” și “ADD\_2” atribuie variabilei “padding” valoarea 1, dacă padding-ul necesar este de 1 octet sau 2 dacă este de 2 octeți. În continuare, se trece la eticheta “ENCODING” care va verifica câte iterații e nevoie pentru a codifica șirul encryptat prin împărțirea dimensiunii șirului respectiv cu tot cu padding-ul luat în considerare la 3. O să explic mai jos logica după care am gândit că este nevoie fix de atâtea iterații.



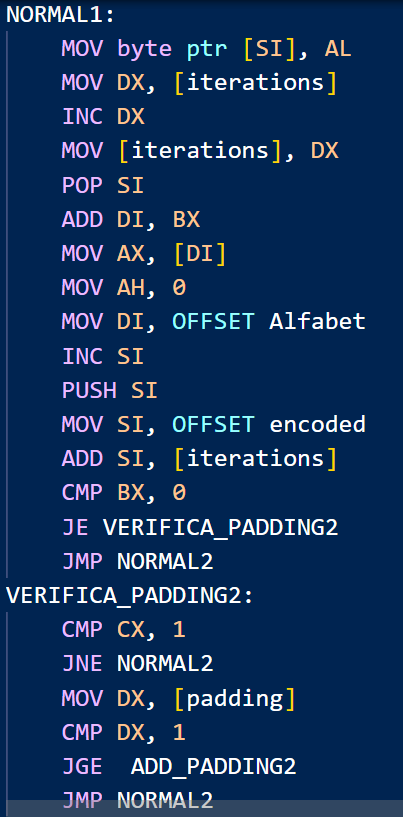
Pentru a explica raționamentul făcut mai sus, am să mă leg de explicațiile furnizate în cerință în ceea ce privește algoritmul de codificare. Am observat că la un bloc de 3 octeți din care vor rezulta 4 caractere COD64 se face următoarele operații: 6, 2+4, 4+2, 6. Pentru a fi mai exact, luăm prima data primii 6 biți din primul octet și apoi indexăm valoarea găsită în alfabetul de codificare pentru a găsit caracterul COD64 aferent(la fel cum vom face la fiecare secvență de 6 biți salvată), apoi salvăm ultimii 2 biți din acel primul octet și primii 4 octeți din cel de-al doilea și realizăm aceeași indexare, apoi ultimii 4 biți din octetul al doilea + primii 2 din octetul al treilea, apoi ultimii 6 biți din octetul al treilea. Pentru a realizare aceste operații am folosit operații de shiftare la stânga și la dreapta, precum și măști cu care pot face acest lucru.

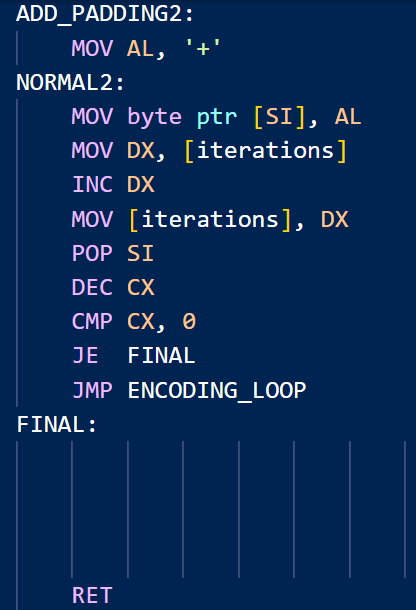
Menționez și faptul că am folosit variabila pusă la dispoziție “iterations” pentru a ține un contor al poziției la care am rămas în șirul “encoded”.





Aici am tratat și cazurile în care este nevoie de padding cu caracterul ‘+’. Pentru a decide când trebuie să codific șirul cu caracterul ‘+’, am verificat secvența de 6 biți întâlnită în momentul când am salvat ultimii 4 biți din cel de-al doilea octet + primii 2 biți din cel de-al treilea octet și secvența de 6 biți formată din ultimii 6 biți din octetul al treilea. Am comparat această secvență cu 0, iar dacă ma aflu la ultima iterație adaug padding, altfel codific normal. Mai concret, dacă e nevoie de un padding de un caracter de ‘+’, atunci compar valoarea din variabila “padding” cu 1, iar dacă e nevoie de un padding de 2 octeți de ‘+’, compar această valoare cu 2.





În final, am decis să utilizez manual un jump și o decrementare pe CX, deoarece cu LOOP nu puteam să mă întorc înapoi din cauza numărului de linii peste care trebuie să se sară.