****România

Ministerul Apărării Naționale

Academia Tehnică Militară “Ferdinand I” București

Facultatea de Sisteme Informatice și Securitate Cibernetică

**TEMĂ SEMESTRIALĂ**

**ARHITECTURA SISTEMELOR DE CALCUL**

**DOCUMENTAȚIE PRIVIND IMPLEMENTAREA TEMEI**

Grad, nume și prenume: sd. cap. Andrei-Sebastian DODOȚ

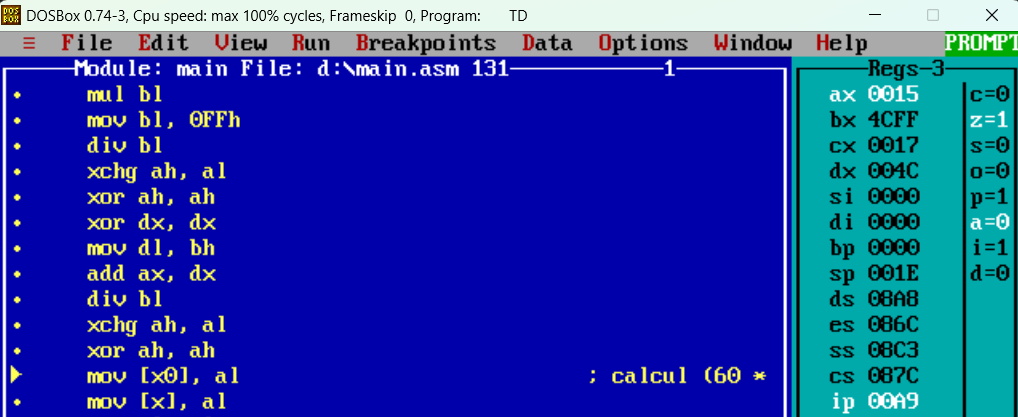
Grupa de studii: C112E

Profesori coordonatori: lt. drd. ing. Mihai COCA, slt. ing. Dragoș IOANA

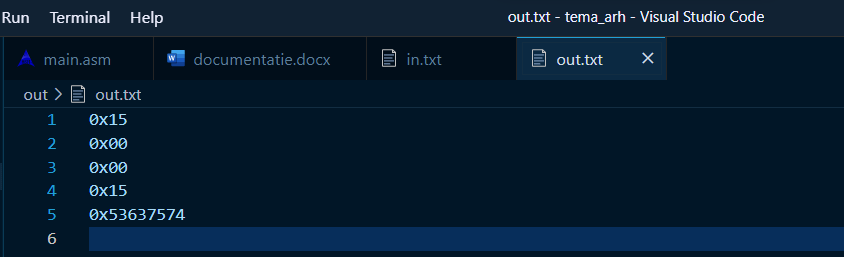
În cadrul procesului de implementare a temei s-a reușit implementarea și testarea cu succes a tuturor celor 4 cerințe, cu următoarele specificații pentru fiecare sub-problemă:

*TODO1: Completati subrutina SEED astfel incat la final sa fie salvat in variabila 'x' si 'x0' continutul termenului initial*

Pentru evitarea overflow-ului am ales să efectuez operația mod % 255 (rezultatul final nefiind afectat, conform teoremei congruenței modulo n) după fiecare operație din cadrul calculului termenului , așa cum se poate vedea în următorul exemplu particularizat pentru valorile menționate ale registrelor ch (0eh), cl (17h), dh (26h), dl (4CH):



Am folosit registrele AX, BX, și DX pentru fiecare pas din calculul, precum și instrucțiunile MUL, DIV (pentru operația mod), XOR pentru „curățarea” registrelor, XCHG pentru a obține o imagine mai bună a rezultatelor la fiecare iterație.

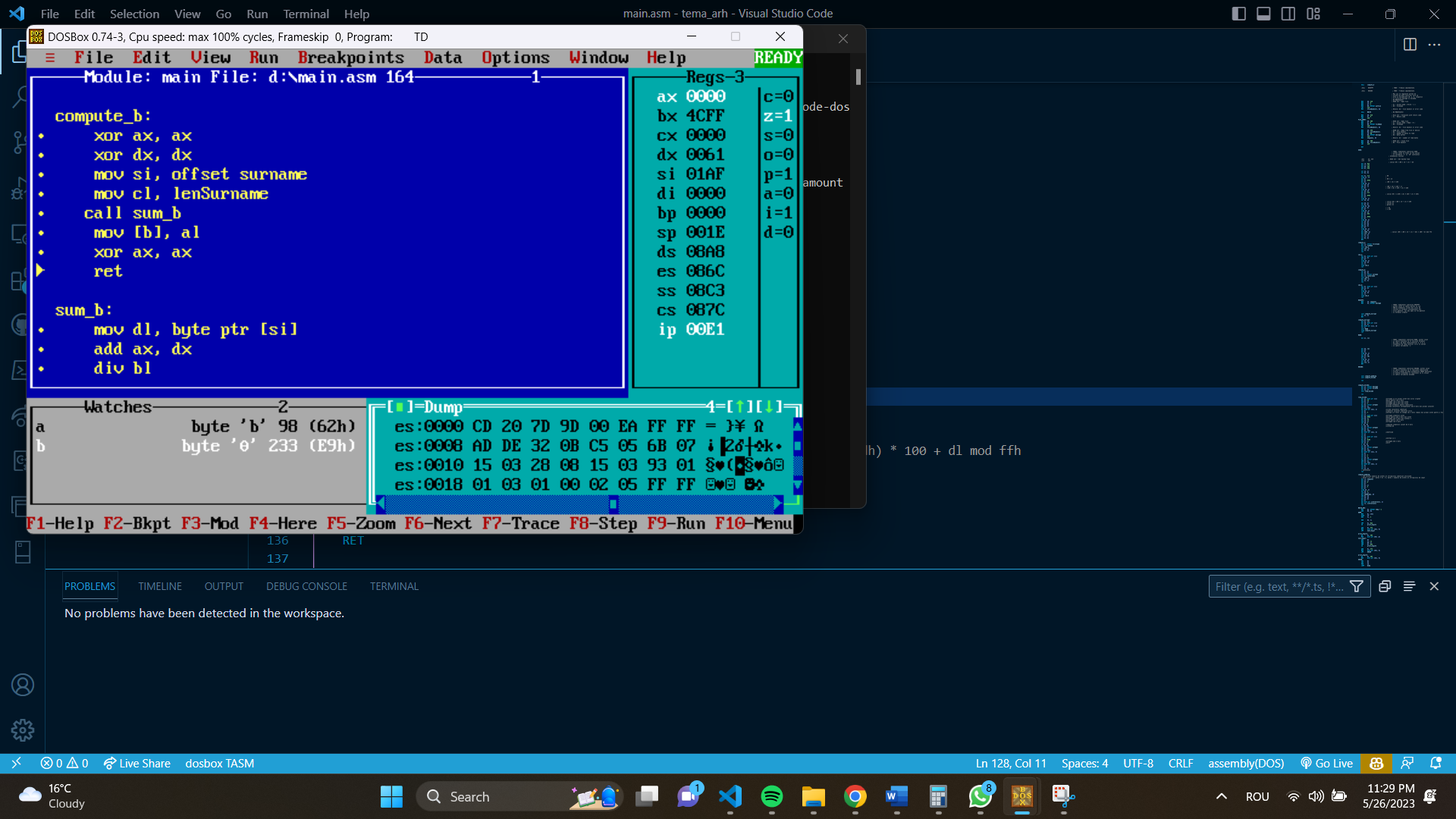
În fișierul out.txt, rezultatul este următorul:

*TODO2: Completati subrutina RAND, astfel incat in cadrul acesteia va fi calculat termenul de rang n pe baza coeficientilor a, b si a termenului de rang inferior (n-1) si salvat in cadrul variabilei 'x'*

Pentru a calcula cu succes termenul este nevoie de calculul valorilor *a* și *b.* Așadar, programul este construit astfel încât procedeul de criptare să funcționeze pentru orice nume și prenume am introduce. Am implementat 2 subrutine, *compute\_a* (calculul valorii *a* conform enunțului), respectiv *compute\_b* (calculul valorii *a* conform enunțului). Totodată, după fiecare iterație am efectuat operația modulo 255 pentru siguranța evitării overflow-ului. Pentru efectuarea sumei a fost nevoie de câte o buclă repetitivă.

În cele din urmă, subrutina RAND calculează termenul la fiecare iterație *i,* generând termenul , folosit în procedeul de criptare al string-ului de input. Rezultatul se scrie în variabila *x*.

Pentru exemplul din enunț, am obținut următorul rezultat:



*TODO3: Completati subrutina ENCRYPT astfel incat in cadrul buclei sa fie XOR - at elementul curent din sirul de intrare cu termenul corespunzator din sirul generat, iar mai apoi sa fie generat si termenul urmator*

Pentru procedeul de criptare am implementat o subrutină *compute\_encrypt* care folosește registrul SI pentru parcurgerea octet cu octet al string-ului *message* și efectează respectiv operația XOR cu termenul generat de subrutina RAND.

ENCRYPT:

    MOV     CX, [msglen]

    MOV     SI, OFFSET message

    mov ax, [x0]

    call compute\_encrypt

    ret

compute\_encrypt:

    mov [x], ax

    mov ax, [x]

    mov ah, byte ptr [si]

    xor ah, al

    mov byte ptr [si], ah

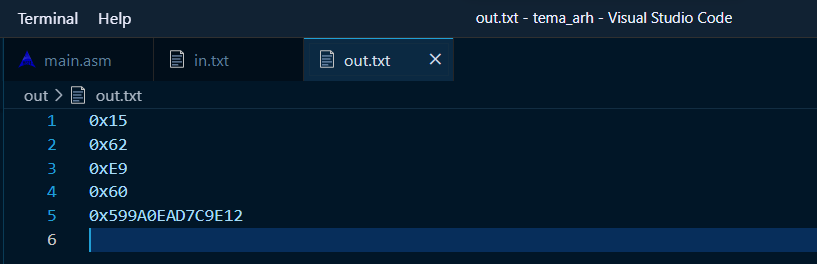
    inc si

    call RAND

    loop compute\_encrypt

    ret

Pornind de la exemplul din enunț, rezultatul după 3 iterații al termenului este :



De asemenea, rezultatul operației de criptare este:

*TODO4: Completati subrutina ENCODE, astfel incat in cadrul acesteia va fi realizata codificarea sirului criptat pe baza alfabetului COD64 mentionat in enuntul problemei si rezultatul va fi stocat in cadrul variabilei encoded*

Pentru implementarea codificării a fost nevoie de inițializarea alfabetului de codificare, prin variabila *alfabet*:

alfabet     DB  'Bqmgp86CPe9DfNz7R1wjHIMZKGcYXiFtSU2ovJOhW4ly5EkrqsnAxubTV03a=L/d'

Pentru a calcula numărul de octeți al string-ului codificat (*output\_length*) am ajuns, prin subrutina *compute\_padding*, la următoarele rezultate:

*iterations = numărul grupărilor a câte 3 octeți*

*padding = numărul octeților necesari pentru a alcătui încă o grupare de 3 octeți (caracterul ‘+’)*

Subrutina *compute\_encode* folosește registrele SI și DI pentru a referi adresele de memorie ale șirurilor de caractere *message* și *encoded*, zonă de memorie nul-inițializată la momentul respectiv.

Mai departe, subrutina *loop\_encode* din cadrul *compute\_encode* iterează câte 3 octeți pe care îi regrupează în cuvinte a câte 6 biți, realizându-se operația de codificare conform *alfabetului*. (codificare base64).

Gruparea se realizeză prin incărcarea secvențială în registrul AX, prin intermediul registrului SI, a câtor 3 octeți asupra cărora de realizează operația AND cu masca potrivită pentru extragerea celor 6 biți necesari. Astfel, identificăm 4 măști, încărcate în fiecare caz (case1, case2, case3, case4) în variabila WORD \_word: 0fc00h, 03f0h, 0fc0h, 3fh.

Subrutina *write\_encoded* efectuează scrierea în zona de memorie alocată string-ului encoded, a caracterului corespunzător din alfabet pentru fiecare grupare.

Instrucțiunea jnz add\_padding din următorul code-snippet al subrutinei *compute\_encode*:

    cmp [padding], 0

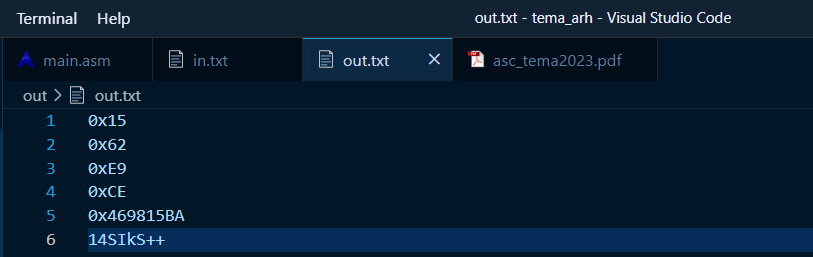
    jnz add\_padding

este un jump condiționat către subrutina de adăugare a padding-ului, după caz.

Subrutina *add\_padding* continuă codificarea caracterelor rămase, numărul lor fiind 3 - [padding].

Subrutina *complete\_padding* efectuează adăugarea caracterului ‘+’ în număr de [padding], pentru a completa ultima grupare de 3 octeți.

Rezultatul final, în urma execuției programului, asupra exemplului din cerință, este următorul:



Rezultatul final, în urma execuției programului, asupra exemplului din testul 3, este următorul:

