# Academia Tehnică Militară ”Ferdinand I”

# Facultatea de Sisteme Informatice și Securitate Cibernetică



**-Temă Arhitecturi-**

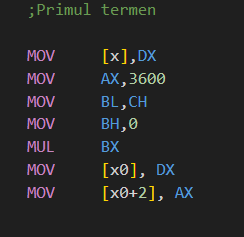
**Sd. Cap. Bordei Alin-Viorel**

**Grupa C112A**

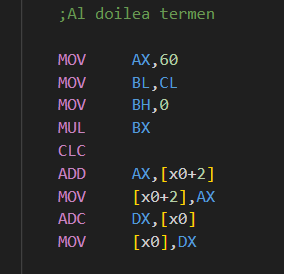
# To Do 1: Subrutina *Seed*

Pentru a calcula *x0*, am împărțit ecuția pe mai mulți pași. Primul pas a fost să calculez CH\*3600. Acest calcul depășește 2 octeți, valoarea maximă ce o poate reține un registru. Rezultatul înmulțirii este dat pe DX:AX. Pentru a reține cei trei octeți rezultați, voi depăși zona definită a lui x0, în [x0+2], lucru care nu oprește rularea programului.

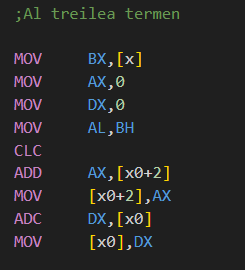
Prima linie reține în memorie, la offsetul x, valoarea din registrul DX, care după întrerupere este populat cu valori ce îmi vor trebui ulterior. Fac această mutare deoarece DX poate fi modificat de MUL, în cazul în care înmulțirea a două numere depășește 2 octeți.



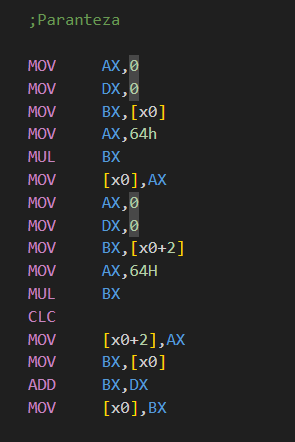
Al doilea pas a fost calularea lui CL\*60 și adaugarea lui sumă. Folosind Mul, calculez CL\*60, luând în considerare posibilitatea depășirii a doi octeți. După înmulțire, mă asigur că flagul de carry este 0 (folosind CLC). Adun in [x0+2] (unde sunt reținuți 2 cei mai nesimnificativi octeți), după care adun in [x0] (unde se află 2 cei mai semnificativi octeți), ținând cont de carry flag-ul modificat la suma anterioară.



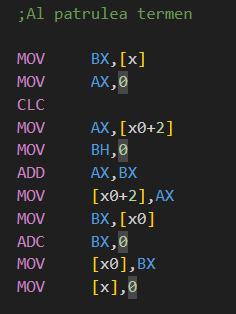
Al treilea pas a fost să aduc din memorie valoarea lui DH după întrerupere și să o adun în memorie, urmărind aceeași logică ca mai pasul anterior.



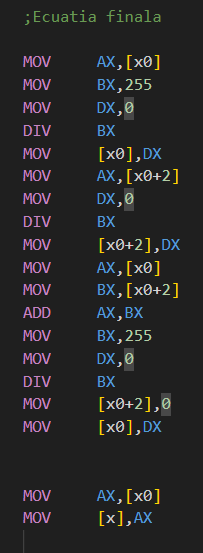
Al patrulea pas a fost sa înmulțesc rezultatul cu 100. Deoarece trebuie să înmmulțesc un număr pe 3 octeți cu 100, voi înmulțesc întâi 2 cei mai semnificativi octeți cu 100, adun rezultatul în [x0], înmulțesc 2 cei mai nesemnificativi octeți cu 100, adun rezultatul în [x0+2], apoi adun DX la [x0]. DX a fost făcut 0 înainte de ultima înmulțire, astfel acționează similar cu ADC [x0],0



Al cincilea pas a fost sa adaug valoarea din DL de după întrerupere.



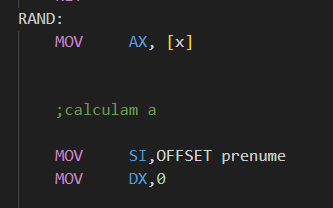
Ultimul pas a fost să împart rezultatul la 255. Pentru a face acest lucru am împățit [x0] la 255, [x0+2] la 255, după care am adunat rezultatele și am mai împărțit odată la 255. Restul împărțirii l-am luat din registrul DX, după ce m-am asigurat ca acesta este 0.



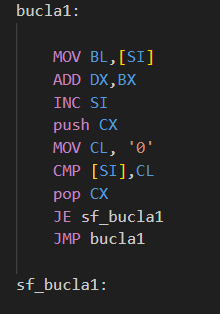
La final populez zonele din memorie cu rezultatele aferente

# To do 2: Subrutina *Rand*

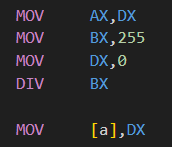
Încep prin a inițializa regiștrii pe care am să îi folosesc.



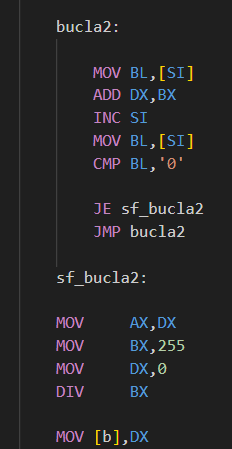
Urmează calculul valorilor ASCII a prenumelui, folosindu-mă de o buclă:



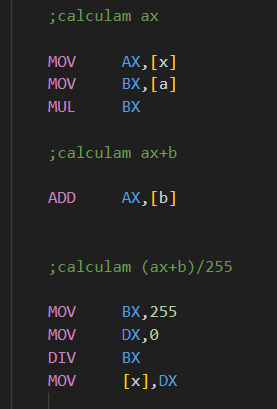
După calculez [a] după formula dată în cerință:



Urmez aceeași logică pentru a calcula [b]:



Următorul pas este de a calcula după formula dată în cerință, executând câteva calcule simple și actualizez [x]:

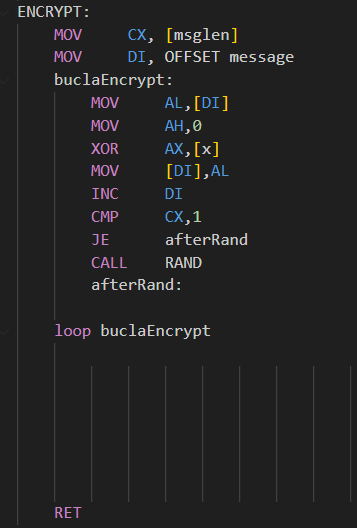


# To do 3: Subrutina *Encrypt*

Creez un loop, inițializând CX cu [msglen], deoarece trebuie să fac operația XOR pe fiecare literă a cuvântului dat. Inițializez DI cu OFFSET message deoarece trebuie să parcurg cuvântul dat.

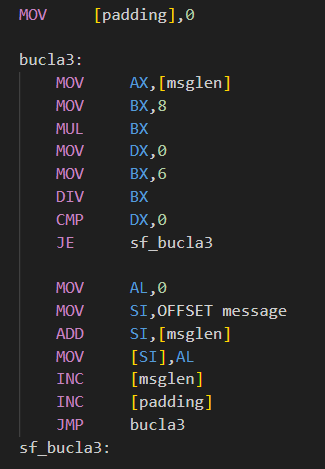
În buclă, mut [DI] în AL deoarece îmi doresc să extrag octet cu octet, după care mă asigur că AH este 0 (deși cel mai probabil este deja, prefer o abordare mai precaută).

După XOR, am CMP CX,1 cu scopul de a păstra ultimul Xn.

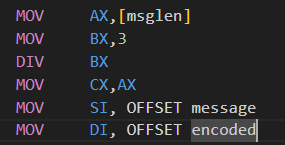


# To do 4: Subrutina *Encode*

Pentru început, fac padding-ul pentru cuvânt. Acest lucru îl realizez prin înmulțirea dimensiunii cuvântului ([msglen]) cu 8, pentru a afla numărul de biți, după care împart la 6 (dimensiunea calupului de biți). Dacă această împărțire NU are rest, ies din loop, altfel adaug un octet null și cresc dimensiunea lui [msglen] cu 1.

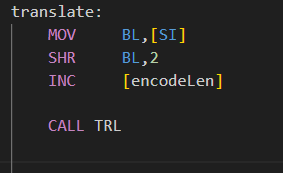


Urmează inițializarea datelor ce vor fi folosite în loop-ul pentru codare. Pe lângă cele evidente, inițializez CX cu [msglen]/3 deoarece voi trata câte 3 octeți odată în codare. Pot fi sigur că această împărțire nu va avea rest, deoarece dimensiunea cuvântului + dimensiunea paddingului va fi multiplu de 3.



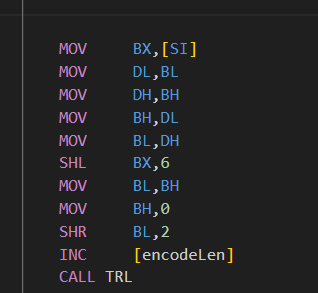
Primul calup de 6 biți îl creez prin șiftarea la dreapta cu 2 biți

(10101010) ---> (00101010). După care apelez o subrutină creeată de mine numită TRL.



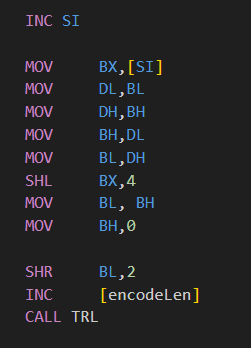
Pentru al doilea calup, aduc din memorie 2 octeți, schimb BH cu BL și vice versa, pentru a putea face șiftările corespunzător.

(10101010 01010101) ---> (10010101 01000000) ---> (10010101 ) ---> (00100101)



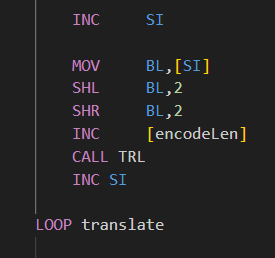
Pentru al treilea calup, urmez un proces similar cu cel de al doilea calup:

(10101010 01010101) ---> (10100101 01010000) ---> (10100101) ---> (00101001)



Al patrulea și ultimul calup, asemenea primului, aduc un singur octet din memorie:

(01010101) ---> (01010100) ---> (00010101)

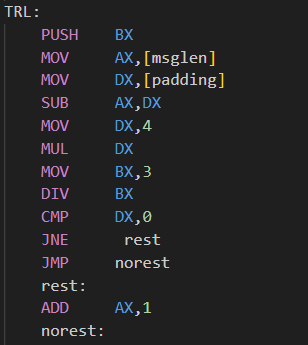


# Subrutina *TRL*

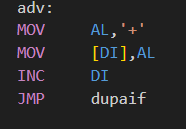
Subrutină creeată din proprie inițiativă, pentru a ajuta la codificarea cuvântului. Rolul acesteia este de a căuta în stringul *CODE64*, caracterul corespunzător și să îl pună în stringul *encoded*



Primul pas este de a vedea dacă am ajuns în zona de padding. În caz afirmativ, voi completa cu ”+” în encoded, în caz contrar voi lua elementul corespuzător din *CODE64*



Caz afirmativ:



Caz negativ:

