Tema 1 laborator ASC

Descrierea soluției:

Pentru citirea datelor de intrare, am folosit procedurile **citire_numar**, **citire_mesaj_clar** si **citire_mesaj_criptat**.

Urmează să verificăm dacă numărul **p** din cerință este prim cu ajutorul procedurii **verif_prim**. În procedura, ne folosim de registrul **$t0** pentru a parcurge toți potențialii divizori de la 2 până la $\left[\frac{p}{2}\right]$ și verificăm dacă am găsit un divizor prin efectuarea restului împărțirii lui **p** la **$t0**. Dacă restul este 0, atunci am găsit un divizor al lui p și încărcăm valoarea 1 în registrul **$t2**. În final, dacă registrul **$t2** are valoarea 0, înseamnă că nu am găsit niciun divisor al lui **p**, deci numărul este prim și procedura întoarce valoarea 1, iar în caz contrar, întoarce valoarea 0.

Dacă procedura a întors valoarea 0, atunci apelăm procedura **exit_nonprim** care afișează un mesaj și încheie execuția programului. În schimb, dacă numărul este prim, este garantat faptul că grupul $\left(\mathbb{Z}_p^*, \cdot\right)$ este ciclic, deci poate fi generat de un singur element.

Astfel, apelăm procedura **cauta_generator**, care determină și afișează valoarea generatorului, și, de asemenea, va stoca în vectorul **v**, reținut la nivel de memorie, valorile generate în ordine de către generator. În procedură, ne folosim de registrul **$s1** pentru a parcurge fiecare număr **g** de la 2 la p-1 și a genera toate puterile până când dăm de elementul neutru(1). După generarea puterilor, verificăm mai întâi dacă ordinul lui g, adică numărul puterilor generate până la elementul neutru, este mai mic decât ordinul grupului(p-1), caz în care incrementăm **g** cu 1 și efectuăm din nou

generarea puterilor pentru noul candidat. Dacă ord(g)=p-1, atunci este posibil că g să fie generator, însă trebuie să ne asigurăm că au fost generate toate elementele grupului. Pentru a verifica acest lucru, apelăm procedura **verif_generator** care va cauta secvențial fiecare element x din intervalul [2,p-1] în vectorul **v**. Marcăm apariția lui x în vectorul **v** cu ajutorul registrului **$t3**, iar în cazul în care după fiecare dintre căutări valoarea regsitrului **$t3** este 1, înseamnă că în vectorul **v** au fost generate, întradevăr, toate elementele grupului, deci afișăm **g** și ne întoarcem în programul principal.

Urmează acum să criptam mesajul clar citit, așa că apelăm procedura **criptare_mesaj_clar**. Acum, pentru a cripta întreg mesajul, vom cripta fiecare litera, pe rând, în felul următor: În registrul **$t3** reținem numărul **i** corespunzător literei, scăzând din codul caracterului codul ASCII al primei litere din alfabet, adică 65. Astfel, pentru litera A avem numărul corespunzător 0, pentru B->1, C->2 etc. Conform izomorfismului, îi asociem numărului **i** puterea $\boldsymbol{g^i}$, unde g este generatorul, iar $\boldsymbol{g^i}$ este chiar elementul de pe poziția **i** din vector(în registrul $t4 reținem **4*i**, adică locația din memorie relativă la **v** a poziției **i** din vector, iar în registrul $t5 încărcăm v[i]). Pentru a obține litera corespunzătoare numărului, adică litera criptată, adunăm cu 65 și stocam caracterul la adresa mesaj_clar_criptat din memorie. Urmând procedeul pentru fiecare litera, în final obținem întreg mesajul criptat la adresa mesaj_clar_criptat, mesaj pe care îl afișăm și apoi ne întoarcem în programul principal.

Pentru a decripta al doilea mesaj introdus de la tastatură, vom apela procedura **decriptare_mesaj**, care urmează un procedeu asemănator celui pentru criptare, numai că de data aceasta numărul corespunzător unei litere din mesajul sursă este g^i, iar numărul asociat acestuia este exponentul **i**.

Pentru a determina exponentul **i**, parcurgem vectorul **v** pentru a afla poziția la care se găsește g^i, după care, pentru a obține litera decriptată corespunzătoare, adunăm cu 65 și stocăm caracterul la adresa mesaj_decriptat din memorie. După decriptarea întregului mesaj, îl afișăm și ne întoarcem în programul principal, unde urmează să oprim execuția programului.

Codul sursă:

```
 .data
    mesaj_clar: .space 512
    mesaj_clar_criptat: .space 512
    mesaj_criptat: .space 512
    mesaj_decriptat: .space 512
    v: .space 512
    prompt_afisare_mesaj_criptat: .asciiz " Mesajul clar criptat este: "
    prompt_afisare_mesaj_decriptat: .asciiz " Mesajul decriptat este: "
    prompt_afisare_generator: .asciiz "Generatorul este: "
    prompt_nonprim: .asciiz "Numarul introdus nu este prim!"
    nl: .asciiz "\n"
.text

main:

    jal citire_numar # apelam procedura de citire a unui numar de la
tastatura
    move $s0, $v0    # incarcam numarul citit (p) in registrul $s0
    jal citire_mesaj_clar    # citim mesajul clar
    jal citire_mesaj_criptat # citim mesajul criptat
```

```
    subu $sp, 4      # alocam spatiu in stiva
    sw $s0, 0($sp)   # incarcam p in stiva
    jal verif_prim   # apelam procedura care intoarce val. 1 daca p este
prim, 0 in caz contrar
    addu $sp, 4      # dezalocam spatiul alocat in stiva

    beqz $v0, exit_nonprim  # daca p nu este prim, atunci apelam procedura
 exit_nonprim pentru a afisa un mesaj


    subu $sp, 4           # alocam spatiu in stiva
    sw $s0, 0($sp)        # incarcam p in stiva
    jal cauta_generator   # apelam procedura care afiseaza   valoarea
generatorului si genereaza puterile in vectorul v
    addu $sp, 4           # dezalocam spatiul alocat in stiva

    jal criptare_mesaj_clar    # apelam procedura care cripteaza mesajul
clar
    jal decriptare_mesaj       # apelam procedura care decripteaza mesajul
 criptat

    j exit # incheiem programul

citire_numar:

    li $v0, 5       # incarcam codul 5 pt a citi un intreg
    syscall         # apelam sistemul
    jr $ra          # ne intoarcem in programul principal

citire_mesaj_clar:


    la $a0, mesaj_clar   # incarcam adresa locatiei unde mesajul va fi
stocat
    li $a1, 511          # lungimea maxima
    li $v0, 8            # incarcam codul 4 pentru afisarea unui string

    syscall              # apelam sistem
    jr $ra                # ne intoarcem in programul principal
```

```
citire_mesaj_criptat:

    la $a0, mesaj_criptat   # incarcam adresa locatiei unde mesajul va fi
stocat
    li $a1, 511          # lungimea maxima
    li $v0, 8            # incarcam codul 4 pentru afisarea unui string

    syscall              # apelam sistem
    jr $ra               # ne intoarcem in programul principal

verif_prim:
    subu $sp, 4      # $sp:()(p)
    sw $fp, 0($sp)   # $sp:($fp v)(p)
    addi $fp, $sp, 4 # $sp:($fp v)$fp:(p)
    subu $sp, 4      # $sp:()($fp v)$fp:(p)
    sw $s0, 0($sp)   # $sp:($s0)($fp v)$fp:(p)
    subu $sp, 4      # $sp:()($s0)($fp v)$fp:(p)
    sw $s1, 0($sp)   # $sp:($s1)($s0)($fp v)$fp:(p)

    lw $s0, 0($fp)  # incarcam in registrul $s0 numarul p
    div $s1, $s0, 2 # in registrul $s1 retinem [p/2]
    li $t0, 2       # registrul $t0 un potential divizor d
    li $t2, 0      # registrul $t2 are valoarea 1 daca am gasit un divizor
 al numarului citit, sau valoarea 0 in caz contrar

loop_prim:

    bgt $t0, $s1, finish_loop_prim  # vom cauta divizorii lui p numai pana
 la [p/2]
    beq $t2, 1, finish_loop_prim    # daca am gasit un divizor al lui p,
atunci nu mai are rost sa continuam

    rem $t1, $s0, $t0               # in registrul $t1 retinem restul
impartirii lui p la d
    seq $t2, $t1, 0                 # daca restul impartirii e 0, atunci
registrul $t2 va avea valoarea 1

    addu $t0, 1                     # incrementam $t0 cu 1
    j loop_prim                     # repeat
```

```
finish_loop_prim:

    lw $s1, -12($fp)    # restauram $s1
    lw $s0, -8($fp)     # restauram $s0
    lw $fp, -4($fp)     # restauram $fp
    addu $sp, 12        # facem pop de 3 ori
    seq $v0, $t2, 0     # incarcam in $v0 valoarea 1 daca $t2 are valoarea
 0(adica este prim), altfel, $v0 ia valoarea 0
    jr $ra  # ne intoarcem la adresa din $ra

#-------------------------------------------------------------------------
-----------------------------------------------

cauta_generator:

    subu $sp, 4      # $sp:()(p)
    sw $fp, 0($sp)   # $sp:($fp v)(p)
    addi $fp, $sp, 4 # $sp:($fp v)$fp:(p)

    subu $sp, 4      # $sp:()($fp v)$fp:(p)
    sw $ra, 0($sp)   # $sp:($ra)($fp v)$fp:(p)
    subu $sp, 4      # $sp:()($ra)($fp v)$fp:(p)
    sw $s0, 0($sp)   # $sp:($s0)($ra)($fp v)$fp:(p)
    subu $sp, 4      # $sp:()($s0)($ra)($fp v)$fp:(p)
    sw $s1, 0($sp)   # $sp:($s1)($s0)($ra)($fp v)$fp:(p)

    lw $s0, 0($fp)   # incarcam in registrul $s0 numarul p

    li $s1, 2        # $s1 este g ( nu are rost sa verficiam daca 1 este
generator)

loop_generator:

    beq $s1, $s0, finish_loop_gen # cautam generatorul in   intervalul
[2,p-1]
    li $t1, 1    # $t1 este k
    li $t2, 1    # $t2 este g_pas_anterior, intial el este g^0 adica 1
    sw $t2, v($0) # v[0]=g^0
    li $t4, 4    # registrul $t4 este un pointer catre pozitia k in vector
```

```
loop_gen_puteri:

    beq $t1, $s0, finish_loop_gen_puteri
                        # $t3 va retine g^k mod p,
    mul $t3, $t2, $s1  # $t3 = g * g_pas_anterior
    rem $t3, $t3, $s0  # $t3 = $t3 mod p

    sw $t3, v($t4) # v[k]=g^k mod p

    beq $t3, 1, finish_loop_gen_puteri # daca am obtinut 1(elementul
neutru), nu mai are sens verificarea

    addu $t1, 1        # incrementam k cu 1
    addu $t4, 4        # incrementam pointerul cu 4
    move $t2, $t3      # g^k mod p devine g_pas_anterior
    j loop_gen_puteri

finish_loop_gen_puteri:

   # dupa parcurgere, k va fi ordinul lui g
   subu $t0, $s0, 1 # $t0=p-1
   bne $t1, $t0, next_gen
# daca ord(g)!=p-1, atunci mai cautam un generator

   subu $sp, 4         # alocam spatiu in stiva
   sw $s0, 0($sp)      # incarcam p in stiva
   jal verif_generator # apelam procedura care intoarce valoarea 1 daca g
a generat toate elementele grupului Zp*, 0 in caz contrar
   addu $sp, 4         # dezalocam spatiu

   beq $v0, 1, finish_loop_gen # daca s-
au generat toate elementele grupului, atunci g este generator si il afisam

next_gen:

   addu $s1, 1 # continuam sa cautam generatorul
   j loop_generator
```

```
finish_loop_gen:

    la $a0, prompt_afisare_generator   # incarcam adresa prompt-ului
    li $v0, 4                          # incarcam codul 4 pentru afisarea
unui string
    syscall                            # apelam sistemul

    move $a0, $s1  # incarcam generatorul g in registrul $a0 pentru
a-l afisa
    li $v0, 1      # incarcam codul 1 pentru afisarea unui intreg
    syscall        # apelam sistemul

    la $a0, nl  # incarcam adresa caracterului newline
    li $v0, 4   # incarcam codul 4 pentru afisarea unui string
    syscall     # apelam sistemul


    lw $s1, -16($fp)    # restauram $s1
    lw $s0, -12($fp)    # restauram $s0
    lw $ra, -8($fp)     # restauram $ra
    lw $fp, -4($fp)     # restauram $fp
    addu $sp, 16        # facem pop de 4 ori
    jr $ra              # ne intoarcem in programul principal


verif_generator:

    subu $sp, 4      # $sp:()(p)
    sw $fp, 0($sp)   # $sp:($fp v)(p)
    addu $fp, $sp, 4 # $sp:($fp v)$fp:(p)
    subu $sp, 4      # $sp:()($fp v)$fp:(p)
    sw $s0, 0($sp)   # $sp:($s0)($fp v)$fp:(p)

    lw $s0, 0($fp)   # incarcam p in $s0


    li $t0, 2  # cu ajutorul registrului $t0 verificam daca fiecare numar
din intervalul [2,p-1] apare in vector
```

```
loop_verif_gen:

    beq $t0, $s0, exit_verif_gen # daca am verificat fiecare numar din
interval, verificam daca am gasit generatorul
    li $t1, 1 # $t1 e un contor(i) pentru pozitiile din vector. Acesta inc
epe de la  pozitia 1, deoarce v[0]=1, pt orice p, asa ca nu are rost sa in
cepem de la 0
    li $t2, 4 # $t2 e pointer catre o pozitie(i) din vector
    li $t3, 0 # $t3 va retine valoarea 1 daca daca numarul $t0 se afla in
vector

loop2_verif_gen:

    beq $t1, $s0, exit_loop2 # parcurgem vectorul de la 1 la p-1
    beq $t3, 1, exit_loop2   # daca am gasit deja numarul $t0 in vector, o
prim cautarea

    lw  $t4, v($t2)     # incarcam in registrul $t4 v[i]
    seq $t3, $t4, $t0   # incarcam valoarea 1 in registrul $t3  daca
v[i]==$t0

    addu $t1, 1         # incrementam i cu 1
    addu $t2, 4         # incrementam pointerul cu 4
    j loop2_verif_gen

exit_loop2:

    addu $t0, 1 # cautam urmatorul numar din interval in vector

    beq $t3, 1, loop_verif_gen # daca $t3 e unu, cautam urmatorul numar
din interval in vector

    li $v0, 0           # daca nu am gasit numarul din interval in vector
( $t3 este 0), intoarcem valoarea 0

    lw $s0, -8($fp)     # restauram $s0
    lw $fp, -4($fp)     # restauram $fp
    addu $sp, 8         # facem pop de 2 ori
    jr $ra
```

```
exit_verif_gen:

    move $v0, $t3       # daca si ultimul numar din interval a fost gasit
in vector, inseamna ca tot grupul a fost generat. Astfel, $v0 ia valoarea
1, iar in caz contrar $v0 va retine 0
    lw $s0, -8($fp)     # restauram $s0
    lw $fp, -4($fp)     # restauram $fp
    addu $sp, 8         # facem pop de 2 ori
    jr $ra

#---------------------------------------------------------------------
-----------------------------------------------

criptare_mesaj_clar:

    la $t0, mesaj_clar # pointer catre adresa lui mesaj_clar
    la $t1, mesaj_clar_criptat#pointer catre adresa lui mesaj_clar_criptat
    lb $t2, 0($t0)   # $t2 retine caracterul  la adresa $t0 din mesaj_clar

loop_criptare:

    beq $t2, 0xA, finish_loop_criptare # parcurgem mesajul clar pana la
intalnirea caracterului '\n'
    subu $t3, $t2, 65  # in registrul $t3 retinem numarul corespunzator
fiecarei litere ( A -> 0, B-> 1, ....)
    mul $t4, $t3, 4    # $t4 retine locatia din memorie a pozitiei $t3 din
 vector

    lw $t5, v($t4)  # in registrul $t5 incarcam g la puterea $t3, valoare
aflata la locatia v($t4), adica v[$t3]
    addu $t5, 65    # adunam cu 65 pentru  a obtine codul ASCII al literei
 corespunzatoare
    sb $t5, 0($t1)  # stocam caracterul in mesaj_clar_criptat

    addu $t0, 1     #  incrementam adresa lui mesaj_clar pentru a cripta
urmatorul caracter din mesaj
    addu $t1, 1     #  incrementam adresa lui mesaj_clar_criptat pentru a
stoca urmatorul caracter criptat
    lb $t2, 0($t0)  #  incarcam in $t2 caracterul actual
    j loop_criptare #  repeat
```

```
finish_loop_criptare:

    sb $0, 0($t1)  # stocam caracterul null la finalul mesajului criptat

    la $a0, prompt_afisare_mesaj_criptat  # incarcam adresa prompt-ului
    li $v0, 4                            # incarcam codul 4 pentru afisarea
 unui string
    syscall                                # apelam sistemul

    la $a0, mesaj_clar_criptat       # incarcam adresa mesajului decriptat
    li $v0, 4              # incarcam codul 4 pentru afisarea unui string
    syscall                             # apelam sistemul

    la $a0, nl  # incarcam adresa caracterului newline
    li $v0, 4   # incarcam codul 4 pentru afisarea unui string
    syscall     # apelam sistemul

    jr $ra

#-------------------------------------------------------------------------
-----------------------------------------------
decriptare_mesaj:

    la $t0, mesaj_criptat # pointer catre adresa lui mesaj_criptat
    la $t1, mesaj_decriptat # pointer catre adresa lui mesaj_decriptat
    lb $t2, 0($t0)     # $t2 retine caracterul  la adresa $t0 din mesaj_cr
iptat

loop_decriptare:

    beqz $t2, finish_loop_decriptare # parcurgem mesajul criptat pana la
intalnirea caracterului nul
    subu $t3, $t2, 65  # in registrul $t3 retinem numarul corespunzator
fiecarei litere ( A -> 0, B-> 1, ....)
    li $t4, 0      # $t4 contorizeaza pozitiile (i) din vectorul v
    la $t5, 0      # $t5 retine locatia in memorie relativa la v
    lw $t7, v($t5) # valoarea lui v intr-o pozitie
```

```
loop_vector_gen_2:

    beq $t7, $t3, finish_loop_vector_gen_2 # parcurgem vectorul de puteri
ale generatorului pana cand intalnim o putere a generatorului egala cu $t3

    addu $t4, 1 # incrementam
    addu $t5, 4 # incrementam
    lw $t7, v($t5) # incarcam in $t7 urmatoarea valoare din vector
    j loop_vector_gen_2 # repeat

finish_loop_vector_gen_2:

    move $t6, $t4   # in registrul $t6 incarcam exponentul puterii
    addu $t6, 65    # adunam cu 65 pentru a obtine codul ASCII al literei
corespunzatoare
    sb $t6, 0($t1)  # stocam caracterul in mesaj_decriptat

    addu $t0, 1     #  incrementam adresa lui mesaj_criptat pentru a
decripta urmatorul caracter din mesaj
    addu $t1, 1     #  incrementam adresa lui mesaj_decriptat pentru a
stoca urmatorul caracter decriptat
    lb $t2, 0($t0)  #  incarcam in $t2 caracterul actual
    j loop_decriptare #  repeat

finish_loop_decriptare:

    sb $0, 0($t1)  # stocam caracterul null la finalul mesajului decriptat

    la $a0, prompt_afisare_mesaj_decriptat  # incarcam adresa prompt-ului
    li $v0, 4                               # incarcam codul 4 pentru
afisarea unui string
    syscall                                 # apelam sistemul

    la $a0, mesaj_decriptat # incarcam adresa mesajului decriptat
    li $v0, 4               # incarcam codul 4 pentru afisarea unui string
    syscall                 # apelam sistemul
    jr $ra                  # ne intoarcem in programul principal
```

```
#-------------------------------------------------------------------------
-----------------------------------------------

exit_nonprim:

    la $a0, prompt_nonprim  # incarcam adresa prompt-ului
    li $v0, 4               # incarcam codul 4 pentru afisarea unui string
    syscall                 # apelam sistemul
    j exit                  # apelam procedura care termina programul

exit:

    li $v0, 10              # incarcam codul 10 pentru oprirea programului
    syscall                 # apelam sistemul
```

Exemple de date de intrare si de iesire:

1)

Input

```
7
ACAD
BCBG
```

Output

```
Generatorul este: 3
 Mesajul clar criptat este: BCBG
 Mesajul decriptat este: ACAD
```

2)

Input

```
21
AEFD
BBBC
```

Output

```
Numarul introdus nu este prim!
```

3)

Input

```
23
SALUTEUSUNTMIPS
GBWMHSQTG
```

Output

```
Generatorul este: 5
 Mesajul clar criptat este: GBWMHEMGMVHSQTG
 Mesajul decriptat este: SALUTMIPS
```