

COMPUTER ARCHITECTURE - RISC V PROJECT

BY ALBERTO CHARABATI

7031859

ALBERTO.CHARABATI@STUD.UNIFI.IT

TABLE OF CONTENTS:

Versatility	4
Added functions	5
About the algorithms	6
#A: Sostituzione	7
#B: Blocchi	8
#C: occorrenze	9
#D: dizionario	10
Tests	11
Code	13

OBJECTIVES OF THE PROGRAM

C yphering 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.

```
2 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

```
28 #A cifrario di cesare / sostitution
29 #B blocchi
30 #C occorrenze
31 #D dizionario
```

encryption the program will use. Thus meaning that, for instance, by entering the input "AC", the program will implement the algorithms A and C.

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.

```
1 .data
2 mycypher: .string "ABCD"
3
4 subst: .word 100
5
6 blocKey: .string "kek"
7
8
9 newLine: .byte 10
10 finalString: .string "Program done"
11
12 stack: .word 1700
13 Supporter: .word 2000
14
15 myplaintext: .string "Example of a cyphered string with numbers 1234"
16
17
18
19
20 .text
21
22 la s0,myplaintext
23 lw s1,Supporter
24 la s2,mycypher
```

ADDED FUNCTIONS

```

327 #Module function
328
329 moduleFunction:
330
331 blt a1,zero,negativeLoop
332
333 positiveLoop:
334
335 blt a1,a0,moduleEnd
336 sub a1,a1,a0
337 j positiveLoop
338
339 negativeLoop:
340
341 bge a1,zero,moduleEnd
342 add a1,a1,a0
343 j negativeLoop
344
345 moduleEnd:
346 jr ra
347 #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.

```

688 splitFunc:
689
690 li t0,0
691 li t1,10
692
693 splitDiv:
694 blt a1,t1,splitEnd #reduced by 10 until only one char left
695 sub a1,a1,t1      #Counter increased each subtraction
696 addi t0,t0,1
697 j splitDiv
698
699
700 splitEnd:
701
702 add a0,t0,zero
703
704 jr ra
705
706 #Split Function ends

```

-Split Function (splitFunc) :

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

```

138 print: #Printing function for the String obtained (cryptd/deciphered)
139 li a0,4
140 add a1,s0,zero
141
142 ecall
143 la a1,newLine
144 ecall
145
146 jr ra
147 #fine print

```

-Print (print):

Prints the string contained in s0 followed by 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:

```

53 A:
54
55 lw a1,subst
56 jal cesareChar
57
58 jal print
59 j call
60
61 B:
62
63 la a0,blocKey
64 jal blocchiCipher
65
66 jal print
67 j call
68
69 C:
70
71 jal occorrenzeFunc
72
73 jal print
74 j call
75
76 D:
77
78 jal dizionarioCipher
79 jal print
80
81 call:
82
83 addi s2,s2,1
84 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 ($a1 = (\text{char} - 65) + \text{key}$). 65 is added to the value obtained if it's a majuscule and 97 if it's a minuscule.

```

153 #Sostitution Function
154
155 cesareChar:
156 la t0,myplaintext
157 li a0,26
158 li t2,0
159 cesareLoop:
160
161
162 lb t1,0(t0)
163
164 beq t1,zero,endCesare
165
166 #Exclusion of non-letteral characters
167 li a0,65
168 blt t1,a0,cesareContinue
169 li a0,122
170 bgt t1,a0,cesareContinue
171 li a0,90
172 ble t1,a0,cesareMajuscule
173 li a0,97
174 bge t1,a0,cesareMinuscule
175
176 j cesareContinue
177
178 cesareMajuscule: #Majuscule treating
179
180 addi t1,t1,-65
181 add t1,t1,a1 #KEY add

```

```

185 addi sp,sp,-8 #Calls module function
186 sw ra, 0(sp)
187 sw a1,4(sp)
188 add a1,t1,zero
189
190 li a0,26
191 jal moduleFunction
192 addi a1,a1,65
193 sb a1,0(t0)
194
195 lw a1,4(sp)
196 lw ra,0(sp)
197 addi sp,sp,8
198
199 j cesareContinue
200
201 cesareMinuscule: #Minuscule treating
202
203
204 addi t1,t1,-97
205 add t1,t1,a1 #KEY add
206
207
208
209 addi sp,sp,-8
210 sw ra, 0(sp) #Calls module function
211 sw a1,4(sp)
212 add a1,t1,zero
213 li a0,26

```

```

209 addi sp,sp,-8
210 sw ra, 0(sp) #Calls module function
211 sw a1,4(sp)
212 add a1,t1,zero
213 li a0,26
214 jal moduleFunction
215 addi a1,a1,97
216 sb a1,0(t0)
217
218 lw a1,4(sp)
219 lw ra,0(sp)
220 addi sp,sp,8
221
222 cesareContinue:
223
224
225 addi t0,t0,1
226 j cesareLoop
227
228
229 endCesare:
230
231
232 jr ra
233 #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:

$$<(((a+b)-64)\%96) + 32>$$

```

352 #BLOCCHI function
353
354 blocchiCipher:
355 add t0,s0,zero #t0 PLAIN TEXT
356 add t1,a0,zero #t1 Head of key
357 li a0,96 #Register used to pass values
358
359
360 loopBlocchi:          #Cypher vector cycle
361 lb t2,0(t0)
362 lb t3,0(t1)
363 beq t2,zero,fineBlocchi
364 bne t3,zero,continuaKey
365 add,t1,a0,zero
366 lb t3,0(t1)
367 continuaKey:
368
369 add a1,t2,t3          #Cyphering formula is applied here
370 addi a1,a1,-64
371
372 addi sp,sp,-4
373 sw ra, 0(sp)          #Calls moduleFunction
374
375 jal moduleFunction
376
377 lw ra,0(sp)
378 addi sp,sp,4
379
380
381 addi a1,a1,32
382 sb a1,0(t0)
383 addi t0,t0,1
384 addi t1,t1,1

```

```

389 j loopBlocchi
390
391 fineBlocchi:
392
393 jr ra
394
395
396 #End BLOCCHI
397
398 #BLOCCHI Decyphering
399
400
401 blocchiDecipher:
402 add t0,s0,zero #T0 CHIPHER TEXT
403 add t1,a0,zero #t1 Head of key
404 li a0,96
405
406
407
408 loopDeBlocchi:
409 lb t2,0(t0)
410 lb t3,0(t1)
411 beq t2,zero,endBlocchiDecipher
412 bne t3,zero,DeKeyContinue
413 add,t1,a0,zero
414 lb t3,0(t1)
415 DeKeyContinue:
416
417 #Decyphering formula is ((cyphered-key)%96) + 32
418
419 sub a1,t2,t3

```

```

421 addi sp,sp,-4
422 sw ra, 0(sp)
423
424 jal moduleFunction
425
426 lw ra,0(sp)
427 addi sp,sp,4
428
429 addi a1,a1,32
430
431 sb a1,0(t0)
432 addi t0,t0,1
433 addi t1,t1,1
434
435
436
437 j loopDeBlocchi
438 endBlocchiDecipher:
439
440 jr ra
441
442
443 #End blocchi Decyphering
444

```


#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).

```

451 #OCCORRENZE function
452
453 occorrenzeFunc:
454
455 add t0,s0,zero #To-be-cyphered vector head
456 add t1,s1,zero #Tcyphered vector head
457 li a3,127 #Marked characters if already used
458 li a4,45 #Dash
459 li a5,32 #Space
460 li a6,9 #If index>9 => has 2 chars
461 li t3,0 #Counter register
462
463 outernOccorrenze:
464 lb t2,0(t0)
465
466 beq t2,zero,occorrenzeEnd
467 beq t2,a3,outernOccorrenzeSkipper
468
469 sb t2,0(t1)
470 addi t1,t1,1
471
472 #All the repetitions of the character in consideration
473
474 addi t3,zero,1 #Index position of to-be-cyphered vector
475
476 add t6,s0,zero
477
478 innerOccorrenze:
479
480 lb t4,0(t6)
481 beq t4,zero,innerOccorrenzeEnd
482 bne t4,t2,internSkip #Skips character if not equal

```

```

485 bgt t3,a6,Major
486
487 #Index<10 and chars match, ASCII char of index is saved.
488 sb a4,0(t1)
489 addi t1,t1,1
490 addi t3,t3,48
491 sb t3,0(t1)
492 addi t3,t3,-48
493 addi t1,t1,1
494 sb a3,0(t6)
495 j internSkip
496
497 #Index>10, calls function to divided numbers
498
499 Major:
500
501 addi sp,sp,-12 #Return adress of split function saved
502 sw ra,8(sp)
503 sw t0,4(sp)
504 sw t1,0(sp)
505
506 add a1,t3,zero #Divisor saved in a1
507
508 jal splitFunc
509
510 #First char in a0, second in a1
511 lw t1,0(sp)
512 lw t0,4(sp)
513 lw ra,8(sp)
514
515 addi sp,sp,12

```

```

517 sb a4,0(t1)
518 addi t1,t1,1
519
520 addi a0,a0,48 #Chars added to the cyphered vector
521 sb a0,0(t1)
522
523 addi t1,t1,1
524 addi a1,a1,48
525
526 sb a1,0(t1)
527 addi t1,t1,1
528
529 sb a3,0(t6)
530
531 internSkip:
532 addi t3,t3,1
533
534 addi t6,t6,1
535 j innerOccorrenze
536
537 innerOccorrenzeEnd:
538
539
540 sw a5,0(t1)
541 addi t1,t1,1
542
543
544 outernOccorrenzeSkipper:
545
546 addi t0,t0,1
547 j outernOccorrenze

```

```

547 j outernOccorrenze
548
549
550 #Increments index
551
552 occorrenzeEnd:
553 addi t1,t1,1
554 sb zero,0(t1) #Final string c
555
556 add t0,s0,zero #Plain text ve
557 add t1,s1,zero #Cyphered vect
558
559 #Vector copied to the beginn
560
561 copierLoop:
562
563 lb t2,0(t1)
564 beq t2,zero,copierEnd
565 sb t2,0(t0)
566
567 addi t1,t1,1
568 addi t0,t0,1
569
570 j copierLoop
571
572
573 copierEnd:
574 addi t0,t0,1
575 sb zero,0(t0)
576 jr ra
577
578 #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

```

713 dizionarioCipher:
714
715 add t0,s0,zero #T0 = head of vector to-be-decyphered
716
717 loopDizionario:
718
719 lb t1,0(t0)
720
721 beq t1,zero,dizionarioEnd #Checks if char is a letter or a number
722 li t3,65
723 blt t1,t3,dizionarioNumberVerifier
724 li t3,122
725 bgt t1,t3,dizionarioContinue
726 li t3,90
727 ble t1,t3,majusc
728 li t3,97
729 bge t1,t3,minusc
730
731 j dizionarioContinue
732
733 dizionarioNumberVerifier:
734
735 li t3,48
736 blt t1,t3,dizionarioContinue
737 li t3,57
738 bgt t1,t3,dizionarioContinue
739
740 #If a char is a number then:
741 li t3,9
742 addi t1,t1,-48
743 sub t1,t3,t1
744 addi t1,t1,48

```

```

746 j dizionarioContinue #For chars that are letters:
747
748 majusc:
749 li t3,122
750 addi t1,t1,-65
751 sub t1,t3,t1
752
753 j dizionarioContinue
754
755 minusc:
756
757 li t3,90
758
759 addi t1,t1,-97
760 sub t1,t3,t1
761
762 dizionarioContinue:
763
764 sb t1,0(t0)
765
766 addi t0,t0,1
767
768 j loopDizionario
769
770 dizionarioEnd:
771 jr ra
772 #End of Dizionario cyphering

```

```

777 #Dizionario decyphering
778
779 dizionarioDecipher: #Same Function (of cyphering)
780 addi sp,sp,-4
781 sw ra,0(sp)
782 jal dizionarioCipher
783 lw ra,0(sp)
784 addi sp,sp,4
785
786
787 jr ra
788 #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:

```
1 .data
2 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:

```
1 .data
2 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:

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

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

ABCD:

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

```
Atwilha kb w yuldanaz opnejc sepd jqixano 1234
Program done
Program done,Yb\Q};O^G5Q01Z!>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 $-41 ` -43 %-44 x-45 I-46
Program done
Program done,Yb\Q};O^G5Q01Z!>0aF/Z>cSzD2sXzJ3sO&CGTS$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

blockKey: .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

RISC V PROJECT

#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,blockKey

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

RISC V PROJECT

#Invalid Character

j deCaller

Ade:

lw a1,subst

jal cesareDecipher

jal print

j deCaller

Bde:

la a0,blockKey

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:

RISC V PROJECT

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

RISC V PROJECT

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)

RISC V PROJECT

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
```


RISC V PROJECT

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)

RISC V PROJECT

```
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
```

RISC V PROJECT

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

RISC V PROJECT

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 #TO CHIPHER TEXT

add t1,a0,zero #t1 Head of key

li a0,96

loopDeBlocchi:

RISC V PROJECT

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 $((\text{cyphered-key})\%96) + 32$

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

RISC V PROJECT

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

RISC V PROJECT

```
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)

RISC V PROJECT

```
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
```

RISC V PROJECT

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

RISC V PROJECT

```
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 subtraction

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

RISC V PROJECT

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

RISC V PROJECT

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

RISC V PROJECT

lw ra,0(sp)

addi sp,sp,4

jr ra

#Dizionario decyphering ends

programEndAlg:

la s0,finalString

jal print