“Talk is cheap. Show me the code. ― Linus Torvalds”

In continuare va voi prezenta modul de lucru, structura si logica din spatele codului, impreuna cu dificultatile intampinate si solutiile la acestea.

Lt. Drd. Ing. Coca Mihai

Slt. Dragos Ioana

Chiuta Mihai

C112C

**Documentatie - Proiect Menumorut**

Contents

[1. Calculul termenilor a si b 2](#_Toc135936839)

[2. SEED 3](#_Toc135936840)

[3. Encriptare 4](#_Toc135936841)

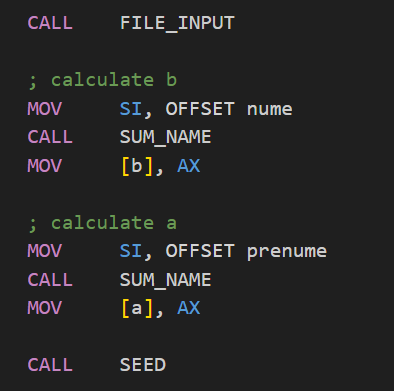
[4. Codificare 5](#_Toc135936842)

# Calculul termenilor a si b

In primul rand, pentru calculul termenilor a si b a fost nevoie de creerea unor 2 noi variabile pentru stocarea numelui, respective al prenumelui:

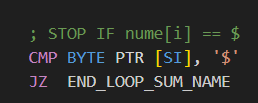


Calculul se intampla inainte de apelarea procedurii SEED in firul principal al programului ( incapsulat in procedura de entry point START )



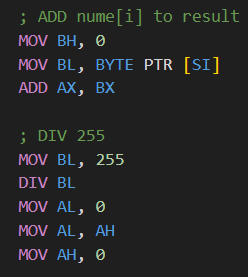
Procedura SUM\_NAME primeste ca parametru offset-ul numelui, respective al prenumelui, in registrul SI si returneaza rezultatul adunarii modulo 255 in AX. Astfel, procedura de fata permite reutilizarea codului atat pentru a cat si pentru b.

Inauntrul functiei SUM\_NAME se itereaza sirul primit in SI pana la intalnirea caracterului ‘$’:



La fiecare iteratie, se aduna in registrul AX caracterul din [SI] si se executa operatia modulo 255 pentru a nu depasi marimea registrului.

\*aici operatia de interes este MOD 255 nu DIV 255, asa cum ar sugera comentariul



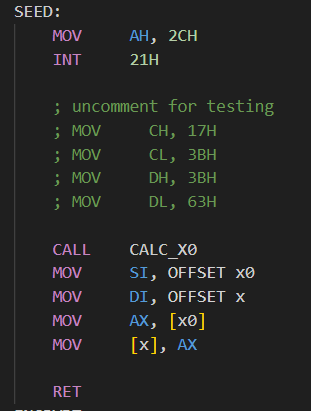
Aceasta tehnica de distribuire a operatiei modulo este un laitmotiv al programului, intrucat a permis efectuarea operatiilor pe un singur registru de 2 octeti, fara complicarea prin alipirea a 2 registrii la nivel logic sau alte opoeratii complicate, asa cum vom vedea in urmatoarele sectiuni.

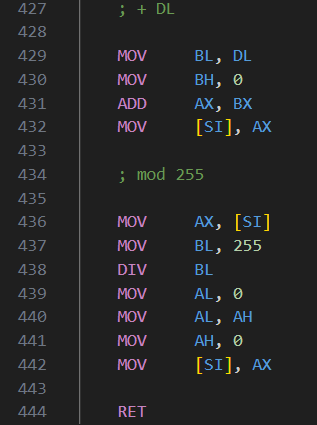
Am realizat de abia la final ca poate ar fi trebuit sa fac o procedura separata si pentru aceasta operatie critica si des intalnita in program.

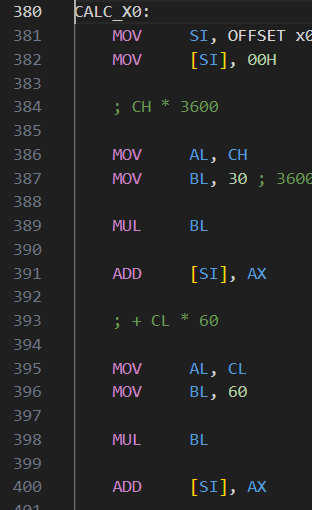
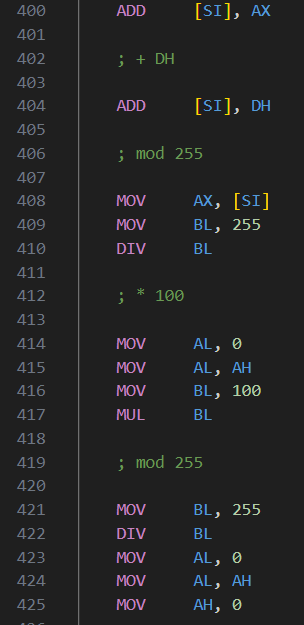
Din pacate, limita de timp nu mi-a permis rescrierea codului astfel incat sa generalizez operatia pentru a o putea scrie intr-o procedura ca mai apoi sa inlocuiesc fiecare ocurenta din cod fara probleme.

# SEED

Functia de seed defapt este un wrapper pentru functia de CALC\_X0, care face toata munca din spate.



In procedura CALC\_X0 avem urmatorul cod:

Dupa cum se vede, aceasta procedura este destul de lunga si simt ca as fi putut imparti in mai multe proceduri codul sau sa mai simplific cumva.

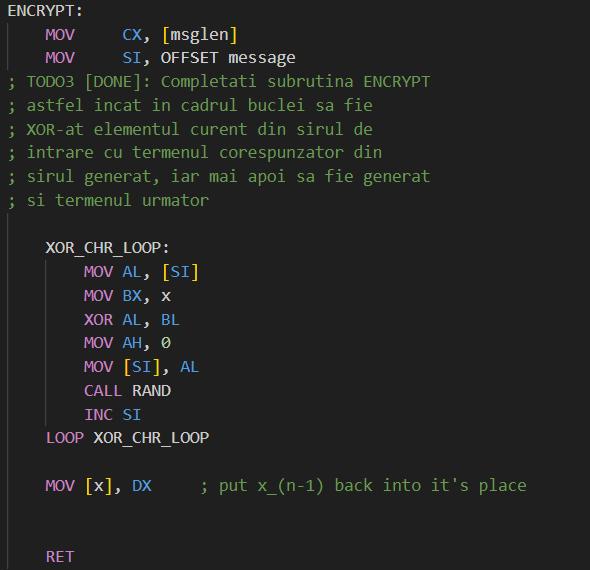
Se poate observa folosirea aceluiasi artificiu de distribuire a mod 255 pentru evitarea depasirii registrului de 2 octeti.

Acest lucru se datoreaza faptului ca numerele raman destul de mici pentru a putea fi reprezentate pe 1 octet, iar prin adunarea lor la nivel de registru pe 2 octeti, nu se mai poate intampla fenomenul de overflow.

De exemplu, 3600 devine prin operatia modulo 255, numarul 30 ( salvat in BL la linia 389 in poza din stanga ).

Rezultatul este salvat in x0, prin intermediul registrului [SI] la linia 442 in poza din partea dreapta.

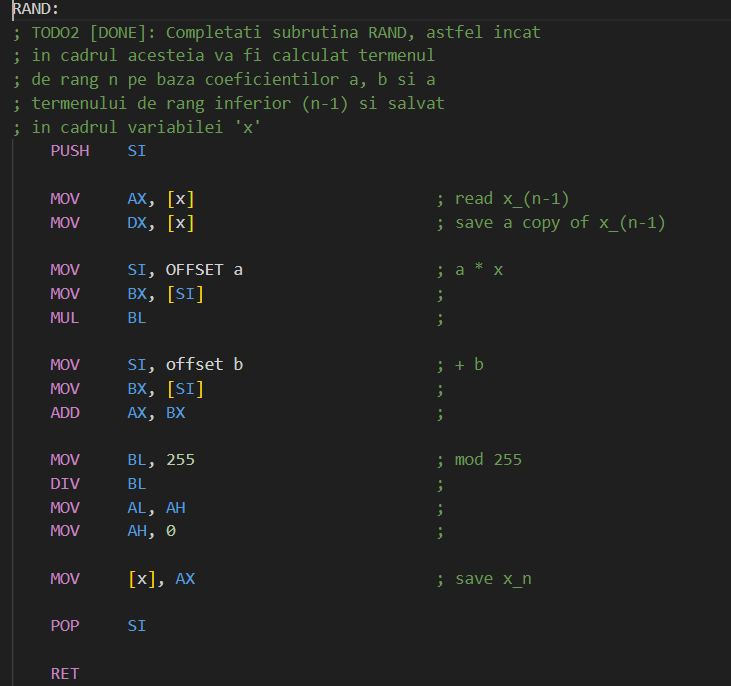
# Encriptare



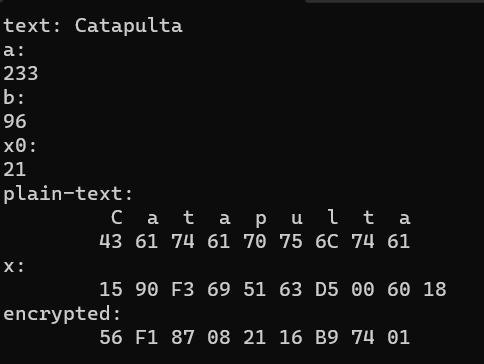
Encriptarea nu a pus multe probleme, insa la rezultat era afisata valoarea xn in loc de xn-1.

Acest lucru se intampla deoarece procedura RAND ( care calculeaza noua valoare a variabilei x pe care o si salveaza ) era apelata la finalul loop-ului, lucru de care nu ma puteam dispensa prea usor.

Astfel, am ales salvarea valorii veche a lui x in registrul DX, ca mai apoi, dupa criptarea textului, sa o mut inapoi in x pentru a fi afisata valoarea corecta ( xn-1 ).



Pentru testarea encriptarii, am facut un program in c++, pe al carei surse am inclus-o in arhiva.



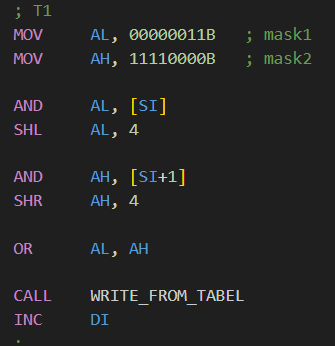
Programul m-a ajutat sa depanez orice problema legata de encriptare intr-un mod efficient. De exemplu, datorita output-ului de la x, am putut sa imi dau seama ca valoarea gresita a variabilei xn-1 era datorata faptului ca afisam defapt valoarea xn, si deci nu era o problema cu procedura RAND.

# Codificare

Codificarea a opus cele mai multe probleme.

Intrucat procedura de codificare este una destul de lunga, dar repetitiva ( as fi putut face o procedura stocata pentru extragerea fiecarui grup de 6 biti ), o voi prezenta in mare.

Fiecare iteratie lucreaza cu cate 3 octeti o data. Pentru extragerea a grupului de 6 biti, a fost nevoie de masti.

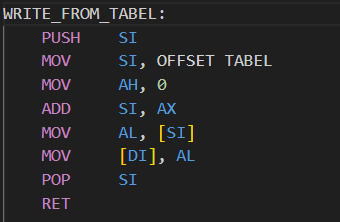


Aici este vorba de al 2-lea grup de 6 biti, care necesita ultimii 2 biti din primul octet ( selectati cu masca 1 ) si primii 4 biti din masca 2 ( selectati cu masca 2 ).

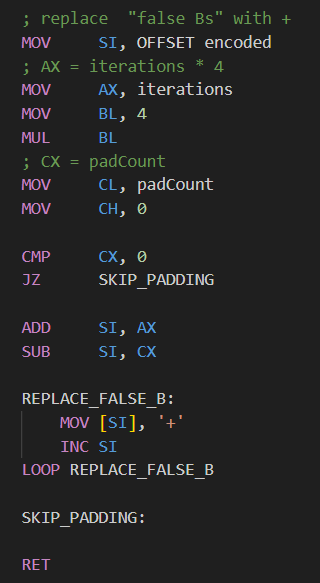
Se face operatia de AND cu primul octet, respectiv al doilea cu mastile aferente pentru captarea bitilor. Acei biti se rearanjeaza in pozitiile lor prin shiftari, apoi rezultatele se suprapun prin operatia OR.

Procedura WRITE\_FROM\_TABLE ia din tabelul encodarii caracterul specific indexului, tabel hardcodat in memorie.



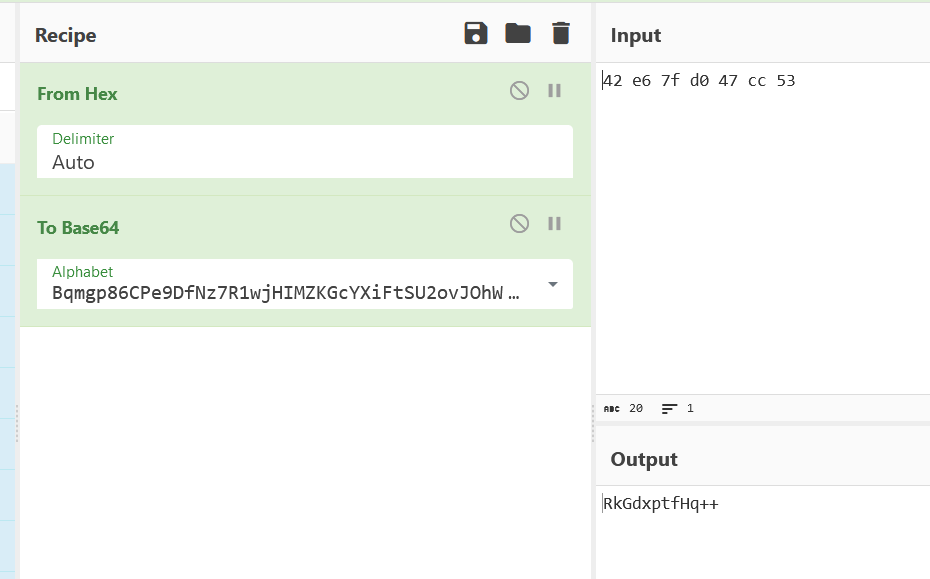


In sfarsit, pentru problema padding-ului m-am gandit sa las programul sa encodeze padding-ul gresit cu ‘B’ in mod intentionat, ca mai apoi sa inlocuiesc B-urile false ( adica ultimele de la final de sir in functie de padCount ), sau sa sara etapa de padding acolo unde era cazul.



Pentru depanarea si verificarea encodarii, am folosit tool-ul CyberChef de care am aflat in cadrul cercului de Cyber, pentru rezolvarea de challenge-uri CTF.

Stiam ca tool-ul poate decodifica din base64, asa ca am incercat sa vad daca nu cumva ma pot folosi de el pentru a encoda si decoda din codul proiectului.



Astfel, am vazut ca pentru decodificarea Base64 se poate introduce un alfabet personalizat, iar eu am introdus alfabetul din tabelul proiectului.

Acest tool mi-a salvat mult timp si s-a dovedit a fi un instrument de verificare precis si de incredere.

## Controlul Versiunii

De asemenea, pentru a nu pierde progress in cazul unei greseli, sau al unui experiment, am ales sa folosest o solutie de control a versiunii ( git ).

