

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 ecuația pe mai mulți pași. Primul pas a fost să calculez $CH \cdot 3600$. 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.

```
;Primul termen

MOV     [x],DX
MOV     AX,3600
MOV     BL,CH
MOV     BH,0
MUL     BX
MOV     [x0], DX
MOV     [x0+2], AX
```

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

```
;Al doilea termen

MOV     AX,60
MOV     BL,CL
MOV     BH,0
MUL     BX
CLC
ADD     AX,[x0+2]
MOV     [x0+2],AX
ADC     DX,[x0]
MOV     [x0],DX
```

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 treilea termen
```

```
MOV     BX,[x]
MOV     AX,0
MOV     DX,0
MOV     AL,BH
CLC
ADD     AX,[x0+2]
MOV     [x0+2],AX
ADC     DX,[x0]
MOV     [x0],DX
```

Al patrulea pas a fost să înmulțesc rezultatul cu 100. Deoarece trebuie să înmulțesc 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

```
;Paranteza
```

```
MOV     AX,0
MOV     DX,0
MOV     BX,[x0]
MOV     AX,64h
MUL     BX
MOV     [x0],AX
MOV     AX,0
MOV     DX,0
MOV     BX,[x0+2]
MOV     AX,64H
MUL     BX
CLC
MOV     [x0+2],AX
MOV     BX,[x0]
ADD     BX,DX
MOV     [x0],BX
```

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

```
;Al patrulea termen  
  
MOV     BX,[x]  
MOV     AX,0  
CLC  
MOV     AX,[x0+2]  
MOV     BH,0  
ADD     AX,BX  
MOV     [x0+2],AX  
MOV     BX,[x0]  
ADC     BX,0  
MOV     [x0],BX  
MOV     [x],0
```

Ultimul pas a fost să împart rezultatul la 255. Pentru a face acest lucru am împărțit [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.

```

;Ecuatia finala

MOV     AX,[x0]
MOV     BX,255
MOV     DX,0
DIV     BX
MOV     [x0],DX
MOV     AX,[x0+2]
MOV     DX,0
DIV     BX
MOV     [x0+2],DX
MOV     AX,[x0]
MOV     BX,[x0+2]
ADD     AX,BX
MOV     BX,255
MOV     DX,0
DIV     BX
MOV     [x0+2],0
MOV     [x0],DX

MOV     AX,[x0]
MOV     [x],AX

```

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.

```
RAND:
    MOV     AX, [x]

    ;calculam a

    MOV     SI,OFFSET prenume
    MOV     DX,0
```

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

```
buc1a1:

    MOV BL,[SI]
    ADD DX,BX
    INC SI
    push CX
    MOV CL, '0'
    CMP [SI],CL
    pop CX
    JE sf_buc1a1
    JMP buc1a1

sf_buc1a1:
```

După calculez [a] după formula dată în cerință:

```
MOV     AX,DX
MOV     BX,255
MOV     DX,0
DIV     BX

MOV     [a],DX
```

Urmez aceeași logică pentru a calcula [b]:

```
buc1a2:
    MOV BL,[SI]
    ADD DX,BX
    INC SI
    MOV BL,[SI]
    CMP BL,'0'

    JE sf_buc1a2
    JMP buc1a2

sf_buc1a2:

    MOV     AX,DX
    MOV     BX,255
    MOV     DX,0
    DIV     BX

    MOV [b],DX
```

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

```
;calculam ax

MOV     AX,[x]
MOV     BX,[a]
MUL     BX

;calculam ax+b

ADD     AX,[b]

;calculam (ax+b)/255

MOV     BX,255
MOV     DX,0
DIV     BX
MOV     [x],DX
```

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.

```
ENCRYPT:
    MOV     CX, [msglen]
    MOV     DI, OFFSET message
    buclaEncrypt:
        MOV     AL, [DI]
        MOV     AH, 0
        XOR     AX, [x]
        MOV     [DI], AL
        INC     DI
        CMP     CX, 1
        JE     afterRand
        CALL    RAND
    afterRand:
        loop   buclaEncrypt

    RET
```


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.

```
MOV     [padding],0

buc1a3:
    MOV     AX,[msglen]
    MOV     BX,8
    MUL     BX
    MOV     DX,0
    MOV     BX,6
    DIV     BX
    CMP     DX,0
    JE      sf_buc1a3

    MOV     AL,0
    MOV     SI,OFFSET message
    ADD     SI,[msglen]
    MOV     [SI],AL
    INC     [msglen]
    INC     [padding]
    JMP     buc1a3

sf_buc1a3:
```

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.

```
MOV     AX,[msglen]
MOV     BX,3
DIV     BX
MOV     CX,AX
MOV     SI, OFFSET message
MOV     DI, OFFSET encoded
```

Primul calup de 6 biți îl creez prin șiftarea la dreapta cu 2 biți
(10101010) ---> (00101010). După care apelez o subrutină creată de mine numită
TRL.

```
translate:
    MOV     BL,[SI]
    SHR     BL,2
    INC     [encodeLen]

    CALL    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)

```
    MOV     BX,[SI]
    MOV     DL,BL
    MOV     DH,BH
    MOV     BH,DL
    MOV     BL,DH
    SHL     BX,6
    MOV     BL,BH
    MOV     BH,0
    SHR     BL,2
    INC     [encodeLen]
    CALL    TRL
```

Pentru al treilea calup, urmez un proces similar cu cel de al doilea calup:
(10101010 01010101) ---> (10100101 01010000) ---> (10100101) ---> (00101001)

```
    INC     SI

    MOV     BX,[SI]
    MOV     DL,BL
    MOV     DH,BH
    MOV     BH,DL
    MOV     BL,DH
    SHL     BX,4
    MOV     BL,BH
    MOV     BH,0

    SHR     BL,2
    INC     [encodeLen]
    CALL    TRL
```

Al patrulea și ultimul calup, asemenea primului, aduc un singur octet din memorie:
(01010101) ---> (01010100) ---> (00010101)

```
INC     SI

MOV     BL,[SI]
SHL     BL,2
SHR     BL,2
INC     [encodeLen]
CALL    TRL
INC     SI

LOOP    translate
```

Subrutina *TRL*

Subrutină creată 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*

```
CODE64    DB  'Bqmgp86CPe9DfNz7R1wjHIMZKGcYXiFtSU2ovJ0hW4ly5EkrqsnAxubTV03a=L/d'
```

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 corespunzător din *CODE64*

```
TRL:
PUSH    BX
MOV     AX,[msglen]
MOV     DX,[padding]
SUB     AX,DX
MOV     DX,4
MUL     DX
MOV     BX,3
DIV     BX
CMP     DX,0
JNE     rest
JMP     norest
rest:
ADD     AX,1
norest:
```

Caz afirmativ:

```
adv:
MOV     AL, '+'
MOV     [DI], AL
INC     DI
JMP     dupaif
```

Caz negativ:

```
fals:
PUSH    DI
MOV     DI, OFFSET CODE64
ADD     DI, BX
MOV     AL, [DI]
POP     DI
MOV     [DI], AL
INC     DI
JMP     dupaif

dupaif:
POP     BX
RET
```