Descriere Tema MIPS

Programul creat este structurat in mai multe functii, care utilizand stack-uri pastreaza valorile registrilor utilizati. La nivelul memoriei am salvat valoarea lui p, puterile generatorului (powc), puterile vizitate (vispow), mesajul clar (decodeds), mesajul criptat (encodeds) si alte varibile de tip asciiz ce contin mesaje ce vor fi afisate pe ecran.

Din main se apeleaza in primul rand functia de citire a lui p, iar dupa citire se verifica primalitatea lui p, utilizand functia prime ce are o complexitate O(p), parcurgand numerele de la 2 la sqrt(p).

Dupa verificarea lui p se cauta un generator ce indeplineste conditiile descrise in cerinta prin functia determineg de complexitate O(p^2), care salveaza intr-un vector de aparitii numerele utilizate si puterile generatorului (powc). Dupa determinarea generatorului se afiseaza pe ecran powc(1), respectiv generatorul la puterea 1.

Urmeaza functia de citire a mesajului clar cat si functia de criptare a lui (encode) de complexitate O(p), care transforma fiecare litera intr-un numar, preia valoarea de pe pozitia puterii generatorului (powc), transforma rezultatul in ascii si il afiseaza.

In final, se citeste mesajul criptat si se decripteaza, utilizand functia decode de complexitate O(p^2), care transforma literele in numere, cautandu-le in vectorul de puteri ale generatorului, converteste rezultatul inapoi in ascii si il afiseaza pe ecran.

.data

p:.word 0

powc:.space 104 # 26 \* 4

vispow:.space 104 # 26 \* 4

decodeds:.space 100 # normal input to be encoded

encodeds:.space 100 # encoded input to be decoded

aa:.asciiz "A"

sp:.asciiz " "

endl:.asciiz "\n"

not\_prime\_msg:.asciiz " is not a prime number, choose another value for p"

error\_log\_msg:.asciiz "Error: "

working:.asciiz "\n\nRegisters work properly!"

generator\_msg:.asciiz "The value of the generator g is: "

.text

# read p function

readp:

subu $sp, 4 # $sp:()

sw $v0, 0($sp) # $sp:($v0)

li $v0, 5

syscall

sw $v0, p

lw $v0, 0($sp) # restore $v0

addu $sp, 4 # clear $sp

jr $ra

# prime function $sp:(x) -> $a0

prime:

subu $sp, 4 # $sp:()(x)

sw $fp, 0($sp) # $sp:($fp v)(x)

addi $fp, $sp, 4 # $sp:($fp v)$fp:(x)

subu $sp, 4 # $sp:()($fp v)$fp:(x)

sw $t0, 0($sp) # $sp:($t0 v)($fp v)$fp:(x)

subu $sp, 4 # $sp:()($t0 v)($fp v)$fp:(x)

sw $t1, 0($sp) # $sp:($t1 v)($t0 v)($fp v)$fp:(x)

subu $sp, 4 # $sp:()($t1 v)($t0 v)($fp v)$fp:(x)

sw $s0, 0($sp) # $sp:($s0 v)($t1 v)($t0 v)($fp v)$fp:(x)

subu $sp, 4 # $sp:()($s0 v)($t1 v)($t0 v)($fp v)$fp:(x)

sw $t2, 0($sp) # $sp:($t2)($s0 v)($t1 v)($t0 v)($fp v)$fp:(x)

lw $t0, 0($fp) # $t0 = x

li $t2, 2 # $t2 = 2 (div)

li $a0, 1 # $a0 = 1 (true)

blt $t0, 2, endprimebad

whileprime:

mul $t1, $t2, $t2 # $t2 ^ 2

bgt $t1, $t0, endprime # finish prime function when $t2 ^ 2 > $t0 (x)

rem $s0, $t0, $t2 # $s0 = $t0 % $t2

beqz $s0, endprimebad # finish prime function with negative result

add $t2, $t2, 1 # $t2++

j whileprime

endprimebad:

li $a0, 0 # $a0 = 0 (false)

endprime:

lw $t2, 0($sp) # restore $t2

lw $s0, 4($sp) # restore $s0

lw $t1, 8($sp) # restore $t1

lw $t0, 12($sp) # restore $t0

lw $fp, 16($sp) # restore $fp

addu $sp, 24 # clear $sp

jr $ra

# check prime function

checkprime:

subu $sp, 8 # $sp:()()

sw $ra, 0($sp) # $sp:($ra v)()

sw $a0, 4($sp) # $sp:($ra v)($a0)

lw $t0, p

subu $sp, 4 # $sp:()($ra v)

sw $t0, 0($sp) # $sp:($t0)($ra v)

jal prime # call prime: $sp:($t0)($ra v) -> $a0

beqz $a0, exitnotprime

lw $ra, 0($sp) # restore $ra

lw $a0, 4($sp) # restore $a0

addu $sp, 8 # clear $sp

jr $ra

# determine the minimum generator g and store it's powers in (powc)

determineg:

subu $sp, 36 # $sp:()()()()()() size = 7 \* 4

sw $t0, 0($sp) # $sp:($t0)

sw $t1, 4($sp) # $sp:($t0)($t1)

sw $t2, 8($sp) # $sp:($t0)($t1)($t2)

sw $t3, 12($sp) # $sp:($t0)($t1)($t2)($t3)

sw $s0, 16($sp) # $sp:($t0)($t1)($t2)($t3)($s0)

sw $s1, 20($sp) # $sp:($t0)($t1)($t2)($t3)($s0)($s1)

sw $s2, 24($sp) # $sp:($t0)($t1)($t2)($t3)($s0)($s1)($s2)

sw $s3, 28($sp) # $sp:($t0)($t1)($t2)($t3)($s0)($s1)($s3)

sw $s4, 32($sp) # $sp:($t0)($t1)($t2)($t3)($s0)($s1)($s4)

li $s0, 1 # $s0 -> g

lw $s2, p # $s2 -> p

lw $s4, p

sub $s4, $s4, 1 # $s4 -> p-1

loopdg:

beq $s0, $s2, enddetermineg

add $s0, $s0, 1 # $s0++

li $s1, 1 # $s1 -> g\*\*0

li $t0, 0 # $t0 -> visual i for powc

li $t1, 0 # $t1 -> real i for powc

sw $s1, powc($t1)

move $t2, $t0 # $t2 -> visual j for vispow

move $t3, $t1 # $t3 -> real j for vispow

loopclearvis: # clear visited

sw $0, vispow($t3)

add $t2, $t2, 1

add $t3, $t3, 4

blt $t2, $s4, loopclearvis

loopdgp: # check powers, and verify if they are visited

beq $t0, $s4, enddetermineg

mul $t3, $s1, 4

lw $t2, vispow($t3)

beq $t2, 1, loopdg

li $t2, 1

sw $t2, vispow($t3)

mul $s1, $s1, $s0

rem $s1, $s1, $s2

add $t0, $t0, 1

add $t1, $t1, 4

sw $s1, powc($t1)

j loopdgp

j loopdg

enddetermineg:

lw $t0, 0($sp) # restore $t0

lw $t1, 4($sp) # restore $t1

lw $t2, 8($sp) # restore $t2

lw $t3, 12($sp) # restore $t3

lw $s0, 16($sp) # restore $s0

lw $s1, 20($sp) # restore $s1

lw $s2, 24($sp) # restore $s2

lw $s3, 28($sp) # restore $s3

lw $s4, 32($sp) # restore $s4

addu $sp, 36 # clear $sp

jr $ra

# just showing the powers of the generator

showgenerator:

subu $sp, 12 # $sp:()()() size = 4 \* 3

sw $t3, 0($sp) # $sp:($t3)

sw $v0, 4($sp) # $sp:($t3)($v0)

sw $a0, 8($sp) # $sp:($t3)($v0)($a0)

la $a0, generator\_msg

li $v0, 4

syscall

li $t3, 4

lw $a0, powc($t3)

li $v0, 1

syscall

la $a0, endl

li $v0, 4

syscall

lw $t3, 0($sp) # restore $t3

lw $v0, 4($sp) # restore $v0

lw $a0, 8($sp) # restore $a0

addu $sp, 12 # clear $sp

jr $ra

# read clear string

readdecodeds:

subu $sp, 12 # 4 \* 3

sw $a0, 0($sp) # $sp:($a0)

sw $a1, 4($sp) # $sp:($a0)($a1)

sw $v0, 8($sp) # $sp:($a0)($a1)($v0)

la $a0, decodeds

li $a1, 101

li $v0, 8

syscall

lw $a0, 0($sp) # restore $a0

lw $a1, 4($sp) # restore $a1

lw $v0, 8($sp) # restore $v0

addu $sp, 12 # clear $sp

jr $ra

# read encoded string - line by line

readencodeds:

subu $sp, 12 # 4 \* 3

sw $a0, 0($sp) # $sp:($a0)

sw $a1, 4($sp) # $sp:($a0)($a1)

sw $v0, 8($sp) # $sp:($a0)($a1)($v0)

la $a0, encodeds

li $a1, 101

li $v0, 8

syscall

lw $a0, 0($sp) # restore $a0

lw $a1, 4($sp) # restore $a1

lw $v0, 8($sp) # restore $v0

addu $sp, 12 # clear $sp

jr $ra

# function for encoding the string

encode:

subu $sp, 24 # 4 \* 6

sw $t0, 0($sp) # $sp:($t0)

sw $t1, 4($sp) # $sp:($t0)($t1)

sw $t2, 8($sp) # $sp:($t0)($t1)($t2)

sw $s2, 12($sp) # $sp:($t0)($t1)($t2)($s2)

sw $a0, 16($sp) # $sp:($t0)($t1)($t2)($s2)($a0)

sw $v0, 20($sp) # $sp:($t0)($t1)($t2)($s2)($a0)($v0)

li $t0, 0

lw $s2, p

whileencode:

lb $t1, decodeds($t0)

sub $t2, $t1, 'A'

blt $t2, $0, endencode

rem $t2, $t2, $s2

mul $t2, $t2, 4

lw $t2, powc($t2)

li $t1, 'A'

add $t1, $t1, $t2

li $v0, 11

move $a0, $t1

syscall

add $t0, $t0, 1

lb $t1, decodeds($t0)

bne $t1, 0, whileencode

endencode:

la $a0, endl

li $v0, 4

syscall

lw $t0, 0($sp) # restore $t0

lw $t1, 4($sp) # restore $t1

lw $t2, 8($sp) # restore $t2

lw $s2, 12($sp) # restore $s2

lw $a0, 16($sp) # restore $a0

lw $v0, 20($sp) # restore $v0

addu $sp, 24 # clear $sp

jr $ra

# function for decoding the string

decode:

subu $sp, 36 # 4 \* 9

sw $t0, 0($sp) # $sp:($t0)

sw $t1, 4($sp) # $sp:($t0)($t1)

sw $t3, 8($sp) # $sp:($t0)($t1)($t3)

sw $t4, 12($sp) # $sp:($t0)($t1)($t3)($t4)

sw $t5, 16($sp) # $sp:($t0)($t1)($t3)($t4)($t5)

sw $t6, 20($sp) # $sp:($t0)($t1)($t3)($t4)($t5)($t6)

sw $s2, 24($sp) # $sp:($t0)($t1)($t3)($t4)($t5)($t6)($s2)

sw $v0, 28($sp) # $sp:($t0)($t1)($t3)($t4)($t5)($t6)($s2)($v0)

sw $a0, 32($sp) # $sp:($t0)($t1)($t3)($t4)($t5)($t6)($s2)($v0)($a0)

li $t0, 0

lw $s2, p

whiledecode:

lb $t1, encodeds($t0) # load $t1 as character

beqz $t1, enddecode # end if $t1 = 0

sub $t1, $t1, 'A' # $t1 = int($t1)

blt $t1, $0, enddecode

rem $t1, $t1, $s2 # $t1 mod p

li $t5, 0 # $t5 -> visual i for powc

li $t6, 0 # $t6 -> real i for powc

lw $t3, p # $t3 = p

whilepows: # find power

lw $t4, powc($t6)

beq $t4, $t1, endwhilepows

add $t5, $t5, 1

add $t6, $t6, 4

blt $t5, $t3, whilepows

endwhilepows:

lb $t4, aa

add $t4, $t4, $t5

move $a0, $t4

li $v0, 11

syscall

add $t0, $t0, 1

j whiledecode

enddecode:

lw $t0, 0($sp) # restore $t0

lw $t1, 4($sp) # restore $t1

lw $t3, 8($sp) # restore $t3

lw $t4, 12($sp) # restore $t4

lw $t5, 16($sp) # restore $t5

lw $t6, 20($sp) # restore $t6

lw $s2, 24($sp) # restore $s2

lw $v0, 28($sp) # restore $v0

lw $a0, 32($sp) # restore $a0

addu $sp, 36 # clear $sp

jr $ra

main:

#################################################

# #

# Created by Darius Buhai @ all rights reserved #

# #

#################################################

## la $a0, working

## li $v0, 4

jal readp

jal checkprime

jal determineg

jal showgenerator

jal readdecodeds

jal encode

jal readencodeds

jal decode

## syscall

j exit

errorlog:

li $v0, 4

la $a0, error\_log\_msg

syscall

jr $ra

exitnotprime:

jal errorlog

li $v0, 1

lw $a0, p

syscall

li $v0, 4

la $a0, not\_prime\_msg

syscall

exit:

li $v0, 10

syscall

Exemple de rezultate:

input:

7

ABCD

BDCG

output:

The value of the generator g is: 3

BDCG

ABCD

input:

8

BBB

AAA

output:

Error: 8 is not a prime number, choose another value for p

input:

23

ABCDEFGHIJKL

ALPHAB

output:

The value of the generator g is: 5

BFCKEUIRQLJW

XJRTXA