COMPUTER ARCHITECTURE - RISC V PROJECT

BY ALBERTO CHARABATI 7031859

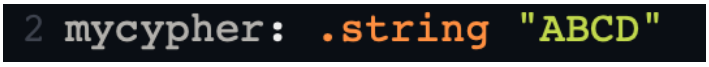
ALBERTO.CHARABATI@STUD.UNIFI.IT

TABLE OF CONTENTS:

OBJECTIVES OF THE PROGRAM

Cyphering is a secret or a disguised way of writing and transferring messages. The program made has the objective of encrypting or cyphering messages (Strings) through different kinds of algorithms, as well as deciphering them using the inverse function of each of the above.

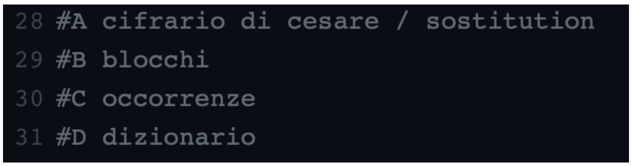
This is possible by implementing in the program 4 differed algorithms that can all be used separately or together as a whole. Thus, by using an input string (mycypher) which, commands the program which algorithm we wish to use in order to obtain the cyphering desired.



```
mycypher: .string "ABCD"
```

VERSATILITY

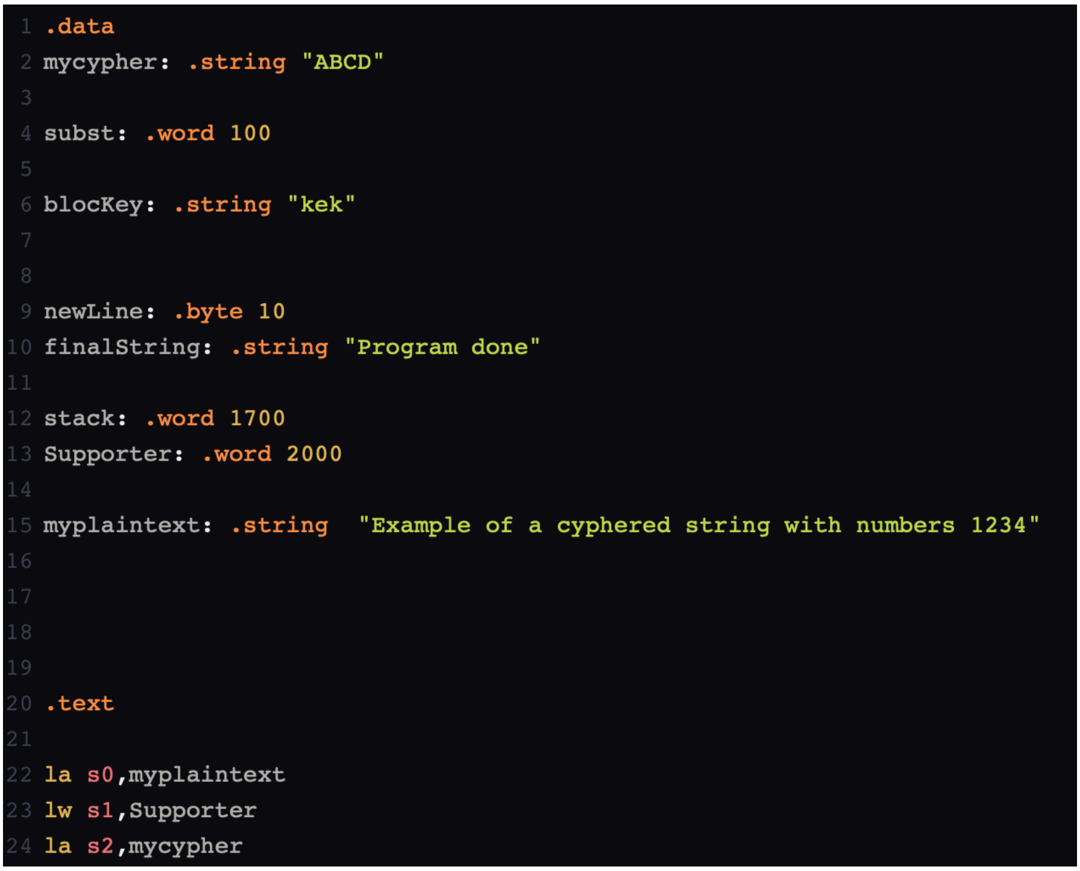
As mentioned above, the program implements 4 different algorithms which can be used as the user desires.
The algorithms used are named with the key letters “A”, “B”, “C” and “D”. And in the next paragraphs I will describe each of the algorithms represented by each of this input letters.
The program is designed to be as versatile as possible, by giving the user the option to use one or more of the algorithms implemented in the program, this by using a key String names as mycypher. The input given to the string will command which encryption the program will use. Thus meaning that, for instance, by entering the input “AC”, the program will implement the algorithms A and C.



```
#A cifrario di cesare / sostitution
#B blocchi
#C occorrenze
#D dizionario
```

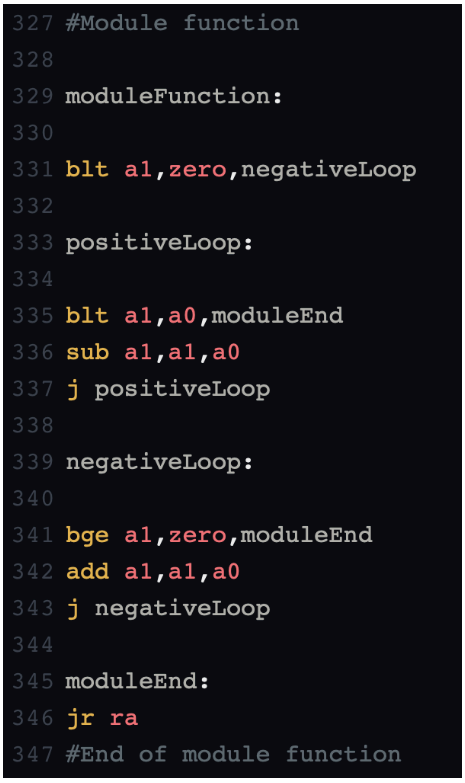
Additionally, the program requires the use of keys and support variables for the algorithms which require so.

Each of the algorithms is made to work in-spot, with the only exception of Occorrenze, which needs to modify it’s length. Furthermore, there are in the program functions that are not “purely” RISC V functions, and these are explained below.



```
.data
mycypher: .string "ABCD"

subst: .word 100

blocKey: .string "kek"


newLine: .byte 10
finalString: .string "Program done"

stack: .word 1700
Supporter: .word 2000

myplaintext: .string  "Example of a cyphered string with numbers 1234"




.text

la s0,myplaintext
lw s1,Supporter
la s2,mycypher
```

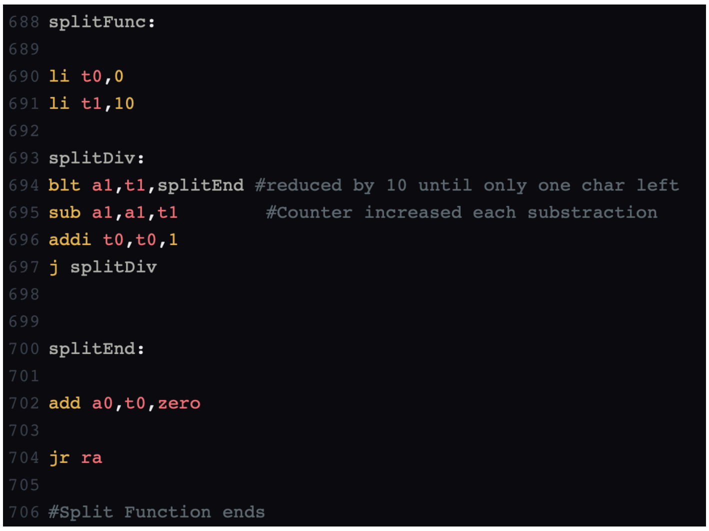
ADDED FUNCTIONS



```
#Module function

moduleFunction:

blt a1,zero,negativeLoop

positiveLoop:

blt a1,a0,moduleEnd
sub a1,a1,a0
j positiveLoop

negativeLoop:

bge a1,zero,moduleEnd
add a1,a1,a0
j negativeLoop

moduleEnd:
jr ra
#End of module function
```

-Module Function (moduleFunction)**:**

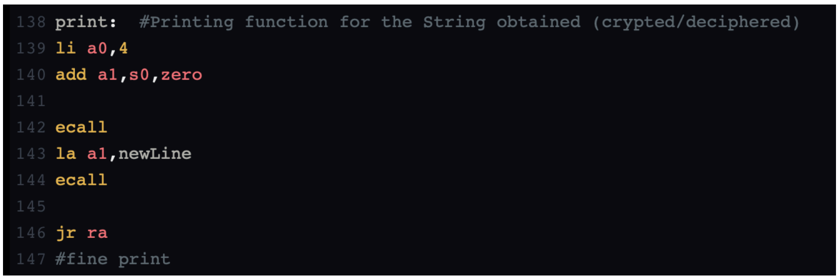
Takes 2 numbers in input (a1, a0), and with two loops chosen given the sign of a1 (positive or negative), by adding the other char if negative until a1 is smaller, or subtracting if positive until a1 is smaller than zero. The output is then saved in a1.



```
splitFunc:

li t0,0
li t1,10

splitDiv:
blt a1,t1,splitEnd #reduced by 10 until only one char left
sub a1,a1,t1        #Counter increased each substraction
addi t0,t0,1
j splitDiv


splitEnd:

add a0,t0,zero

jr ra

#Split Function ends
```

-Split Function (splitFunc) **:**

Takes a two-char number in input and "splits" it into two different bytes (a0 tens, a1 ones).



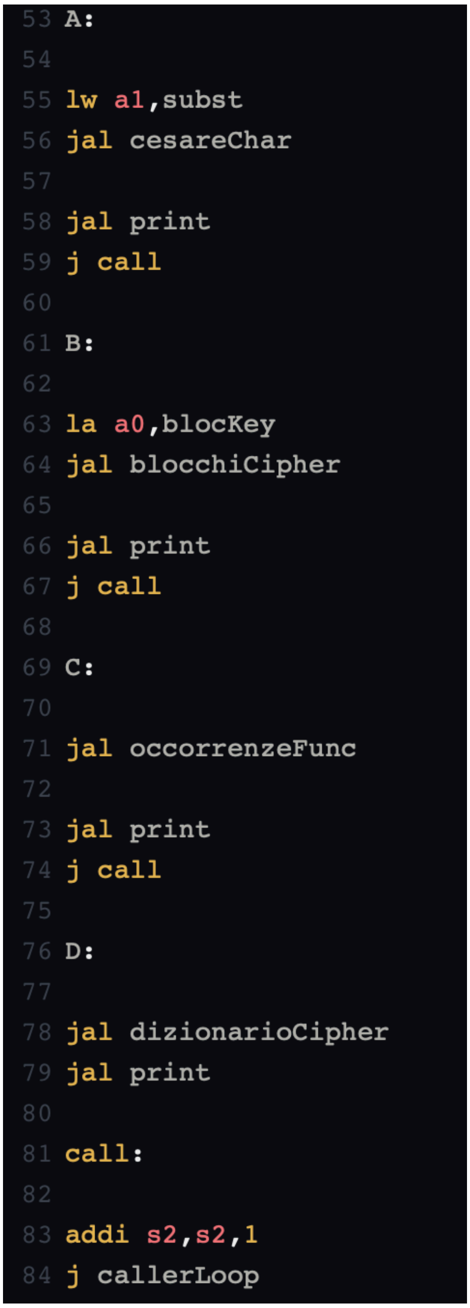
```
print:  #Printing function for the String obtained (crypted/deciphered)
li a0,4
add a1,s0,zero

ecall
la a1,newLine
ecall

jr ra
#fine print
```

-Print (print)**:**

Prints the string contained in s0 followed by a a new Line.

ABOUT THE ALGORITHMS

The program made, implements four different cyphering algorithms, and, as mentioned before, these are all marked by the key letters A, B, C and D:



```
A:

lw a1,subst
jal cesareChar

jal print
j call

B:

la a0,blocKey
jal blocchiCipher

jal print
j call

C:

jal occorrenzeFunc

jal print
j call

D:

jal dizionarioCipher
jal print

call:

addi s2,s2,1
j callerLoop
```

A: Sostituzione

B: Blocchi

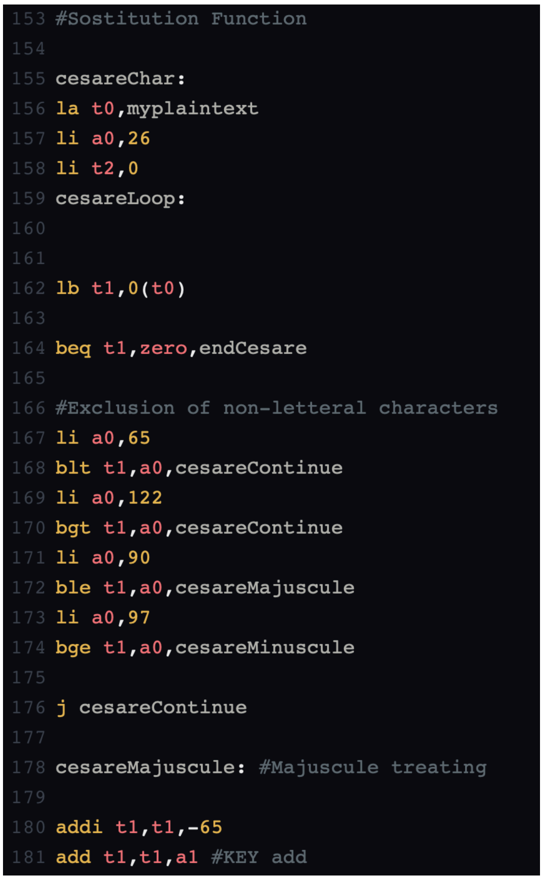
C: Occorrenze

D: Dizionario

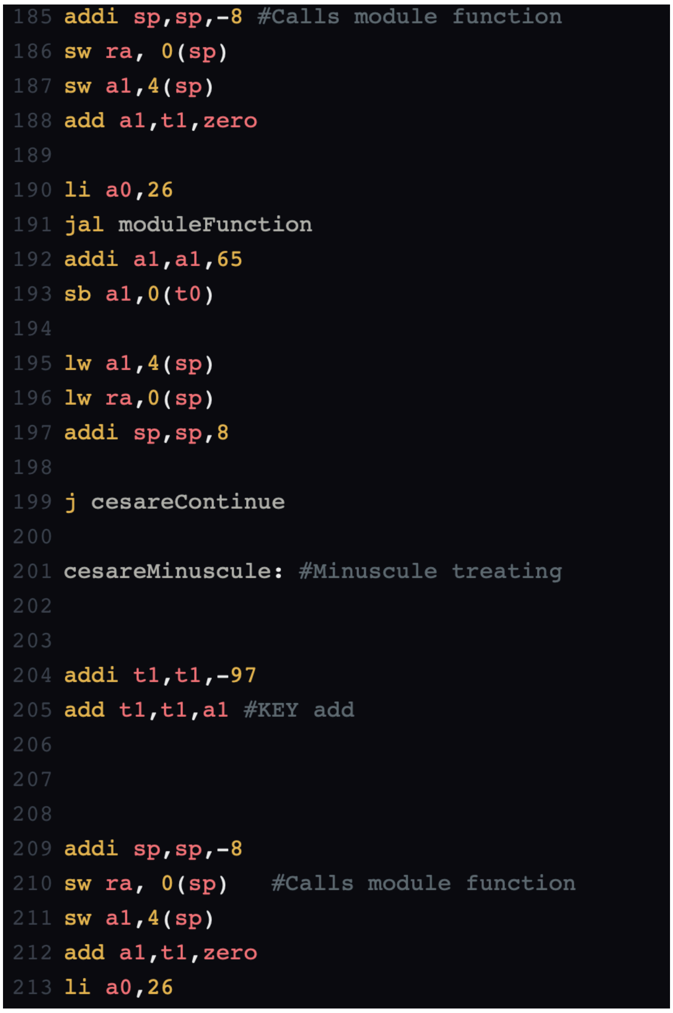
Each of the algorithms can work alone, or combined with other (or all) algorithms, and when implemented it outputs both the encrypted string and decrypted string to the user. These are all set to work one after the other in the MAIN part of the program, for the decrypting function the head of the vector is increased to use the algorithms in the opposite order than before.

#A: SOSTITUZIONE

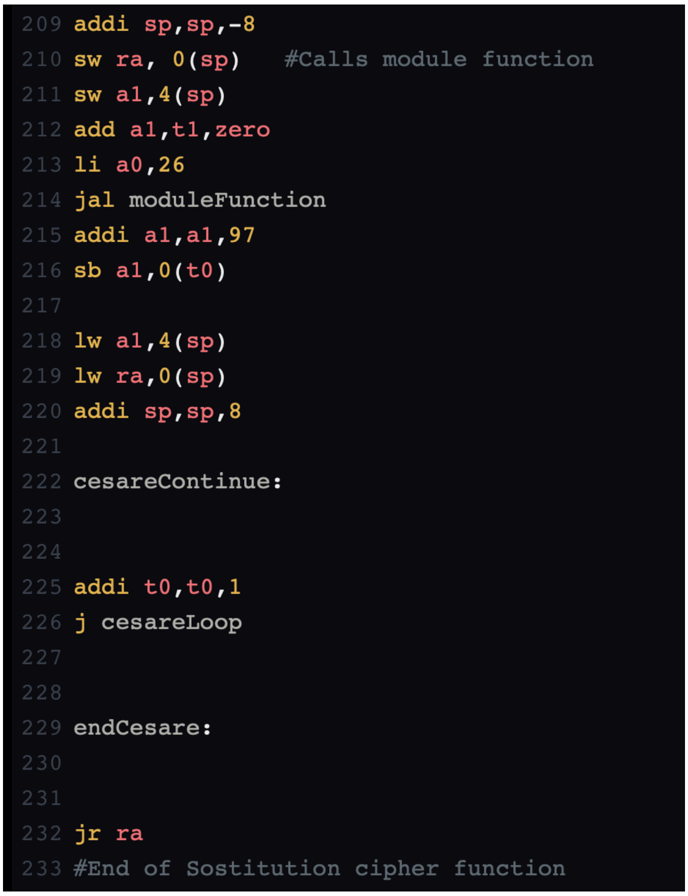
The Sostituzione algorithm, takes in input a parameter key, which is saved in a1. The key indicates how many positions to skip (or jump back if negative) to substitute the current character, giving us even more flexibility by being able to choose the jumps to be made as we prefer. This algorithm cyphers only letters (ASCII chars between 65-90), and in order to achieve that two loops are used (one for majuscules and one for minuscules).
The char is subtracted by 65 to obtain a char between 0-25. Then the parameter key is added via the module function (a1 = (char-65) + key). 65 is added to the value obtained if it's a majuscule and 97 if it's a minuscule.



```
#Sostitution Function

cesareChar:
la t0,myplaintext
li a0,26
li t2,0
cesareLoop:


lb t1,0(t0)

beq t1,zero,endCesare

#Exclusion of non-letteral characters
li a0,65
blt t1,a0,cesareContinue
li a0,122
bgt t1,a0,cesareContinue
li a0,90
ble t1,a0,cesareMajuscule
li a0,97
bge t1,a0,cesareMinuscule

j cesareContinue

cesareMajuscule: #Majuscule treating

addi t1,t1,-65
add t1,t1,a1 #KEY add
```



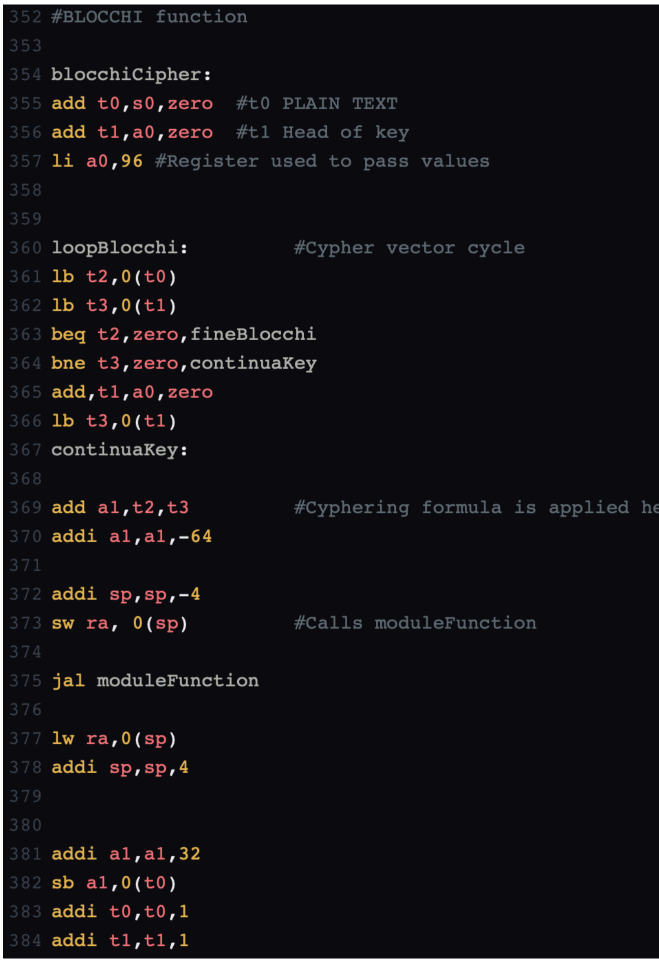
```
addi sp,sp,-8 #Calls module function
sw ra, 0(sp)
sw a1,4(sp)
add a1,t1,zero

li a0,26
jal moduleFunction
addi a1,a1,65
sb a1,0(t0)

lw a1,4(sp)
lw ra,0(sp)
addi sp,sp,8

j cesareContinue

cesareMinuscule: #Minuscule treating


addi t1,t1,-97
add t1,t1,a1 #KEY add



addi sp,sp,-8
sw ra, 0(sp)    #Calls module function
sw a1,4(sp)
add a1,t1,zero
li a0,26
```



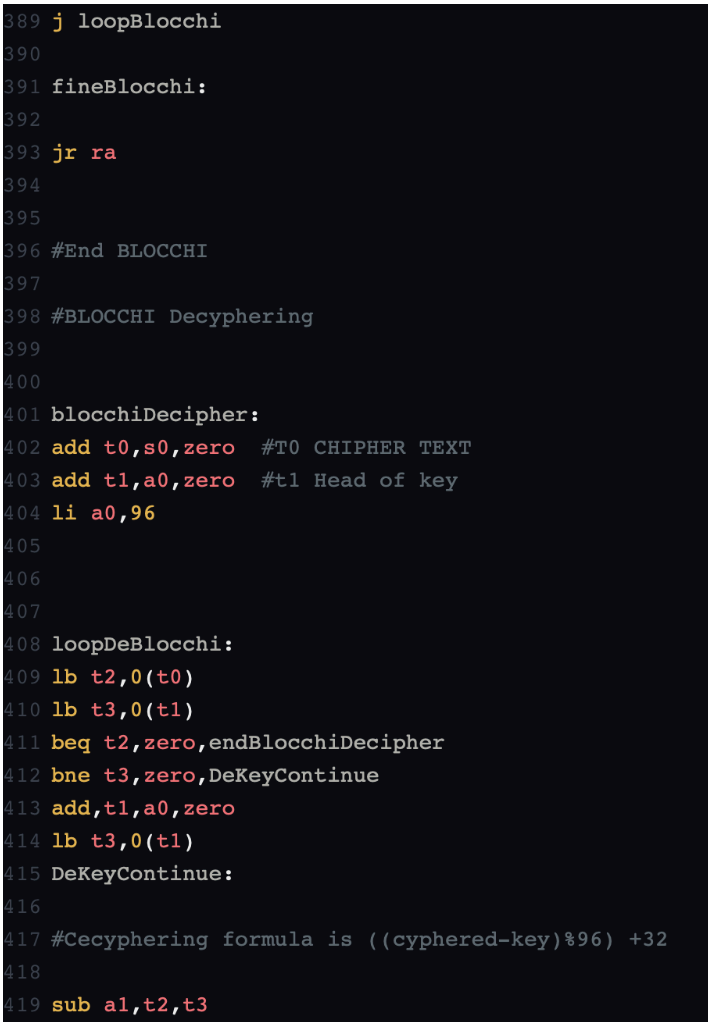
```
addi sp,sp,-8
sw ra, 0(sp)    #Calls module function
sw a1,4(sp)
add a1,t1,zero
li a0,26
jal moduleFunction
addi a1,a1,97
sb a1,0(t0)

lw a1,4(sp)
lw ra,0(sp)
addi sp,sp,8

cesareContinue:


addi t0,t0,1
j cesareLoop


endCesare:


jr ra
#End of Sostitution cipher function
```

#B: BLOCCHI

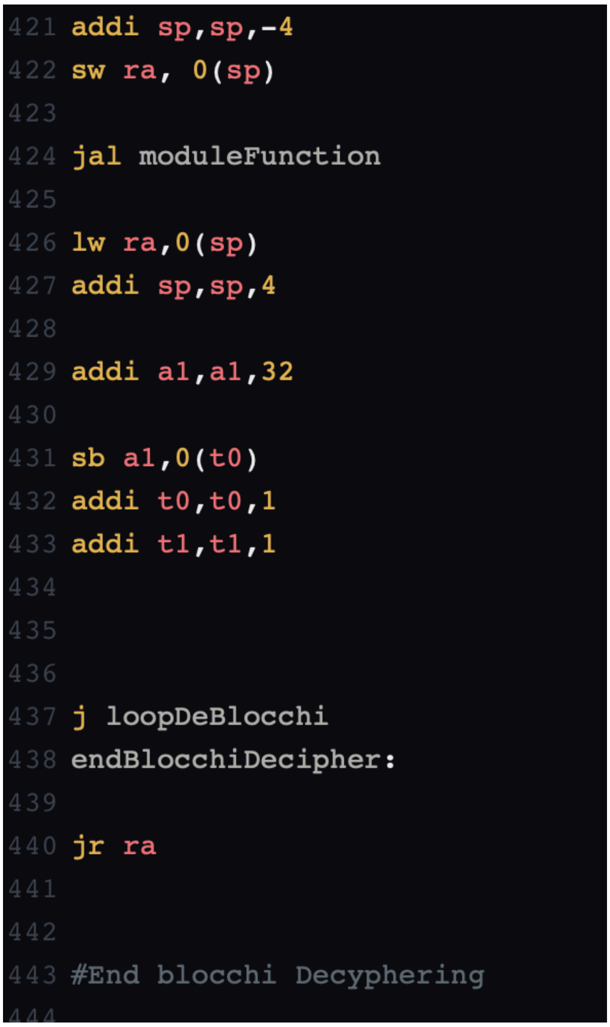
The Blocchi algorithm, works by partitioning the string according to the key and the length of the string. On each "block" obtained, the chars of the string are added to those of the key one by one.
It uses the module function to stay between its limits (32-127) with input of 96.Thus, for the module function to work chars must be between 0 to 95, so 64 is reduced before each application, and 32 is added after using the next formula:
<(((a+b)-64)%96) +32>



```
#BLOCCHI function

blocchiCipher:
add t0,s0,zero  #t0 PLAIN TEXT
add t1,a0,zero  #t1 Head of key
li a0,96 #Register used to pass values


loopBlocchi:          #Cypher vector cycle
lb t2,0(t0)
lb t3,0(t1)
beq t2,zero,fineBlocchi
bne t3,zero,continuaKey
add,t1,a0,zero
lb t3,0(t1)
continuaKey:

add a1,t2,t3          #Cyphering formula is applied he
addi a1,a1,-64

addi sp,sp,-4
sw ra, 0(sp)          #Calls moduleFunction

jal moduleFunction

lw ra,0(sp)
addi sp,sp,4


addi a1,a1,32
sb a1,0(t0)
addi t0,t0,1
addi t1,t1,1
```



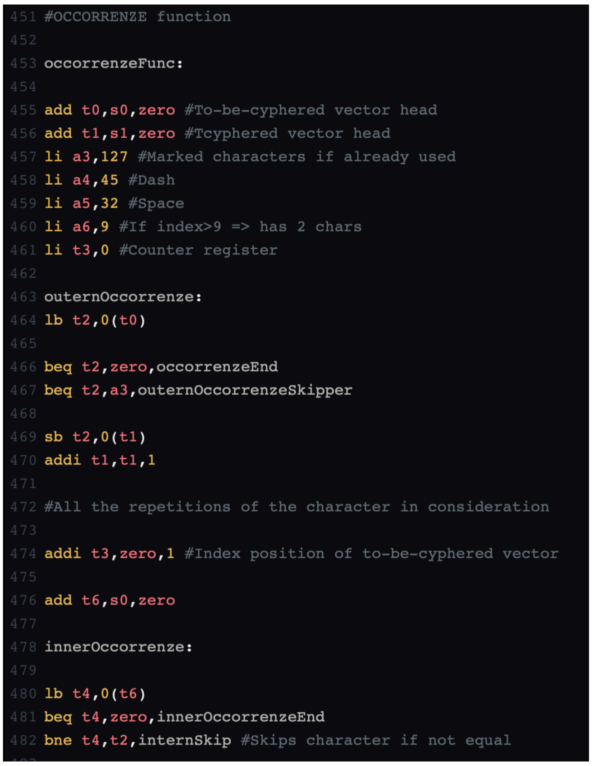
```
j loopBlocchi

fineBlocchi:

jr ra


#End BLOCCHI

#BLOCCHI Decyphering


blocchiDecipher:
add t0,s0,zero  #T0 CHIPHER TEXT
add t1,a0,zero  #t1 Head of key
li a0,96



loopDeBlocchi:
lb t2,0(t0)
lb t3,0(t1)
beq t2,zero,endBlocchiDecipher
bne t3,zero,DeKeyContinue
add,t1,a0,zero
lb t3,0(t1)
DeKeyContinue:

#Cecyphering formula is ((cyphered-key)%96) +32

sub a1,t2,t3
```



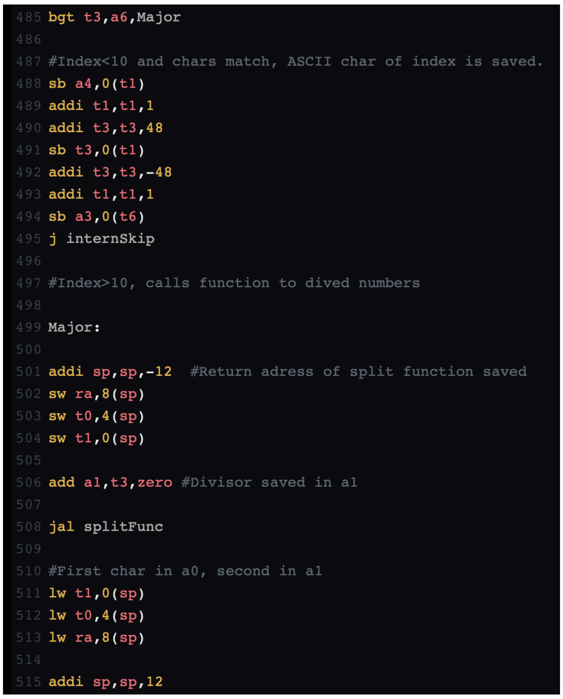
```
addi sp,sp,-4
sw ra, 0(sp)

jal moduleFunction

lw ra,0(sp)
addi sp,sp,4

addi a1,a1,32

sb a1,0(t0)
addi t0,t0,1
addi t1,t1,1



j loopDeBlocchi
endBlocchiDecipher:

jr ra


#End blocchi Decyphering
```

#C: OCCORRENZE

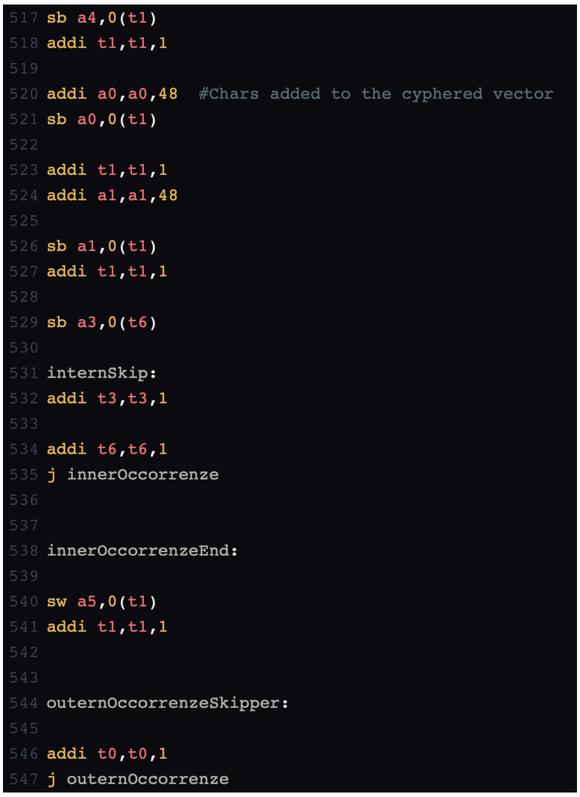
The Occorrenze algorithm cyphers chars between 0-99. It works by setting to each character the positions of where it appears. It uses two loops (one for choosing the char, the second one for checking where it appears in the vector).
In the case of the index of the repetition of a char being bigger than 9, then it is taken to the split function and saved in two consecutive cells.
To evade chars that have already been chosen, they are substituted by "escape" chars (in both loops).
This algorithm can't be made to work on-spot, because the output will always outnumber the starting vector, so an outer vector is used, which overwrites plainText with the cyphered text.
A stack is used to save the return addresses of splitFunc (t0, t1).



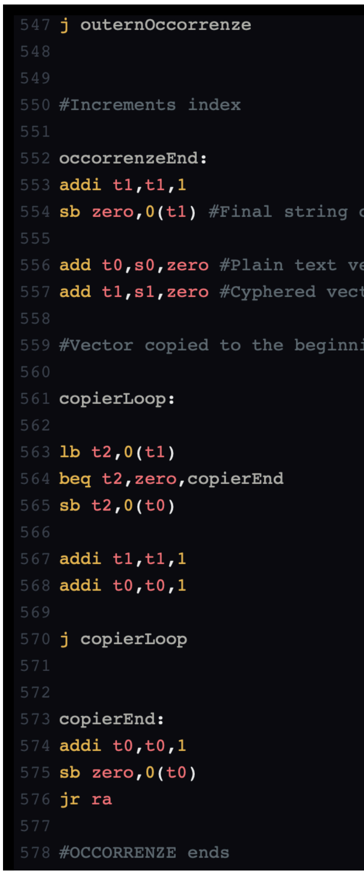
```
#OCCORRENZE function

occorrenzeFunc:

add t0,s0,zero #To-be-cyphered vector head
add t1,s1,zero #Tcyphered vector head
li a3,127 #Marked characters if already used
li a4,45 #Dash
li a5,32 #Space
li a6,9 #If index>9 => has 2 chars
li t3,0 #Counter register

outernOccorrenze:
lb t2,0(t0)

beq t2,zero,occorrenzeEnd
beq t2,a3,outernOccorrenzeSkipper

sb t2,0(t1)
addi t1,t1,1

#All the repetitions of the character in consideration

addi t3,zero,1 #Index position of to-be-cyphered vector

add t6,s0,zero

innerOccorrenze:

lb t4,0(t6)
beq t4,zero,innerOccorrenzeEnd
bne t4,t2,internSkip #Skips character if not equal
```



```
bgt t3,a6,Major

#Index<10 and chars match, ASCII char of index is saved.
sb a4,0(t1)
addi t1,t1,1
addi t3,t3,48
sb t3,0(t1)
addi t3,t3,-48
addi t1,t1,1
sb a3,0(t6)
j internSkip

#Index>10, calls function to dived numbers

Major:

addi sp,sp,-12  #Return adress of split function saved
sw ra,8(sp)
sw t0,4(sp)
sw t1,0(sp)

add a1,t3,zero #Divisor saved in a1

jal splitFunc

#First char in a0, second in a1
lw t1,0(sp)
lw t0,4(sp)
lw ra,8(sp)

addi sp,sp,12
```



```
sb a4,0(t1)
addi t1,t1,1

addi a0,a0,48  #Chars added to the cyphered vector
sb a0,0(t1)

addi t1,t1,1
addi a1,a1,48

sb a1,0(t1)
addi t1,t1,1

sb a3,0(t6)

internSkip:
addi t3,t3,1

addi t6,t6,1
j innerOccorrenze


innerOccorrenzeEnd:

sw a5,0(t1)
addi t1,t1,1


outernOccorrenzeSkipper:

addi t0,t0,1
j outernOccorrenze
```



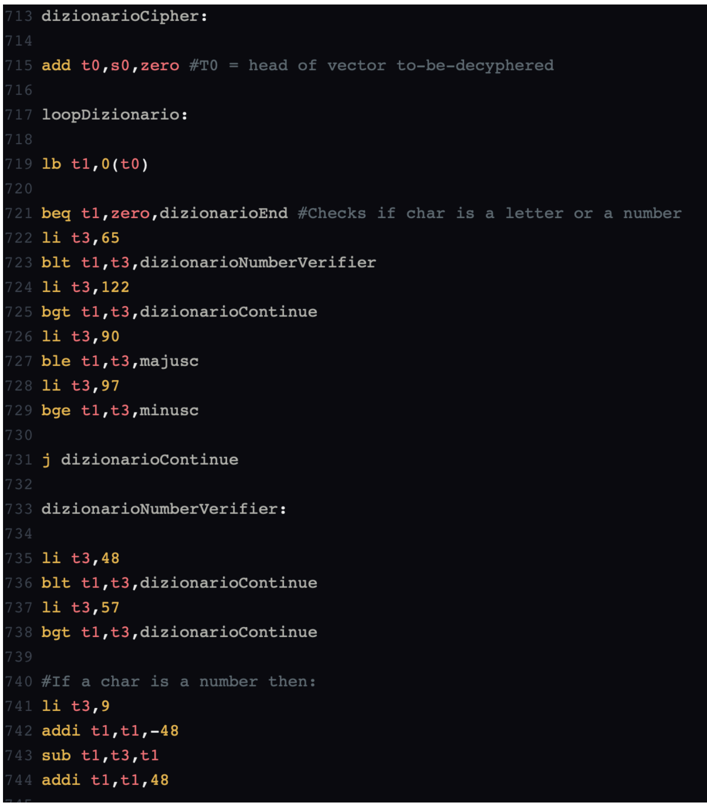
```
j outernOccorrenze


#Increments index

occorrenzeEnd:
addi t1,t1,1
sb zero,0(t1) #Final string c

add t0,s0,zero #Plain text ve
add t1,s1,zero #Cyphered vect

#Vector copied to the beginni

copierLoop:

lb t2,0(t1)
beq t2,zero,copierEnd
sb t2,0(t0)

addi t1,t1,1
addi t0,t0,1

j copierLoop


copierEnd:
addi t0,t0,1
sb zero,0(t0)
jr ra

#OCCORRENZE ends
```

#D: DIZIONARIO

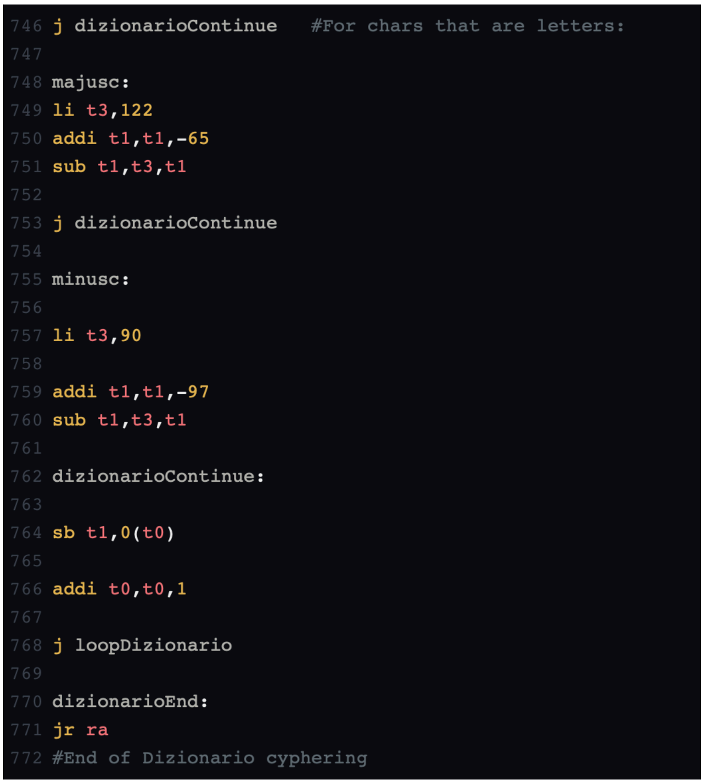
This algorithm maps each char with an ASCII equivalent by a function defined by the following cases:

- If the char is a minuscule -> the equivalent majuscule of the inverse alphabetical letter (ex: a => Z).
- If the char is a majuscule -> the equivalent minuscule of the inverse alphabetical letter (A => z).
- If a number -> an ASCII equivalent subtracted by 9 values.

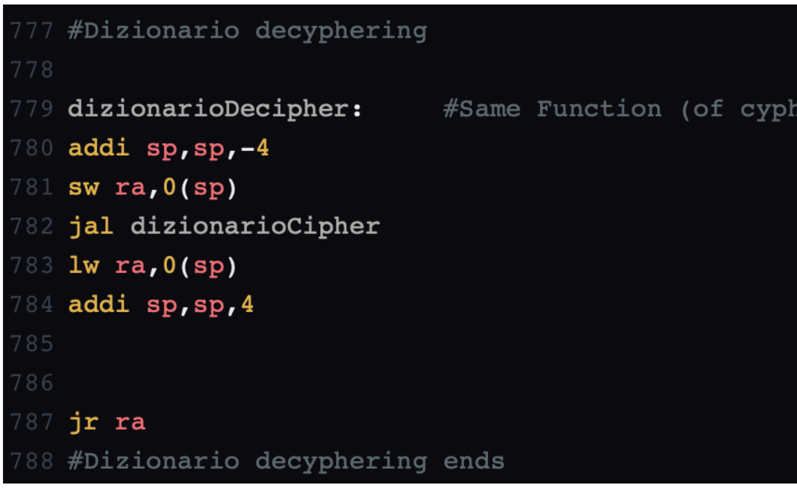
A function is used then which, according to the kind of the char, adds or subtracts an



```
dizionarioCipher:

add t0,s0,zero #T0 = head of vector to-be-decyphered

loopDizionario:

lb t1,0(t0)

beq t1,zero,dizionarioEnd #Checks if char is a letter or a number
li t3,65
blt t1,t3,dizionarioNumberVerifier
li t3,122
bgt t1,t3,dizionarioContinue
li t3,90
ble t1,t3,majusc
li t3,97
bge t1,t3,minusc

j dizionarioContinue

dizionarioNumberVerifier:

li t3,48
blt t1,t3,dizionarioContinue
li t3,57
bgt t1,t3,dizionarioContinue

#If a char is a number then:
li t3,9
addi t1,t1,-48
sub t1,t3,t1
addi t1,t1,48
```



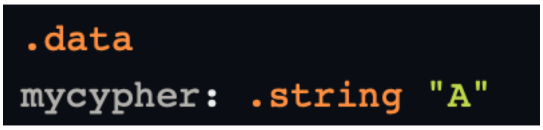
```
j dizionarioContinue   #For chars that are letters:

majusc:
li t3,122
addi t1,t1,-65
sub t1,t3,t1

j dizionarioContinue

minusc:

li t3,90

addi t1,t1,-97
sub t1,t3,t1

dizionarioContinue:

sb t1,0(t0)

addi t0,t0,1

j loopDizionario

dizionarioEnd:
jr ra
#End of Dizionario cyphering
```



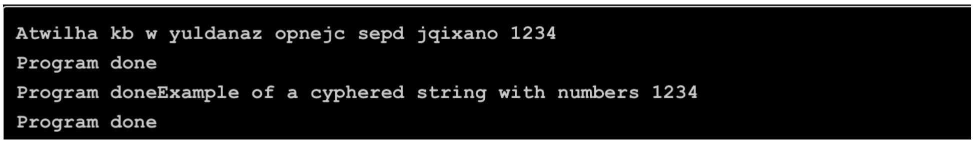
```
#Dizionario decyphering

dizionarioDecipher:     #Same Function (of cyph
addi sp,sp,-4
sw ra,0(sp)
jal dizionarioCipher
lw ra,0(sp)
addi sp,sp,4


jr ra
#Dizionario decyphering ends
```

TESTS

A:

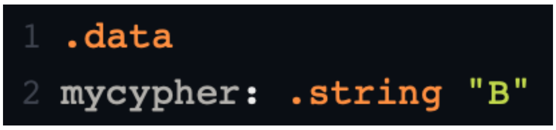


```
.data
mycypher:  .string "A"
```

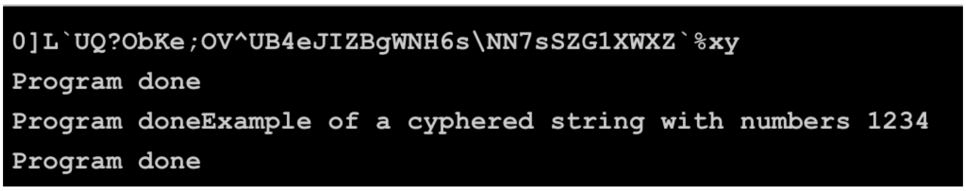


```
Atwilha kb w yuldanaz opnejc sepd jqixano 1234
Program done
Program doneExample of a cyphered string with numbers 1234
Program done
```

B:

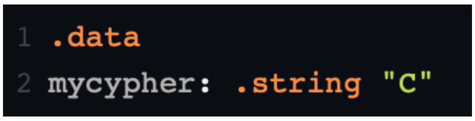


```
.data
mycypher:  .string "B"
```

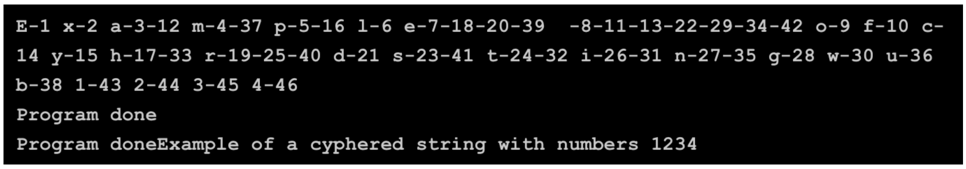


```
0]L`UQ?ObKe;OV^UB4eJIZBgWNH6s\NN7sSZG1XWXZ`%xy
Program done
Program doneExample of a cyphered string with numbers 1234
Program done
```

C:

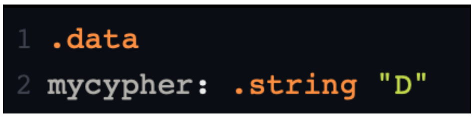


```
.data
mycypher:  .string "C"
```

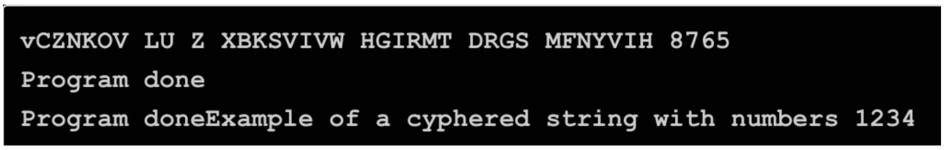


```
E-1 x-2 a-3-12 m-4-37 p-5-16 l-6 e-7-18-20-39  -8-11-13-22-29-34-42 o-9 f-10 c-
14 y-15 h-17-33 r-19-25-40 d-21 s-23-41 t-24-32 i-26-31 n-27-35 g-28 w-30 u-36
b-38 1-43 2-44 3-45 4-46
Program done
Program doneExample of a cyphered string with numbers 1234
```

D:

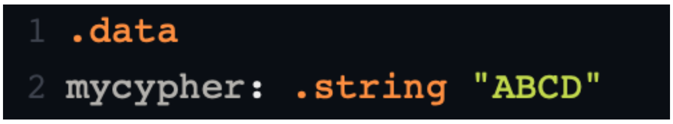


```
.data
mycypher:  .string "D"
```

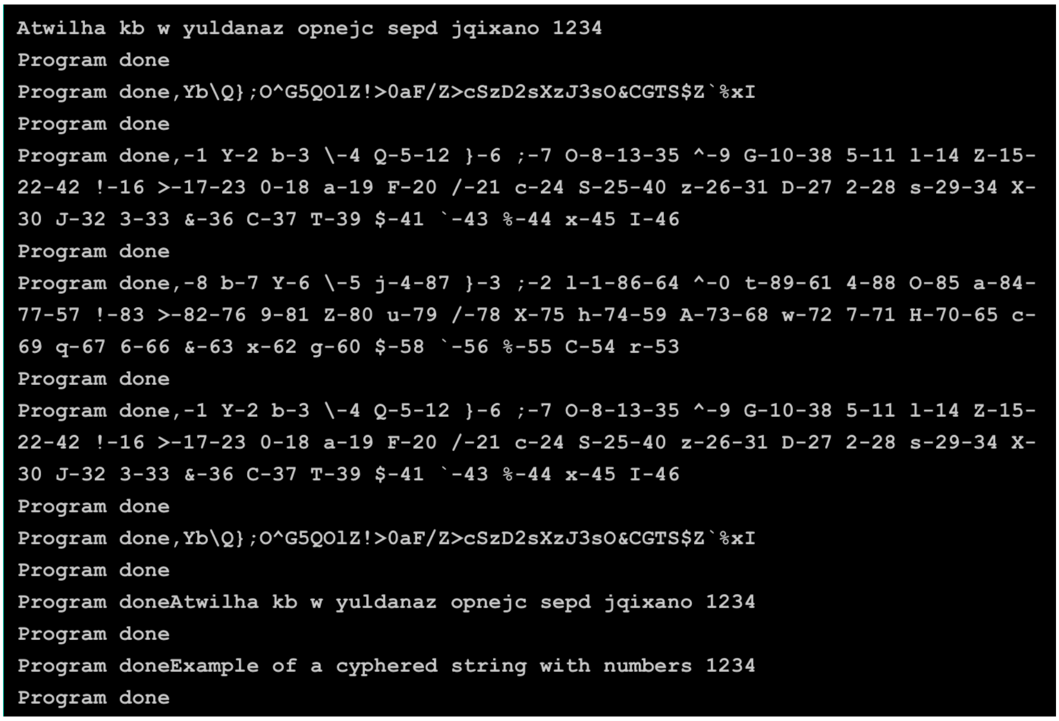


```
vCZNKOV LU Z XBKSVIVW HGIRMT DRGS MFNYVIH 8765
Program done
Program doneExample of a cyphered string with numbers 1234
```

ABCD:



```
.data
mycypher: .string "ABCD"
```



```
Atwilha kb w yuldanaz opnejc sepd jqixano 1234
Program done
Program done,Yb\Q};O^G5QO1Z!>0aF/Z>cSzD2sXzJ3sO&CGTS$Z`%xI
Program done
Program done,-1 Y-2 b-3 \-4 Q-5-12 }-6 ;-7 O-8-13-35 ^-9 G-10-38 5-11 1-14 Z-15-
22-42 !-16 >-17-23 0-18 a-19 F-20 /-21 c-24 S-25-40 z-26-31 D-27 2-28 s-29-34 X-
30 J-32 3-33 &-36 C-37 T-39 $-41 `-43 %-44 x-45 I-46
Program done
Program done,-8 b-7 Y-6 \-5 j-4-87 }-3 ;-2 l-1-86-64 ^-0 t-89-61 4-88 O-85 a-84-
77-57 !-83 >-82-76 9-81 Z-80 u-79 /-78 X-75 h-74-59 A-73-68 w-72 7-71 H-70-65 c-
69 q-67 6-66 &-63 x-62 g-60 $-58 `-56 %-55 C-54 r-53
Program done
Program done,-1 Y-2 b-3 \-4 Q-5-12 }-6 ;-7 O-8-13-35 ^-9 G-10-38 5-11 1-14 Z-15-
22-42 !-16 >-17-23 0-18 a-19 F-20 /-21 c-24 S-25-40 z-26-31 D-27 2-28 s-29-34 X-
30 J-32 3-33 &-36 C-37 T-39 $-41 `-43 %-44 x-45 I-46
Program done
Program done,Yb\Q};O^G5QO1Z!>0aF/Z>cSzD2sXzJ3sO&CGTS$Z`%xI
Program done
Program doneAtwilha kb w yuldanaz opnejc sepd jqixano 1234
Program done
Program doneExample of a cyphered string with numbers 1234
Program done
```

CODE

```
.data

mycypher: .string "ABCD"

subst: .word 100

blocKey: .string "kek"

newLine: .byte 10

finalString: .string "Program done"

stack: .word 1700

Supporter: .word 2000

myplaintext: .string  "Example of a cyphered string with numbers 1234"


.text

la s0,myplaintext

lw s1,Supporter

la s2,mycypher



#A cifrario di cesare / sostitution

#B blocchi
```

```
#C occorrenze

#D dizionario


callerLoop:	#Loop for calling the cryptage algorithms


lb t0,0(s2)

beq t0,zero,chardecipher

li a0,65

beq t0,a0,A

li a0,66

beq t0,a0,B

li a0,67

beq t0,a0,C

li a0,68

beq t0,a0,D

li a0,69


#Invalid character

j call
```

```
A:

lw a1,subst

jal cesareChar

jal print

j call


B:

la a0,blocKey

jal blocchiCipher

jal print

j call


C:

jal occorrenzeFunc

jal print

j call


D:
```

```
jal dizionarioCipher

jal print

call:

addi s2,s2,1

j callerLoop

chardecipher:

addi s2,s2,-1

decipheringLoopCaller:     #Runs myCypher as before but this time using the decription algorithms

li a0,65

lb t0,0(s2)

beq t0,zero,programEndAlg

beq t0,a0,Ade  # ;)

li a0,66

beq t0,a0,Bde

li a0,67

beq t0,a0,Cde

li a0,68

beq t0,a0,Dde

li a0,69
```

```
#Invalid Character
j deCaller
Ade:
lw a1,subst
jal cesareDecipher
jal print
j deCaller
Bde:
la a0,blocKey
jal blocchiDecipher
jal print
j deCaller

Cde:
jal occorrenzeDecipher
jal print
j deCaller

Dde:
```

```
jal dizionarioDecipher
jal print
deCaller:
addi s2,s2,-1
j decipheringLoopCaller
print:        #Printing function for the String obtained (crypted/deciphered)
li a0,4
add a1,s0,zero
ecall
la a1,newLine
ecall
jr ra
#fine print
#Sostitution Function
cesareChar:
la t0,myplaintext
li a0,26
li t2,0
cesareLoop:
```

```
lb t1,0(t0)
beq t1,zero,endCesare
#Exclusion of non-letteral characters
li a0,65
blt t1,a0,cesareContinue
li a0,122
bgt t1,a0,cesareContinue
li a0,90
ble t1,a0,cesareMajuscule
li a0,97
bge t1,a0,cesareMinuscule
j cesareContinue
cesareMajuscule: #Majuscule treating
addi t1,t1,-65
add t1,t1,a1 #KEY add
addi sp,sp,-8 #Calls module function
sw ra, 0(sp)
sw a1,4(sp)
add a1,t1,zero
```

```
li a0,26

jal moduleFunction

addi a1,a1,65

sb a1,0(t0)

lw a1,4(sp)

lw ra,0(sp)

addi sp,sp,8

j cesareContinue

cesareMinuscule: #Minuscule treating

addi t1,t1,-97

add t1,t1,a1 #KEY add

addi sp,sp,-8

sw ra, 0(sp)   #Calls module Function

sw a1,4(sp)

add a1,t1,zero

li a0,26

jal moduleFunction

addi a1,a1,97

sb a1,0(t0)
```

```
lw a1,4(sp)
lw ra,0(sp)
addi sp,sp,8
cesareContinue:
addi t0,t0,1
j cesareLoop


endCesare:
jr ra
#End of Sostitution cipher function
#Sostitution decipher function


cesareDecipher:
add t0,s0,zero


cesareDeLoop:
lb t1,0(t0)
beq t1,zero,endDeCesare
#Exclution of non-letteral characters
```

```
li a0,65
blt t1,a0,cesareDecipherContinue
li a0,122
bgt t1,a0,cesareDecipherContinue
li a0,90
ble t1,a0,cesareDecipherMajuscule
li a0,97
bge t1,a0,cesareDecipherMinuscule
j cesareDecipherContinue
cesareDecipherMajuscule:  #Majuscule treating
addi t1,t1,-65
sub t1,t1,a1    #Key subtracted
addi sp,sp,-8   #Calls module function
sw ra, 0(sp)
sw a1,4(sp)
add a1,t1,zero
li a0,26
jal moduleFunction
addi a1,a1,65
```

```
sb a1,0(t0)
lw a1,4(sp)
lw ra,0(sp)
addi sp,sp,8

j cesareDecipherContinue
cesareDecipherMinuscule:  #Minuscule treating
addi t1,t1,-97
sub t1,t1,a1 #Key Subtracted
addi sp,sp,-8   #Calls module function
sw ra, 0(sp)
sw a1,4(sp)
add a1,t1,zero
li a0,26
jal moduleFunction
addi a1,a1,97
sb a1,0(t0)
lw a1,4(sp)
lw ra,0(sp)
```

```
addi sp,sp,8



cesareDecipherContinue:

addi t0,t0,1

j cesareDeLoop



endDeCesare:

jr ra

#End of decipherage algorithm

#Module function

moduleFunction:

blt a1,zero,negativeLoop

positiveLoop:

blt a1,a0,moduleEnd

sub a1,a1,a0

j positiveLoop

negativeLoop:

bge a1,zero,moduleEnd

add a1,a1,a0
```

```
j negativeLoop

moduleEnd:

jr ra

#End of module function

#BLOCCHI function

blocchiCipher:

add t0,s0,zero #t0 PLAIN TEXT

add t1,a0,zero  #t1 Head of key

li a0,96 #Register used to pass values

loopBlocchi:        #Cypher vector cycle

lb t2,0(t0)

lb t3,0(t1)

beq t2,zero,fineBlocchi

bne t3,zero,continuaKey

add,t1,a0,zero

lb t3,0(t1)

continuaKey:

add a1,t2,t3        #Cyphering formula is applied here (((cyphered + key)-64)mod96)+32

addi a1,a1,-64
addi sp,sp,-4

sw ra, 0(sp)        #Calls moduleFunction

jal moduleFunction

lw ra,0(sp)

addi sp,sp,4

addi a1,a1,32

sb a1,0(t0)

addi t0,t0,1

addi t1,t1,1

j loopBlocchi

fineBlocchi:

jr ra

#End BLOCCHI

#BLOCCHI Decyphering

blocchiDecipher:

add t0,s0,zero #T0 CHIPHER TEXT

add t1,a0,zero  #t1 Head of key

li a0,96

loopDeBlocchi:
```

```
lb t2,0(t0)

lb t3,0(t1)

beq t2,zero,endBlocchiDecipher

bne t3,zero,DeKeyContinue

add,t1,a0,zero

lb t3,0(t1)

DeKeyContinue:

#Cecyphering formula is ((cyphered-key)%96) +32

sub a1,t2,t3

addi sp,sp,-4

sw ra, 0(sp)

jal moduleFunction

lw ra,0(sp)

addi sp,sp,4

addi a1,a1,32

sb a1,0(t0)

addi t0,t0,1

addi t1,t1,1

j loopDeBlocchi
```

```
endBlocchiDecipher:
jr ra
#End blocchi Decyphering
#OCCORRENZE function
occorrenzeFunc:
add t0,s0,zero #To-be-cyphered vector head
add t1,s1,zero #Tcyphered vector head
li a3,127 #Marked characters if already used
li a4,45 #Dash
li a5,32 #Space
li a6,9     #If index>9 => has 2 chars
li t3,0 #Counter register
outernOccorrenze:
lb t2,0(t0)
beq t2,zero,occorrenzeEnd
beq t2,a3,outernOccorrenzeSkipper
sb t2,0(t1)
addi t1,t1,1
#All the repetitions of the character in consideration
```

```
addi t3,zero,1 #Index position of to-be-cyphered vector
add t6,s0,zero
innerOccorrenze:
lb t4,0(t6)
beq t4,zero,innerOccorrenzeEnd
bne t4,t2,internSkip #Skips character if not equal
bgt t3,a6,Major
#Index<10 and chars match, ASCII char of index is saved.
sb a4,0(t1)
addi t1,t1,1
addi t3,t3,48
sb t3,0(t1)
addi t3,t3,-48
addi t1,t1,1
sb a3,0(t6)
j internSkip
#Index>10, calls function to dived numbers
Major:
addi sp,sp,-12   #Return adress of split function saved
```

```
sw ra,8(sp)
sw t0,4(sp)
sw t1,0(sp)
add a1,t3,zero #Divisor saved in a1
jal splitFunc
#First char in a0, second in a1
lw t1,0(sp)
lw t0,4(sp)
lw ra,8(sp)
addi sp,sp,12
sb a4,0(t1)
addi t1,t1,1
addi a0,a0,48  #Chars added to the cyphered vector
sb a0,0(t1)
addi t1,t1,1
addi a1,a1,48
sb a1,0(t1)
addi t1,t1,1
sb a3,0(t6)
```

```
internSkip:

addi t3,t3,1

addi t6,t6,1

j innerOccorrenze

innerOccorrenzeEnd:

sw a5,0(t1)

addi t1,t1,1

outernOccorrenzeSkipper:

addi t0,t0,1

j outernOccorrenze

#Increments index

occorrenzeEnd:

addi t1,t1,1

sb zero,0(t1) #Final string code

add t0,s0,zero #Plain text vector head

add t1,s1,zero #Cyphered vector head

#Vector copied to the beginning position

copierLoop:

lb t2,0(t1)
```

```
beq t2,zero,copierEnd

sb t2,0(t0)

addi t1,t1,1

addi t0,t0,1

j copierLoop

copierEnd:

addi t0,t0,1

sb zero,0(t0)

jr ra

#OCCORRENZE ends

#Occorrenze decypher function

occorrenzeDecipher:

add t0,s0,zero #t0 Tests to-be-deciphered vector

add t1,s1,zero   #t1 Tests to-be-deciphered support vector

li a0,0      #Counter

li a1,32 #Space

li a2,45 #Dash

deOuternOccorrenze:

lb t2,0(t0)
```

```
addi t0,t0,2                    #Skips first dash to next-spot character

beq t2,zero,deOccorrenzeEnd

innerDeOccorrenze:

lb t3,0(t0)

beq t3,a2,innerDeOccorrenzeSkipper    #Verifies character in consideration isn't neither a dash or space

beq t3,a1,outerDeOccorrenzeEnd

beq t3,zero,deOccorrenzeEnd

#insercion of character from t2 on index in t3

#if next character is also a number, both get unified

addi t3,t3,-48

#Checks if there are 2 numbers

addi t0,t0,1

lb t4,0(t0)

addi t0,t0,-1

beq t4,a2,goForward

beq t4,a1,goForward

beq t4,zero,goForward

#Two character index

addi t4,t4,-48
```

```
slli a4,t3,3                #Multiplies 10th char by 10, adds second char

slli t3,t3,1

add t3,a4,t3

add t3,t3,t4

addi t0,t0,1

goForward:

addi t3,t3,-1

add t1,t1,t3

sb t2,0(t1)

addi a0,a0,1

add t1,s1,zero #Restarts from head the to-be-deciphered vector

innerDeOccorrenzeSkipper:

addi t0,t0,1

j innerDeOccorrenze

outerDeOccorrenzeEnd:

addi t0,t0,1

j deOuternOccorrenze

deOccorrenzeEnd:

add t1,s1,zero
add t1,t1,a0
sb zero,0(t1)     #Zero added to mark the string end
add t1,s1,zero
add t0,s0,zero #Head of decyphered vector
copierLoopDe: #Loop that copies decyphered vector to plaintext address
lb t2,0(t1)
beq t2,zero,copierEndDe
sb t2,0(t0)
addi t1,t1,1
addi t0,t0,1
j copierLoopDe
copierEndDe:
sb zero,0(t0)
jr ra
#End of Occorrenze decyphering function
#Split Function
#Devides numbers bigger than 9 (a1: to-be-devided -> a0: first char, a1: second char
splitFunc:
li t0,0
li t1,10
splitDiv:
blt a1,t1,splitEnd #reduced by 10 until only one char left
sub a1,a1,t1        #Counter increased each substraction
addi t0,t0,1
j splitDiv
splitEnd:
add a0,t0,zero
jr ra
#Split Function ends
#DIZIONARIO cyphering function
dizionarioCipher:
add t0,s0,zero #T0 = head of vector to-be-decyphered
loopDizionario:
lb t1,0(t0)
beq t1,zero,dizionarioEnd #Checks if char is a letter or a number
li t3,65
blt t1,t3,dizionarioNumberVerifier
li t3,122
```

```
bgt t1,t3,dizionarioContinue
li t3,90
ble t1,t3,majusc
li t3,97
bge t1,t3,minusc
j dizionarioContinue
dizionarioNumberVerifier:
li t3,48
blt t1,t3,dizionarioContinue
li t3,57
bgt t1,t3,dizionarioContinue
#If a char is a number then:
li t3,9
addi t1,t1,-48
sub t1,t3,t1
addi t1,t1,48
j dizionarioContinue   #For chars that are letters:
majusc:
li t3,122
```

```
addi t1,t1,-65

sub t1,t3,t1

j dizionarioContinue

minusc:

li t3,90

addi t1,t1,-97

sub t1,t3,t1

dizionarioContinue:

sb t1,0(t0)

addi t0,t0,1

j loopDizionario

dizionarioEnd:

jr ra

#End of Dizionario cyphering

#Dizionario decyphering

dizionarioDecipher:         #Same Function (of cyphering)

addi sp,sp,-4

sw ra,0(sp)

jal dizionarioCipher
```

```
lw ra,0(sp)

addi sp,sp,4

jr ra

#Dizionario decyphering ends

programEndAlg:

la s0,finalString

jal print
```