



PROGETTO ADE 2019/2020

Utente: Barbieri Emanuele Luca

Matricola: 6147014

Email: emanuele.barbieri@stud.unifi.it

Data: 24/06/2020

INDICE

INTRODUZIONE

1 .data

- 1.1 *Input*
- 1.2 *Messaggio Stringa*
- 1.3 *Errori*
- 1.4 *Elementi extra*

2 .text

- 2.1 *Fase Blocco Errori*
- 2.2 *Fase di ricavo spazio*
- 2.3 *Inizio Programma*
 - 2.3.1 *Fase Cifratura*
 - 2.3.1.1 *Cifrario Cesare*
 - 2.3.1.2 *Cifrario a Blocchi*
 - 2.3.1.3 *Cifrario a Occorrenze*
 - 2.3.1.4 *Dizionario*
 - 2.3.2 *Fase Decifratura*
 - 2.3.2.1 *Inversione*

3 Esempi

4 Codice

5 Note informativa

INTRODUZIONE

Il progetto consiste nella CIFRATURA e DECIFRATURA di una variabile stringa. Non si applica soltanto UN solo tipo di algoritmo, ma ne esistono ben cinque, dove quattro sono per la cifratura e il quinto è per la decifratura del testo.

Questi algoritmi sono:

- 1) **Cifrario Cesare**, per semplificazione prende il nome di Algoritmo A
- 2) **Cifrario a Blocchi**, chiamato Algoritmo B
- 3) **Cifrario a Occorrenze**, chiamato Algoritmo C
- 4) **Dizionario**, chiamato Algoritmo D
- 5) **Inversione**, chiamato algoritmo E. Quest'ultimo viene utilizzato per la decifratura.

Mescolandoli e applicandoli al testo da cifrare si ottiene una sequenza di stringhe molto diverse tra di loro e, soprattutto, si ottiene un testo non facilmente intuibile.

.data

La prima cosa essenziale da fare all'interno di un programma assembly è mettere in evidenza degli elementi che saranno fondamentali all'interno del codice.

Queste variabili verranno inseriti in *.data*.

All'interno di questa sezione, per quanto riguarda questo codice, troviamo:

- 1) Valori input essenziali
- 2) Messaggi
- 3) Elementi extra

I **valori input** sono principalmente 4 che possono essere suddivisi in 3 variabili stringhe e 1 costante intera.

Le stringhe sono: *mychip*, *myplaintext*, *key*.

Il valore costante: *k*.

A cosa servono queste variabili?

- I. Il Myplaintext: Sarebbe la stringa più importante all'interno del codice, in quanto consiste nel **testo** che noi vogliamo *cifrare*
- II. Mychip: All'interno del codice, come già accennato nell'introduzione, sono presenti 4 algoritmi per la cifratura. Il mychip è una stringa che contiene la sequenza degli algoritmi che devono essere applicati al testo

Per esempio:

Dato il *myplaintext*="Voglio Cifrare Questo TESTO" e il *mychip*="AABCD" significa che al "Voglio Cifrare Questo TESTO" applico i seguenti algoritmi: **A->A->B->C->D**

- III. K & Key: questi elementi verranno utilizzati negli algoritmi A e B, per questo motivo verranno spiegati nel momento in cui si inizierà a parlare singolarmente di ognuno di loro.

I **messaggi** sono semplicemente delle variabili stringa che servono soprattutto per rendere l'esecuzione del codice più comprensibile. Nel *.data* troviamo dieci messaggi, tra cui, sei servono per capire il procedimento che sta eseguendo il programma (*stringaMessaggio1...stringaMessaggio6*), mentre le ultime quattro sono messaggi che si presentano solo nel momento in cui si verificano errori nel Mychip e nel Myplaintext (*stringa_errore1...stringa_errore4*)

Errori

Gli errori si suddividono in due categorie:

- 1) *Errori gestiti dal codice*
- 2) *Errori non gestiti*

1) I primi sono controllati dal programmatore. I possibili errori che si possono presentare sono:

- L'inserimento di *Lettere* che non coincidono con gli algoritmi sviluppati, quindi un errore presente nel mychipper.

Esempio

Se gli Algoritmi sono: "A,B,C,D", trovare nel mychipper lettere come "a,b,E,F,f,g..." risulterebbe un errore

- I caratteri nel mychipper devono essere maggiori di 0 e minori di 5
- I caratteri nel Myplaintext devono essere maggiori di 0 e minori di 100.
- Nel mychipper se eseguo troppe volte l'algoritmo C, il myplaintext supererebbe i 100 caratteri, questo risulterebbe un errore. Infatti, l'algoritmo calcola quanti C possono essere eseguiti in base alla lunghezza della stringa da cifrare. Questo è un piccolo grafico che spiega come i caratteri nel myplaintext aumentano se ho una sequenza di C nel mychipper. Nell'algoritmo viene utilizzata la formula evidenziata in **scuro**. (x = lunghezza myplaintext)

$$f(x) = \begin{cases} (x * 3) + (x - 1), & 0 < x < 10 \\ (x * 4) + (x - 1), & x \geq 10 \end{cases}$$



Figura 1: Quella violetta corrisponde alla $(x * 3) + (x - 1)$, mentre $(x * 4) + (x - 1)$ arancione scuro

Esempio

Myplaintext = "Ciao!" quindi $x = 5$, Mychipers = "CCC"

X	5	24	119
Y	24	119	594

Osservazione: Notare come il risultato di Y completa la tabella di X. Infatti, il myplaintext parte con $x = 5$ fino ad arrivare a 594.

Attenzione: algoritmo, come detto in precedenza, effettua la formula per valori decimali usata per approssimare sia valori decimali e non. (Figura1)

L'algoritmo di controllo evita che il nostro myplaintext superi i 100 caratteri.

Nel momento in cui si parlerà del *.text* si vedrà come questi algoritmi sono stati programmati.

2) Mentre gli *errori non gestiti*, sono quelli che il programmatore non può controllare. Nel programma i possibili errori sono tre:

- Inserire nel mychipers e nel myplaintext un *Numero di caratteri* che è divisibile per quattro.

Esempio

```
14 mychipers: .string "ABCD"
```

```
17 myplaintext: .string "Amo Assembly"
```

Il mychipers è composto da *quattro caratteri* che è divisibile per 4; stessa cosa per il Myplaintext, composto da *dodici caratteri*.

- L'uso delle *virgolette* in qualsiasi testo stringa nel *.data* non viene considerato.

Esempio

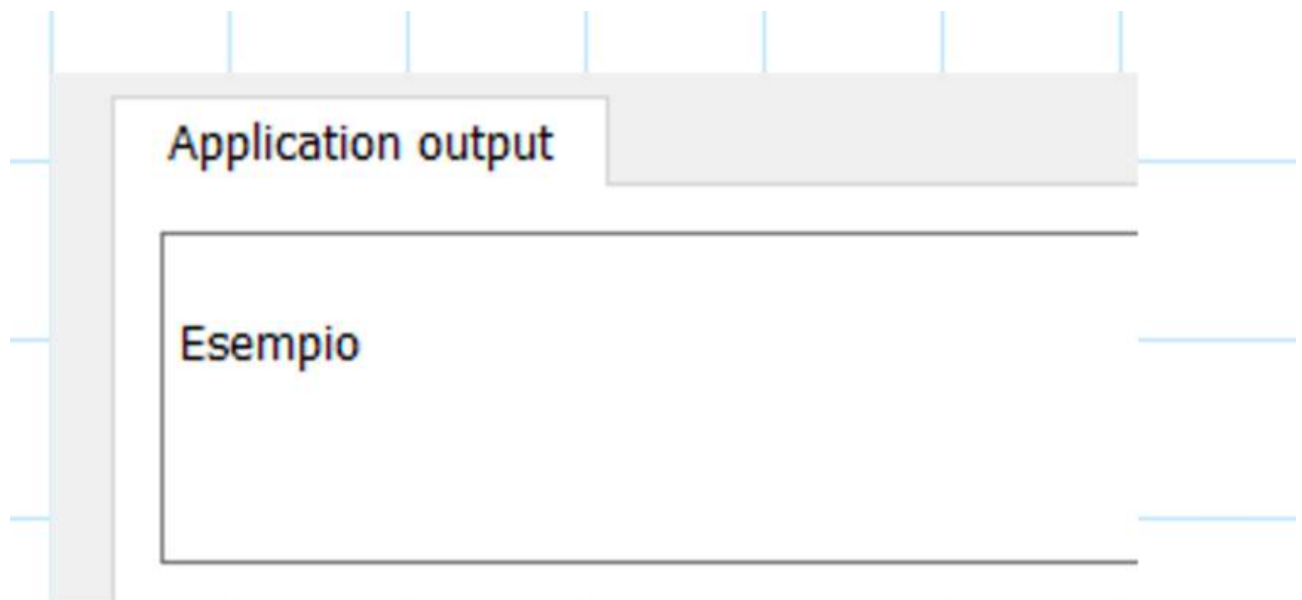
Prendiamo un semplice programma di stampa, dove la nostra stringa contiene delle virgolette.

```

1 .data
2 myplaintext: .string "Esempio"
3 .text
4 la a1, myplaintext
5 li a0, 4
6 ecall

```

Esecuzione



- Errore nella Key, in cui all'interno ci deve essere almeno un carattere. Ho provato a gestirlo, ma questo non viene considerato dal ripes

```

#controllo key
la a0, key
jal lunghezzaArray
la a1, stringa_errore5
beq a0, zero, errore_controllo

```

Per evitare questi errori, sta alla *discrezione dell'utente* a non inserirli.

L'ultimo argomento che manca da trattare nel capitolo del *.data* sono gli **elementi extra**.

Gli elementi extra sono quegli elementi che non sono definiti nella traccia del problema, ma che servono soprattutto per il funzionamento di alcuni algoritmi. In questo caso vengono utilizzati per l'algoritmo C e sono due:

- Space: una stringa di solo spazi vuoti. Gli spazi sono 484; ovviamente il numero non è casuale, ma cerca di considerare, nella funzione che abbiamo visto prima, il caso peggiore.

Caso peggiore:

In totale possiamo avere massimo 99 caratteri, di cui 9 per i valori non decimali e (99-9) per i valori decimali

$$f(x) = \begin{cases} (9 * 3) + (9 - 1) \\ (90 * 4) + (90 - 1) \end{cases}$$

$$(9 * 3) + (9 - 1) = 35$$

$$(90 * 4) + (90 - 1) = 449$$

$$35 + 449 = 484$$

Anche se è un numero che non verrà mai raggiunto in quanto 95 sono i possibili caratteri nel codice ASCII, non considerando i caratteri particolari, il suo massimo valore dovrebbe essere intorno ai 472.

Il calcolo su 95 caratteri:

$$(9 * 3) + (9 - 1) = 35$$

$$(86 * 4) + (86 - 1) = 429$$

$$35 + 429 = 464$$

Dato che posso aggiungere altri 4 caratteri (99-95), che saranno uguali a quelli inseriti precedentemente prenderanno solo 2 o 3 posizioni (trattino-valore posizionale)

$$464 + (4 * 3) = 476$$

- StringaV: semplicemente una stringa vuota.

L'ordine con cui sono posizionati le variabili nel *.data* non sono casuali, ma hanno una sequenza precisa soprattutto *myplaintext*(1°), *space*(2°) e infine *stringaV*(3°). Infatti, la variabile *space* serve per "allocare" maggior spazio per il *myplaintext*.

Vediamo nel dettaglio come avviene questo “allocazione” nella memoria. Gli elementi vengono inseriti nella memoria partendo dal basso verso l’alto.

```

4 .data
5 stringa_errore1: .string "Errore Mychiper: ALG #-che?"
6 stringa_errore2: .string "Errore Mychiper: cifratura non eseguibile per troppe C"
7 stringa_errore3: .string "Errore Myplaintext: troppo lungo da cifrare"
8 stringaMessaggio1: .string "ALG_A:"
9 stringaMessaggio2: .string "ALG_B:"
10 stringaMessaggio3: .string "ALG_C:"
11 stringaMessaggio4: .string "ALG_D:"
12 stringaMessaggio5: .string "La tua stringa e': "
13 stringa_Messaggio6: .string "ESECUZIONE DECIFRATURA"
14 mychiper: .string "CC"
15 k: .word 50
16 key: .string "OLE"
17 myplaintext: .string "Ciao!"
18 space: .string "
19 stringaV: .string ""

```

0x10000050		C		
0x1000004c	o	p	p	e
0x10000048	r		t	r
0x10000044	e		p	e
0x10000040	i	b	i	l
0x1000003c	s	e	g	u
0x10000038	o	n		e
0x10000034	r	a		n
0x10000030	r	a	t	u
0x1000002c		c	i	f
0x10000028	p	e	r	:
0x10000024	y	c	h	i
0x10000020	r	e		M
0x1000001c	E	r	r	o
0x10000018	h	e	?	
0x10000014		#	-	c
0x10000010		A	L	G
0x1000000c	p	e	r	:
0x10000008	y	c	h	i
0x10000004	r	e		M
0x10000000	E	r	r	o

Address	+0	+1	+2	+3
0x100000ac	g	a		e
0x100000a8	t	r	i	n
0x100000a4	u	a		s
0x100000a0	L	a		t
0x1000009c	D	:		
0x10000098	A	L	G	-
0x10000094	C	:		
0x10000090	A	L	G	-
0x1000008c	B	:		
0x10000088	A	L	G	-
0x10000084	A	:		
0x10000080	A	L	G	-
0x1000007c	a	r	e	
0x10000078	c	i	f	r
0x10000074		d	a	
0x10000070	u	n	g	o
0x1000006c	p	o		l
0x10000068	t	r	o	p
0x10000064	x	t	:	
0x10000060	i	n	t	e
0x1000005c	y	p	l	a
0x10000058	r	e		M
0x10000054	E	r	r	o

E così via fino ad arrivare a **Myplaintext** con il suo enorme spazio a disposizione

0x10000108				
0x10000104				
0x10000100				
0x100000fc				
0x100000f8				
0x100000f4				
0x100000f0				
0x100000ec				
0x100000e8				
0x100000e4				
0x100000e0				
0x100000dc	!			
0x100000d8	C	i	a	o

.text

Il codice che utilizza le variabili che abbiamo inizializzato in .data ed effettua gli algoritmi, gli troviamo in .text.

Notiamo che prima di iniziare ad eseguire il codice, ci troviamo di fronte alla **FASE DI BLOCCO ERRORI**.

Tenendo presente gli errori trattati nel .data abbiamo il seguente codice:

```
22 #FASE DI BLOCCO ERRORI
23
24 #controllo mychipper massimo 5 caratteri
25 la a0,mychipper
26 jal lunghezzaArray
27 la a1,stringa_errore4
28 li t0,6
29 bge a0,t0,errore_controllo
30 #controllo mychipper: disponibilità algoritmi
31 la a0,mychipper
32 jal controllo1
33 la a1,stringa_errore1
34 beq a0,zero,errore_controllo
35 #controllare myplaintext lunghezza < 100
36 la a0,myplaintext
37 jal lunghezzaArray
38 la a1,stringa_errore3
39 li t0,100
40 bge a0,t0,errore_controllo
41 #controllo per eseguire un tot di C
42 la a0,mychipper
43 jal controllo_conteggioC
44 beq a0,zero,finito_controllo
45 add s0,a0,zero
46 la a0,myplaintext
47 jal lunghezzaArray
48 jal controllo2
49 bge a0,s0,finito_controllo
50 la a1,stringa_errore2
51 j errore_controllo
52
53 finito_controllo:
```

Ogni pezzo di codice che elabora il controllo, prima di effettuare un salto condizionato memorizza il messaggio di errore, così se è presente qualcosa di anomalo ecco che salta verso *errore_controllo*, che ha semplicemente un codice di stampa.

```
823 errore_controllo:
824 li a0,4
825 ecall
```

Per quanto riguarda quelle fasi in cui si controlla semplicemente la lunghezza della stringa(sotto riportate), la loro composizione è abbastanza semplice:

1. Prendere la stringa interessata
2. Controllare la lunghezza
3. Scegliere il valore massimo
4. Salto condizionato

```
24 #controllo mychipper massimo 5 caratteri
25
26 la a0,mychipper
27 jal lunghezzaArray
28 la a1,stringa_errore4
29 li t0,6
30 bge a0,t0,errore_controllo
39 #controllare myplaintext lunghezza < 100
40
41 la a0,myplaintext
42 jal lunghezzaArray
43 la a1,stringa_errore3
44 li t0,100
45 bge a0,t0,errore_controllo
```

Dove *lunghezzaArray* è così definita:

```
538 #metodo per la lunghezza di una stringa
539 #lunghezzaArray(a0 = stringa)
540 lunghezzaArray:
541     li t0,0 #indice=0
542 loop_lunghezzaArray:
543     add t1,a0,t0
544     lb t2,0(t1)
545     beq t2,zero,end_loop_LunghezzaArray
546     addi t0,t0,1
547     j loop_lunghezzaArray
548
549 end_loop_LunghezzaArray:
550     add a0,t0,zero
551     jr ra
```

Gli ultimi errori da verificare sono:

- 1) Mychip: disponibilità algoritmi (**controllo1**)
- 2) Mychip: controllo C (**controllo_conteggioC**, **controllo2**)

1)

```
32 #controllo mychip: disponibilità algoritmi
33
34 la a0,mychip
35 jal controllo1
36 la a1,stringa_errore1
37 beq a0,zero,errore_controllo

778 #controllo1(a0 = stringa)
779
780 controllo1:
781 li t0,0 # contatore
782 li t1,65 # A
783 li t2,69 # E
784 li t3,1 #valore booleano TRUE
785
786 loop_controllo1:
787 add t4,t0,a0
788 lb t5,0(t4)
789 beq t5,zero,end_controllo1
790 blt t5,t1,trovato_errore1
791 bge t5,t2,trovato_errore1
792 addi t0,t0,1
793 j loop_controllo1
794 trovato_errore1:
795 li t3,0 #FALSE
796 end_controllo1:
797 add a0,t3,zero
798 jr ra
```

In alto livello possiamo considerare *controllo1*, scritto in questa maniera.
(linguaggio utilizzato python)

```
def controllo1(stringa):
    indice = 0
    lettera_iniziale= 65
    lettera_finale = 69
    esito = True
    while(indice < len(stringa)):
        lettera = ord(stringa[indice])
        if lettera >= lettera_iniziale and lettera < lettera_finale:
            indice = indice +1
        else:
            esito = False
            return esito
    return esito
```

In python il comando “ord” serve per trasformare la lettera nel suo rispettivo codice ASCII

2)

```
47 #controllo per eseguire un tot di C
48
49 la a0,mychipper
50 #controllo_conteggioC(a0 = stringa)
51 jal controllo_conteggioC
52 beq a0,zero,finito_controllo
53 add s0,a0,zero #salvo conteggio per confronto
54 la a0,myplaintext
55 #lunghezzaArray(a0 = stringa)
56 jal lunghezzaArray
57 #controllo2(a0 = int_lunghezzaArray)
58 jal controllo2
59 bge a0,s0,finito_controllo
60 la a1,stringa_errore2
61 j errore_controllo
62
63 finito_controllo:
```

Nel dettaglio *controllo_conteggioC*:

```
818 #conteggio C = conta quanti C sono presenti nella stringa
819 #controllo_conteggioC (a0 = stringa)
820
821 controllo_conteggioC:
822 li t0,0
823 li t1,67
824 li t2,0 #count_C
825 controllo2_zero_loop:
826 add t3,a0,t0 #indirizzo
827 lb t4,0(t3)
828 beq t4,zero,end_loop_controllo2_zero
829 bne t4,t1,salta_controllo2_zero
830 addi t2,t2,1
831 salta_controllo2_zero:
832 addi t0,t0,1
833 j controllo2_zero_loop
834 end_loop_controllo2_zero:
835 add a0,t2,zero
836 jr ra
```

Alto livello (usato linguaggio Python):

```
def controllo_conteggioC(stringa):
    indice = 0
    lettera_C = 67
    count_C = 0
    while indice < len(stringa):
        lettera_stringa = ord(stringa[indice])
        if lettera_stringa == lettera_C:
            count_C = count_C + 1
        indice = indice + 1
    return count_C
```

Una volta che in *controllo_conteggioC* ho contato quanti "C" erano presenti nel mychip, devo controllare se il numero di "C" coincide con quello che effettivamente posso eseguire. Per sapere quanti "C" possono essere eseguiti chiamo il metodo *controllo2*.

Nel dettaglio *controllo2*:

```
803 #controllo2 (a0 = int lunghezzaArray)
804 controllo2:
805 li t0,1 #contatore = 1, perché almeno 1 può farlo
806 li t1,100 # i 100 caratteri che non devono essere superati
807 add t2,a0,zero
808 again:
809 slli t3,t2,2 #t2*4
810 addi t4,t2,-1 #(t2-1)
811 add t2,t3,t4
812 bge t2,t1,end_loop_controllo2
813 addi t0,t0,1
814 j again
815 end_loop_controllo2:
816 add a0,t0,zero
817 jr ra
```


In alto livello (linguaggio usato python):

```
def controllo2(int_lunghezzaArray):  
    count = 1  
    valore_max = 100  
    #una specie di do-while  
    while True:  
        fase_mul = int_lunghezzaArray * 4  
        fase_sub = int_lunghezzaArray - 1  
        int_lunghezzaArray = fase_mul + fase_sub  
        if int_lunghezzaArray > valore_max:  
            break  
        count = count + 1  
    return count
```

Una volta che abbiamo controllato che non ci sono possibili errori nel codice, possiamo passare alla *FASE DI RICAPO SPAZIO*.

La **FASE DI RICAPO SPAZIO**, è molto piccola e semplice, in quanto consiste nel cancellare il contenuto della variabile *.space* che troviamo in *.data*. Infatti *.space* è una variabile che contiene spazi vuoti, in cui il codice ASCII è uguale a 32. Quindi in memoria non avremmo tecnicamente degli spazi liberi su cui scrivere. Ecco come *.space* si presenta in codice ascii
(secondo gli elementi che troviamo nell'immagine *.data*)

Address	+0	+1	+2	+3
0x10000160				
0x1000015c				
0x10000158				
0x10000154				
0x10000150				
0x1000014c				
0x10000148				
0x10000144				
0x10000140				
0x1000013c				
0x10000138				
0x10000134				
0x10000130				
0x1000012c				
0x10000128				
0x10000124				
0x10000120				
0x1000011c				
0x10000118				
0x10000114	!			
0x10000110	C	i	a	o
0x1000010c	O	L	E	
0x10000108	2			

Display type: ASCII
Go to: Select
Up
Down
Save Address

Il colore azzurro è dove inizia il .space, come si vede sembra vuoto(ASCII), ma in realtà è così:

0x10000138	32	32	32	32
0x10000134	32	32	32	32
0x10000130	32	32	32	32
0x1000012c	32	32	32	32
0x10000128	32	32	32	32
0x10000124	32	32	32	32
0x10000120	32	32	32	32
0x1000011c	32	32	32	32
0x10000118	32	32	32	32
0x10000114	33	0	0	0
0x10000110	67	105	97	111
0x1000010c	79	76	69	0
0x10000108	50	0	0	0

Display type: Decimal Go to: Select Up Down Save Address

Il codice è il seguente:

```

64 #FASE DI RICAVO SPAZIO
65 la a0,space
66 jal lunghezzaArray
67 add s3,a0,zero # salvo la lunghezza,usata in cancella
68 la a0,space
69 jal cancella

```

Ho provato ad inserire a0 in a1, ma genera errore. Quindi ho optato un registro di salvataggio

Nel dettaglio *cancella*:

```

754 #algoritmo usato per entrambi
755 #cancella(a0 = space,s3 = lunghezza salvata)
756
757 cancella:
758 li t0,0 #indice
759 add t1,zero,zero #valore che serve per cancellare
760 cancella_loop:
761 bge t0,s3,cancella_end_loop
762 add t2,t0,a0
763 sb t1,0(t2)
764 addi t0,t0,1
765 j cancella_loop
766 cancella_end_loop:
767 jr ra

```

In alto livello (linguaggio utilizzato python):

#Di regola, in Risc lo zero dovrebbe rappresentare null

```
def cancella(stringa, lunghezza):  
    indice = 0  
    cancellino = 0 #zero per risc  
    while indice < lunghezza:  
        stringa[indice] = cancellino  
        indice = indice +1  
    return stringa
```

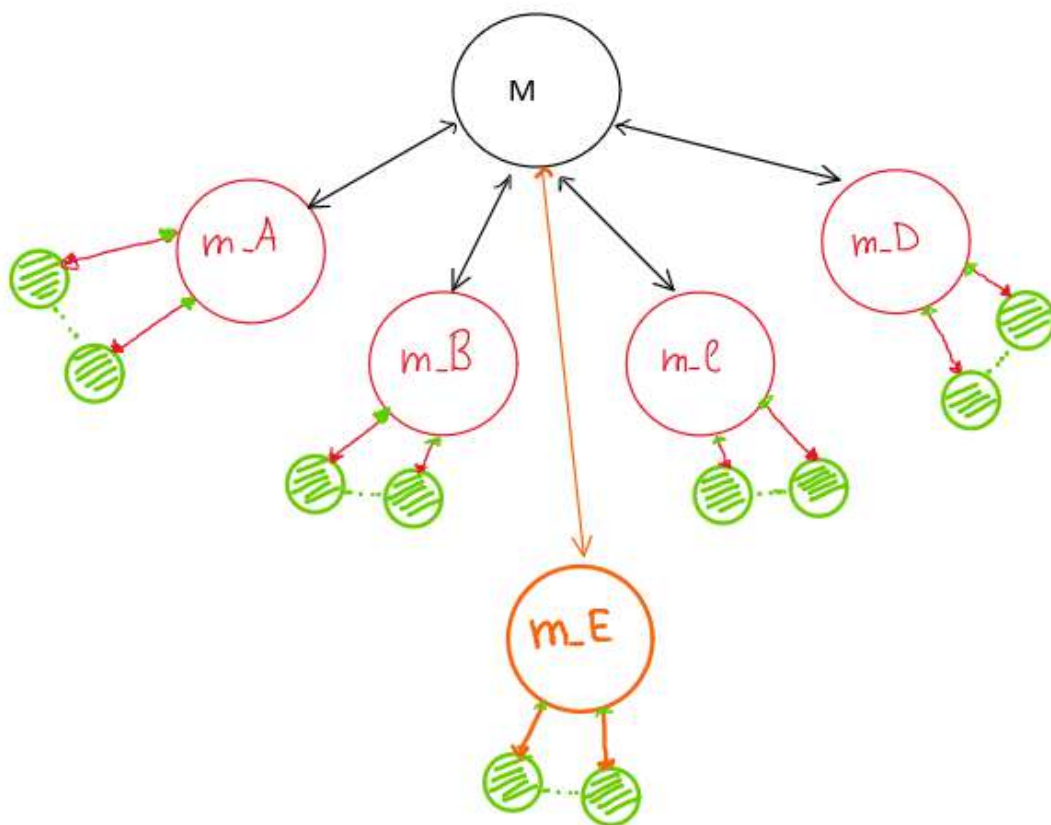
INIZIO PROGRAMMA

La struttura della fase di esecuzione del codice possiamo vederla in questa maniera:

- 1) Un main-centrale che richiama i main-algoritmi
- 2) Main-algoritmi che chiamano i loro metodi per poi ritornare al main-centrale

Questi due procedimenti vengono utilizzati sia per la cifratura e sia per la decifratura.

Graficamente si presenta in questo modo:



Il colore delle frecce indica quale stringa viene modificata

Mychipiper

Myplaintext

Chiamata metodi

I pallini verdi sono i metodi che vengono eseguiti dal main-algoritmi, il loro numero non corrisponde al numero di funzioni utilizzati da quest'ultimi. Possiamo notare dall'immagine come il m_E ha un colore diverso dagli altri, in quanto non modifica il myplaintext ma viene passata come stringa il mychipiper, usata appunto per la decifratura.

FASE DI CIFRATURA&DECIFRATURA

Intro:

```
71 #FASE CIFRATURA
72 #Stampa Intro
73 la a1,stringaMessaggio5
74 li a0,4
75 ecall
76 li a1,13
77 li a0,2
78 ecall
79 la a1,myplaintext
80 li a0,4
81 ecall
82 li a1,13
83 li a0,2
84 ecall
```

È un codice che presenta la stringa che vogliamo cifrare.

Dopo la presentazione della stringa, inizia la fase di preparazione, dove carico nei vari registri, i valori che servono per il funzionamento del codice.

Codice:

```
85 main:
86
87 li s0,0 #indice di mychipiper
88 lw s1,k
89 li s2,0 #segnale per cifratura e decifratura
90
91 loop:
92 la a0,mychipiper
93 li t0,65 #A
94 li t1,66
95 li t2,67
96 li t3,68
97
```

Il codice sembrerebbe abbastanza comprensibile, solo che è preferibile accennare qualcosa sui primi tre valori $s0, s1, s2$.

Durante tutto il codice, questi tre valori non vengono toccati, in quanto sono indispensabili.

Per quanto riguarda $s0$, come evidenziato sopra, essendo l'indice del mychip la sua modifica causerebbe possibili errori.

Mentre i registri $s1$ e $s2$ sono importanti sia per la fase di cifratura e sia per decifratura.

Il registro $s1$ ha la sua importanza nel caso sia presente nel mychip l'algoritmo A, e vedremo come $s1$, nel caso della decifratura, viene modificato.


Il registro $s2$ possiamo considerarlo come una variabile booleana, che serve al codice per capire quando ci troviamo nella fase di cifratura e decifratura. Dato che è un valore booleano, significa che può assumere soltanto due valori:


- 1) $S2 = 0$ eseguo cifratura
- 2) $S2 = 1$ eseguo decifratura.

In quest'ultima fase abbiamo il caricamento del carattere i -esimo appartenente al mychip dove $0 < i \leq 5$. Una volta prelevata quest'ultima viene confrontata con tutti i possibili algoritmi che sono stati sviluppati. In caso di uguaglianza viene richiamato l'algoritmo interessato.

Continuo codice:

```
98 add t5,s0,a0
99 lb t6,0(t5)
100 #fase di controllo
101 beq s2,zero,decifratura_false
102 beq t6,zero,end_main
103 decifratura_false:
104 beq t6,zero,decifratura
105 beq t6,t0,algoritmo_A
106 beq t6,t1,algoritmo_B
107 bne t6,t2,continua
108 beq s2,zero,cifraturaC
109 j algoritmo_C_decifratura
110 cifraturaC:
111 j algoritmo_C
112 continua:
113 beq t6,t3,algoritmo_D
```

Utilizzo $s2 \rightarrow$ 

Utilizzo $s1 \rightarrow$ 

```

115 decifratura:
116 #Per la decifratura
117 la a1,stringa_Messaggio6
118 li a0,4
119 ecall
120 li a1,13
121 li a0,2
122 ecall
123 la a0,mychipper
124 sub s1,zero,s1 #-k
125 li s0,0 #l'indice viene ripristinato
126 li s2,1 #decifratura true
127 j algoritmo_E

```

Osservazione: notiamo come s2 ci permette di uscire dal ciclo, infatti nel momento in cui assume il valore 1 esegue la riga 102 (beq t6,zero,end_main)

CIFRATURA CESARE

[testo]

L'algoritmo A ha come input un testo da cifrare, nel nostro caso *myplaintext*, e una costante *K*. Dove sommo ad ogni carattere del testo il valore *k*. Le maiuscole e le minuscole vengono preservate, mentre i caratteri che non fanno parte dell'alfabeto rimangono intatte.

Esempio:

myplaintext = "Ciao!" e *k*=1, il nuovo myplaintext = "Djbp!". Perché tenendo presente la tabella ASCII abbiamo che C+1=D, i+1=j

codice Risc-V:

L'inizio è molto banale, prende i valori iniziali e finali che definiscono gli intervalli dell'alfabeto da considerare (maiuscolo e minuscolo); la scelta di aver preso gli estremi superiori + 1 (come 123,91) servono soprattutto per eseguire una condizione di minore stretto (carattere < 123)

```

129 algoritmo_A:
130
131 la s3,myplaintext
132 li s4,97 #a
133 li s5,123 # z+1
134 li s6,65 #A
135 li s7,91 # Z+1
136
137 li t0,0

```

Una volta inizializzato, prendo il carattere del myplaintext e controllo se la lettera è maiuscola o minuscola.

```

139 loop_A:
140 add t1,t0,s3
141 lb a1,0(t1)
142 beq a1,zero,end_loop_A
143 blt a1,s7,maiusc
144 blt a1,s5,minusc
145 j aggiorna

```

Nel caso a1 (carattere) è una *possibile* maiuscola allora esegue:

```

147 maiusc:
148 blt a1,s6,aggiorna #controllo se il carattere appartiene all'intervallo
149 add a0,zero,s1 #lw a0,k sostituisce
150 add a2,s6,zero
151 add a3,s7,zero
152 jal miniciclo
153 sb a0,0(t1)
154 j aggiorna

```

Una volta che si conferma che il carattere appartiene all'intervallo delle maiuscole (blt a1,s6,aggiorna == false) il codice prosegue.

Prima di chiamare il metodo “miniciclo” prende i parametri necessari (a0 = k, a1=carattere, a2 = estremo inferiore intervallo,a3= estremo superiore intervallo)

Nel momento in cui a1 (carattere) è definita come una *possibile* minuscola, l'esecuzione sarà il seguente:

```

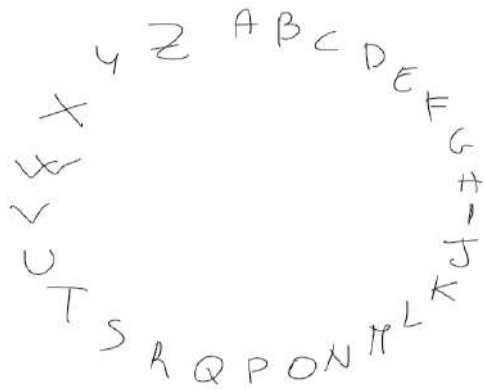
156 minusc:
157 blt a1,s4,aggiorna
158 add a0,zero,s1 #lw a0,k
159 add a2,s4,zero
160 add a3,s5,zero
161 jal miniciclo
162 sb a0,0(t1)
163 j aggiorna

```

I valori cambiamo, ma il procedimento coincide con la spiegazione precedente.

Adesso definiamo il metodo che viene chiamato, “miniciclo”.

Prima di tutto dobbiamo pensare all'intervallo considerato come se fosse un cerchio. Infatti, i due intervalli, caratteri maiuscoli e caratteri minuscoli, possono essere rappresentati in questa maniera (esempio Maiuscole):



Perché se immaginiamo l'intervallo in maniera lineare, nel momento in cui k assume un valore abbastanza grande da permettergli di uscire dall'intervallo, il carattere cifrato non assumerebbe il valore desiderato.

[codice]

```
557 #metodo per algoritmo A
558 #a0 = k, a1 = valore_carattere, a2= estremo_inf, a3 = estremo_sup
559 miniciclo:
560 add t2,a1,a0
561
562 loop_miniciclo_inf:
563 bge t2,a2,loop_miniciclo_sup
564 sub t2,a2,t2
565 sub t2,a3,t2
566 j loop_miniciclo_inf
567
568 loop_miniciclo_sup:
569 blt t2,a3,fine
570 sub t2,t2,a3
571 add t2,a2,t2
572 j loop_miniciclo_sup
573
574 fine:
575 add a0,t2,zero
576 jr ra
```

Dopo aver eseguito la somma (add t2,a1,a0), l'algoritmo si divide in:

- 1) Loop_miniciclo_inf nel caso la somma è inferiore all'estremo inferiore
- 2) Loop_miniciclo_sup nel caso la somma è superiore all'estremo superiore

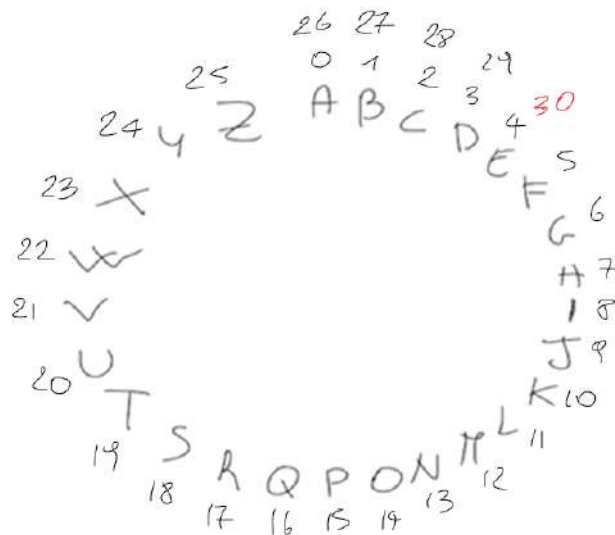
In alto livello sarebbe[codice usato python]:

```
def miniciclo(k, valore_carattere, estremo_inf, estremo_sup):  
    temp = k + valore_carattere  
    while(temp < estremo_inf):  
        temp = estremo_sup - estremo_inf - temp  
  
    while(temp >= estremo_sup):  
        temp = estremo_inf + (temp - estremo_sup)  
  
    return temp
```

Facciamo un **esempio** di esecuzione grafica utilizzando come intervallo le lettere maiuscole:

Dati:

$k = 30$, $\text{valore_carattere} = A(65)$, $\text{estremo_inf} = 65$, $\text{estremo_sup} = Z+1(91)$



Alla fine, ci stamperà E.

Osservazione: notiamo come la lettera A parte da zero.

In caso di decifratura, viene richiamato lo stesso codice, solo che gli viene passata -k

CIFRARIO A BLOCCHI

[Testo]

Dato un testo da cifrare (myplaintext) e una chiave (key), il cifrario a blocchi fa in modo che ogni elemento del myplaintext viene cifrato sommando la codifica ASCII di ogni suo carattere con quello della chiave come segue:

$$\{[(\text{cod}(\text{valoreLettera}_{i\text{-esima}}) - 32) + (\text{cod}(\text{Key}_{j\text{-esima}}) - 32)] \% 96\} + 32$$

Dove $0 \leq i \leq \text{lunghezza}(\text{myplaintext})$ e $0 \leq j \leq \text{lunghezza}(\text{key})$

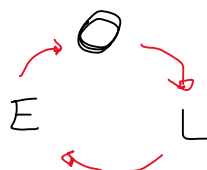
Inizio codice:

```
189 algoritmo_B:
190
191 #inizio algoritmo
192 li s3,32
193 li s4,128
194 la s5,myplaintext
195 la a0,key
196 #a0 = indirizzo stringa
197 jal lunghezzaArray
198
199
200 add s6,a0,zero #salvo in s6 la lunghezza
201 li s7,0 #indice per key
202
203 #Fase Algoritmo
204 li t0,0 #per il loop
```

In questa prima fase carico i valori che verranno utilizzati in seguito per l'algoritmo B. Da tener presente come (197)lunghezzaArray ha bisogno come parametro l'indirizzo di key e non del myplaintext, che a sua volta viene salvato in s6(200).

I registri s6 e s7 sono importanti per chiarire l'elemento key_i-esimo da utilizzare. Infatti per la key, si utilizzerà la stessa tecnica che abbiamo usato per l'algoritmo A. Quindi la Key può essere immaginata in questa maniera.

[figura key]



Dopo questa fase di preparazione inizia l'esecuzione della cifratura a blocchi:

```
203 #Fase Algoritmo
204 li t0,0 #per il loop
205
206 loop_algoritmo_blocchi:
207     add t1,t0,s5
208     lb t2,0(t1)
209     beq t2,zero,end_loopB
210     blt t2,s3,salta
211     bge t2,s4,salta
212     la a0,key
213     add a1,s7,zero #carico l'indice
214     jal ciclo_key
215     # a0 valore key[indice]
216     add s7,a1,zero
217     add a1,t2,zero #carico valore
218     #fase decisionale tra cifratura e decifratura
219     beq s2,zero,esegui_formula
220     jal formula_decifratura
221     j avanti
222 esegui_formula:
223     jal formula
224 avanti:
225     sb a0,0(t1)
226     addi s7,s7,1
227 salta:
228     addi t0,t0,1
229
230     j loop_algoritmo_blocchi
```

Vedendo questo codice è facilmente intuibile che è diviso in 3 parti:

- 1) Controllo se t2(valore lettera) appartiene all'intervallo, semplicemente se $t2 < s3$ and $t2 \geq s4$.
- 2) Preparo i parametri a0,a1 per dopo mandarli in (214)ciclo_key
- 3) Verifico se eseguire la cifratura o la decifratura. Notare come questo viene deciso dal registro s2 visto precedentemente.

Una volta che ho determinato i valori, quest'ultimi vengono salvati nel mychipер.

I metodi che vengono chiamati nell'algoritmo B sono ciclo_key, formula, formula_decifrata.

In alto livello possiamo tradurlo più o meno così [linguaggio usato python]:

```
while i < len(stringa):
    temp = ord(stringa[i])
    if temp > 32 and temp < 127:
        temp2 = ord(cicloKey(indice, key))
        if boolCifratura == true:
            temp = chr(formula_conversione(temp, temp2))
        else:
            temp = chr(formula_decifratura(temp, temp2))
    risultato.append(temp)
    i = i+1
    indice[0] = indice[0]+1
```

Ciclo_key:

```
577 #metodi per algoritmi B
578
579 ciclo_key:
580 #passati come parametri a0 indirizzo key, a1 indice
581     bne a1,s6,salta_istruzione
582     li a1,0
583 salta_istruzione:
584     add t3,a0,a1
585     lb a0,0(t3)
586     jr ra
```

Nel momento in cui l'indice è uguale alla massima lunghezza, l'indice viene inizializzato.

Quindi in alto livello può essere scritto in questa maniera:

```
def cicloKey(indice, key):
    if indice[0] == len(key):
        indice[0] = 0
    return key[indice[0]]
```

Formula:

```
588 #a0 = valore_key, a1 = valore_lettera
589 formula:
590     addi sp,sp,-4
591     sw ra,0(sp)
592     sub t2,a0,s3
593     sub t3,a1,s3
594     add a0,t2,t3
595     jal modulo
596     add a0,a0,s3
597     lw ra,0(sp)
598     addi sp,sp,4
599     jr ra
```

La “novità” che troviamo in questo metodo, che poi verrà utilizzato in seguito negli altri algoritmi in maniera più assidua, è l’istruzione “addi sp,sp,-4”.

Precisamente questa istruzione alloca nello stack uno spazio, affinché può salvare al suo interno il valore di ritorno (ra) del main. Il suo utilizzo è fondamentale, perché senza memorizzare il suo ra, avremmo un loop infinito in quanto il codice non tornerebbe nel main.

Il metodo esegue precisamente questo codice in python:

```
def formula_conversione(valore_stringa, valore_key):  
    return (((valore_stringa-32)+(valore_key-32))%96)+32
```

Ovviamente per eseguire l’operazione “%96” esegue “jal modulo”

Modulo:

```
624 modulo:  
625 li t3,96  
626 div t4,a0,t3  
627 mul t4,t4,t3  
628 sub a0,a0,t4  
629 jr ra
```

Mentre nella fase di decifratura viene utilizzato come metodo “formula_decifratura”:

```
600 #stessi parametri di formula  
601 formula_decifratura:  
602 addi sp,sp,-4  
603 sw ra,0(sp)  
604 add t2,a0,zero #conservo Valore Key  
605 sub a0,a1,s3 #valore_stringa-32  
606 jal modulo  
607 add t5,zero,a0  
608 sub a0,t2,s3  
609 jal modulo  
610 add t3,zero,a0  
611 sub t4,t5,t3  
612 bge t4,zero,positivo  
613 addi t4,t4,128  
614 j finish  
615 positivo:  
616 add t4,t4,s3  
617 finish:  
618 lw ra,0(sp)  
619 addi sp,sp,4  
620 add a0,t4,zero  
621 jr ra
```

Alto livello[linguaggio usato python]:

```
def formula_decifratura(valore_stringa, valore_key):  
    risultato = (valore_stringa-32)%96-(valore_key-32)%96  
    if risultato > 0:  
        risultato = risultato+32  
    else:  
        risultato = risultato+128  
    return risultato
```

CIFRARIO AD OCCORENZE

[testo preso dalla spiegazione del progetto]

A partire dal primo carattere del plaintext (alla posizione 1), **il messaggio viene cifrato come una sequenza di stringhe separate da esattamente 1 spazio** (ASCII 32) in cui ciascuna stringa ha la forma “ $x-p_1\ldots-p_k$ ”, dove x è la prima occorrenza di ciascun carattere presente nel messaggio, $p_1\ldots p_k$ sono le k posizioni in cui il carattere x appare nel messaggio (con $p_1 < \ldots < p_k$), ed in cui ciascuna posizione è preceduta dal carattere separatore ‘-’ (per distinguere gli elementi della sequenza delle posizioni).

Note:

- non c’è un ordine prestabilito per le lettere una volta codificata la stringa
- la codifica usa due separatori: lo spazio (ASCII 32) ed il trattino (ASCII 45). Ciò che sta tra due trattini deve essere sempre un numero che indica l’occorrenza di una lettera nella stringa di base, mentre ciò che segue lo spazio è sempre il carattere di riferimento, eccetto il caso in cui il carattere di riferimento sia lo spazio in sé (si veda l’esempio sotto).
- il cyphertext ottenuto con questa codifica ha generalmente una lunghezza maggiore del plaintext di partenza

Esempio

Pt = “sempio di messaggio criptato -1”

La cifratura con questo algoritmo produrrà un cyphertext ct = “e-2-12 s-1-13-14 m-3-11 p-4-24 i-5-9-18-23 o-6-19-28 -7-10-20-29 d-8 a-15-26 g-16-17 c-21 r-22 t-25-27 --30 1-31”.

L’algoritmo C, è un algoritmo molto lungo e complesso. A differenza degli altri algoritmi che salvano direttamente la codifica giusta nel myplaintext, il cifrario ad occorrenze fa uso della variabile *stringaV* (una stringa vuota), accennata quando abbiamo parlato del *.data*.

Prima di parlare direttamente del codice, vediamo come funziona graficamente. Per farlo prendiamo un semplice esempio.

Esempio:

myplaintext = “ciaoo!” stringaV= “”

1-fase

Myplaintext = "\$iaoo!" stringaV = "c-1 "

2-fase

Myplaintext = "\$\$aoo!" stringaV = "c-1 i-2 "

3-fase

Myplaintext = "\$\$\$oo!" stringaV = "c-1 i-2 a-3 "

4-fase

Myplaintext = "\$\$\$\$\$!" stringaV = "c-1 i-2 a-3 o-4-5 "

5-fase

Myplaintext = "\$\$\$\$\$\$" stringaV = "c-1 i-2 a-3 o-4-5 !-6"

Da precisare come "\$", è un valore usato nell'esempio per capire come avviene il funzionamento dell'algoritmo. Infatti, il "\$" viene utilizzata per marcare i caratteri che sono stati già presi in considerazione, così da evitare di riprendere le stesse lettere.

Inizio codice:

```
252 #CIFRATURA
253 algoritmo_C:
254 la a0,stringaV #indirizzo SV
255 la a1,myplaintext #indirizzo stringa
256 li s3,45 # -
257 li s4,32 # spazio
258 li s5,0 #contatore primo loop
259 li s6,0 #contatore stringa_vuota ( nel main)
260 li s7,31 #valore usato per marcare
```

Continuo codice:

```
263 loop1:
264 add t0,a1,s5 #indirizzo
265 lb a1,0(t0) #lettera_myplaintext
266 lb s8,1(t0) #lettera_myplaintext successiva
267 beq a1,zero,end_loop
268 beq a1,s7,aggiornamento
269 add a2,s6,zero #inserisco il contatore
270 jal inserisci_carattere
271 addi s6,s6,1 #aumento il contatore
272 add a2,a1,zero
273 la a1,myplaintext
274 #a0 stringaV #a1 stringa #a2 carattere da cercare
275 jal inserisci_posizione
276 add a2,zero,a0
277 #inserimento spazio
278 beq s8,zero,aggiornamento
279 la a0,stringaV
280 add a1,zero,s4
281 jal inserisci_carattere
282 addi s6,a2,1
283 aggiornamento:
284 addi s5,s5,1
285 la a0,stringaV
286 la a1,myplaintext
287 j loop1
```


Questo codice prende i caratteri che non sono marcati e gli inserisce nella stringaV, e poi inizia a ricercare gli stessi nelle varie posizioni.

I metodi che possiamo intravedere in questa immagine sono *inserisci_carattere* e *inserisci_posizione*.

Metodo *inserisci_carattere*:

```
672 #Algoritmi C cifratura
673 #a0 = l'indirizzo, a2 = indice, a1 = carattere
674 inserisci_carattere:
675 add t3,a0,a2 #aggiornamento stringaV
676 sb a1,0(t3)
677 jr ra
```

Il metodo *inserisci_posizione* serve, come dice la parola, cercare le varie posizioni che l'elemento occupa nel myplaintext.

Dato che è molto lungo, lo divideremo a metà.

Prima Parte:

```
679 inserisci_posizione:
680 addi sp,sp,-20 # 5 posizioni
681 sw ra,16(sp)
682 li t0,0 #contatore stringa
683 add t1,s6,zero #contatore stringaV
684 loop_posizione:
685 add t2,t0,a1 #stringa[i]
686 lb t3,0(t2)
687 beq t3,zero,end_loop2
688 beq t3,s7,aggiornamento_loop
689 bne t3,a2,aggiornamento_loop
690 sb s7,0(t2) #marcatura
691 add a1,s3,zero #trattino
692 sb a2,12(sp) #carico il carattere
693 add a2,t1,zero #contatore
694 #a0 stringaV #a1 trattino #a2 contatore
695 jal inserisci_carattere
696 addi t1,t1,1
697 sw t0,8(sp)
698 sw t1,4(sp)
699 sw a0,0(sp)
```

Nello stack dovremmo avere una cosa del genere

INDIRIZZO_STRINGAV	0
INDICE_STRINGAV	4
INDICE_MYPLAINTEXT	8
CARATTERE	12
RA	16

Seconda Parte:

```
700 add a0,t0,zero
701 #a0 indice da trasformare in numero
702 addi a0,a0,1 #perche' vuole da 1
703 jal numero_stringa
704 add t0,a0,zero
705 add t1,a1,zero
706 lw a0,0(sp) #indirizzo stringa vuota
707 lw a2,4(sp) #contatore stringa vuota
708 li t2,48
709 beq t0,t2,no_decimale
710 add a1,t0,zero #valore decimale
711 #a0 stringaV #a1 valore intero decimale #a2 contatore stringa vuota
712 jal inserisci_carattere
713 addi a2,a2,1 #contatore stringa vuota
714 no_decimale:
715 #inserimento forse di a2 non serve
716 add a1,t1,zero
717 #a0 stringaV #a1 valore numerico #a2 contatore
718 jal inserisci_carattere
719 addi t1,a2,1
720 lw t0,8(sp)
721 lb a2,12(sp) #carattere
722 la a1,myplaintext
723 aggiornamento_loop:
724 addi t0,t0,1
725 j loop_posizione
726
727 end_loop2:
728 lw ra,16(sp)
729 addi sp,sp,20
730 add a0,t1,zero
731 jr ra
```

In alto livello può essere tradotto in questa maniera:

```
def inserisci_posizione(stringaV,stringa,carattere):
    count_s = 0
    #count_sV = s6 è inutile in python
    while(count_s < len(stringa)):
        carattere_stringa = ord(stringa[count_s])
        if carattere_stringa != 31 and carattere_stringa == ord(carattere):
            stringa[count_s] = 31 #marcatura
            stringaV = stringaV+"-"
            decimale = numero_stringa(count_s+1) #array
            if decimale[0] == 48:
                stringaV = stringaV+chr(decimale[1])
            else:
                stringaV = stringaV+chr(decimale[0])+chr(decimale[1])
        count_s = count_s+1
```

Questa funzione a sua volta, utilizza un altro metodo chiamato *stringa_numero*. Quest'ultimo permette di tradurre la posizione del carattere, in un numero secondo la codifica ascii.

```
733 #a0 = posizione
734 numero_stringa:
735 li t0,48
736 li t1,48 #valore decimale
737 li t2,0 #count
738 li t3,9
739 loop_while:
740 bge t2,a0,end_while
741 bne t2,t3,else
742 li t2,0
743 addi t1,t1,1
744 li t0,48
745 addi a0,a0,-10
746 j loop_while
747 else:
748 addi t0,t0,1
749 addi t2,t2,1
750 j loop_while
751 end_while:
752 add a0,t1,zero #valore decimale
753 add a1,t0,zero
754 jr ra
```

Alto livello:

```
def numero_stringa(numero):
    temp0 = 48 # zero
    temp1 = 48 # zero
    count = 0
    risultato = []
    while count < numero:
        if count == 9:
            count = 0
            temp1 = temp1 + 1
            temp0 = 48
            numero = numero - 10
        else:
            temp0 = temp0 + 1
            count = count + 1
        risultato.append(temp1)
        risultato.append(temp0)
    return risultato
```

Ovviamente quello che abbiamo fatto è modificare, **non** il myplaintext, ma la stringaV. Per questo motivo il nostro obiettivo rimane di inserire la cifratura inserita nella stringaV nel myplaintext, per dopo essere utilizzata negli altri algoritmi.

Parte finale:

```
290 end loop:
291 #qui inizia il passaggio da stringaV a myplaintext
292 la a0,stringaV
293 la a1,myplaintext
294 jal conversione
295 la a0,stringaV
296 jal lunghezzaArray
297 add s3,a0,zero
298 la a0,stringaV
299 jal cancella
300 la a1,stringaMessaggio3
301 li a0,4
302 ecall
303 li t0,13
304 add a1,t0,zero
305 li a0,2
306 ecall
307 la a1,myplaintext
308 li a0,4
309 ecall
310 li a1,13
311 li a0,2
312 ecall
313 addi s0,s0,1
314 j loop
```

I metodi che utilizzati sono “conversione” e “cancella”. Il primo permette il passaggio da stringaV a myplaintext, mentre il secondo cancella la stringaV per poi essere riutilizzata.

Il metodo *conversione*:

```
772 conversione:
773 li t0,0
774 ciclo_conversione:
775 add t1,t0,a0 #indirizzo stringaV
776 add t2,t0,a1 #indirizzo myplaintext
777 lb t3,0(t1) #valore stringaV
778 beq t3,zero,end_ciclo_conversione
779 sb t3,0(t2)
780 addi t0,t0,1
781 j ciclo_conversione
782 end_ciclo_conversione:
783 jr ra
```

Il metodo *cancella*:

```
758 #cancella(a0 = space,s3 = lunghezza salvata)
759
760 cancella:
761 li t0,0 #indice
762 add t1,zero,zero #valore che serve per cancellare
763 cancella_loop:
764 bge t0,s3,cancella_end_loop
765 add t2,t0,a0
766 sb t1,0(t2)
767 addi t0,t0,1
768 j cancella_loop
769 cancella_end_loop:
770 jr ra
```

CIFRARIO AD OCCORRENZE-Decifratura

Inizio:

```
317 algoritmo_C_decifratura:
318
319 la a0,myplaintext
320 li s3,0 #indice per l'intero loop
321 li s4,32
322 li s5,45
323 jal lunghezzaArray
324 add s6,a0,zero
```

Esecuzione:

```
326 loop_esterno:
327 la a0,myplaintext
328 add t0,s3,a0 #indirizzo
329 lb s7,0(t0)
330 beq s7,zero,end_decifraturaC
331 addi s3,s3,1
332 loop_interno:
333 jal definisci_numero # ritorna a0 = valore and a1=indice
334 addi s3,s3,1
335 beq a0,s4,loop_esterno
336 beq a0,s5,update
337 #INIZIA A DEFINIRE IL NUMERO
338 jal dammi_il_numero
339 add s3,a1,zero
340 la t0,stringaV
341 add t1,a0,t0 #stringaV[valore]
342 sb s7,0(t1)
343 update:
344 beq s3,s6,loop_esterno #istruzione da aggiustare con len(stringa)
345 la a0,myplaintext
346 j loop_interno
```

Come nella cifratura, il carattere cifrato viene inserito nella stringaV.

I metodi che vengono utilizzati all'interno dell'esecuzione sono "*definisci_numero*" e "*dammi_il_numero*".

Il primo serve per capire che tipo di numero deve essere trasformato in un indice posizionale, quindi può essere decimale o non.

Il secondo permette questa trasformazione in posizione.

- Il metodo `definisci_numero`:

```

348 #definisci_numero(a0 = indirizzo)
349 definisci_numero:
350 addi sp,sp,-8
351 sw ra,4(sp)
352 add t0,s3,zero #metto l'indice in una variabile temporanea
353 add t1,t0,a0 #indirizzo
354 lb a1,1(t1) #temp
355 lb a0,0(t1) #valore
356 beq a0,s5,passa_oltre
357 beq a0,s4,passa_oltre
358 sb a0,0(sp)
359 add a0,a1,zero
360 jal boolean_decimal
361 beq a0,zero,no_decimal
362 lb a0,0(sp) #ripristina valore di a0
363 j passa_oltre
364 no_decimal:
365 lb a1,0(sp) #ripristino a0
366 li a0,48 #dato che a0 e' il valore decimale, ho posto a0 = 48 cioe' zero
367 passa_oltre:
368 lw ra,4(sp)
369 addi sp,sp,8
370 jr ra

```

Al suo interno viene utilizzato il metodo *“boolean_decimal”*:

```

372 #a0 = valore
373 boolean_decimal:
374 li t0,1
375 li t1,45
376 li t2,32
377 beq a0,t1,cambia
378 beq a0,t2,cambia
379 beq a0,zero,cambia
380 j passa_avanti
381 cambia:
382 li t0,0
383 passa_avanti:
384 add a0,t0,zero
385 jr ra

```

Questi due metodi possono essere tradotti in alto livello in questa maniera

```
def boolean_decimal(valore):
    boolean = True
    trattino = 45
    spazio = 32
    if valore == trattino or valore == spazio or valore == None:
        boolean = False
    return boolean

def definisci_numero(stringa, indice):
    valore_decimale = ord(stringa[indice])
    valore_non_decimale = ord(stringa[indice+1])
    numero = []
    if valore_decimale != 45 and valore_decimale != 32:
        decimale = boolean_decimal(valore_non_decimale)
        if decimale == False:
            numero.append(48) #il vero valore decimale
            numero.append(valore_decimale) #il valore non_decimale
        else:
            numero.append(valore_decimale)
            numero.append(valore_non_decimale)
    return numero
```

Osservazione: Questo codice in python potrebbe dare problemi in quanto l'istruzione `stringa[indice+1]` non darebbe *none* come risultato, ma un *IndexError: string index out of range*. Mentre in risc, non viene tradotto con zero

- Il metodo `dammi_il_numero` è:

```
397 #prende due valori: a0,a1
398 dammi_il_numero:
399 addi sp,sp,-8
400 add t0,s3,zero #indice
401 sw ra,4(sp)
402 sw t0,0(sp)
403 li t0,48
404 beq a0,t0,noDecimale
405 jal stringa_numero
406 lw t0,0(sp)
407 addi t0,t0,1
408 addi a0,a0,-1 #decrementi il numero principale
409 j fine_numero
410 noDecimale:
411 jal stringa_numero
412 addi a0,a0,-1
413 lw t0,0(sp)
414 fine_numero:
415 lw ra,4(sp)
416 addi sp,sp,8
417 add a1,t0,zero
418 jr ra
```


A sua volta contiene il metodo `stringa_numero`:

```
387 #riceve i valori calcolati da definisci_numero che a sua volta sono passati
388 # a dammi_il_numero
389 stringa_numero:
390 li t0,48
391 sub a0,a0,t0
392 sub a1,a1,t0
393 li t1,10
394 mul t2,a0,t1
395 add a0,t2,a1
396 jr ra
```

In alto livello abbiamo qualcosa del genere:

```
def boolean_decimal(valore):
    boolean = True
    trattino = 45
    spazio = 32
    if valore == trattino or valore == spazio or valore == None:
        boolean = False
    return boolean

def definisci_numero(stringa,indice):
    valore_decimale = ord(stringa[indice])
    valore_non_decimale = ord(stringa[indice+1])
    numero = []
    if valore_decimale != 45 and valore_decimale != 32:
        decimale = boolean_decimal(valore_non_decimale)
        if decimale == False:
            numero.append(48) #il vero valore decimale
            numero.append(valore_decimale) #il valore non_decimale
        else:
            numero.append(valore_decimale)
            numero.append(valore_non_decimale)
    return numero
```

Una volta fatto tutto ciò, inizia la parte finale. Quello che è stato fatto alla cifratura di C, viene fatta anche alla decifratura e cioè la conversione tra `stringaV` e `Myplaintext`.

Prima di inserire i nuovi valori per il nuovo `myplaintext`, quest'ultimo viene cancellato.

Codice finale:

```
421 end_decifraturaC:
422 la a0,myplaintext
423 jal lunghezzaArray
424 add s3,a0,zero
425 la a0,myplaintext
426 jal cancella
427 #Adesso che myplaintext e' vuoto inserisco il nuovo myplaintext
428 la a0,stringaV
429 la a1,myplaintext
430 jal conversione
431 la a0,stringaV
432 jal lunghezzaArray
433 add s3,a0,zero
434 la a0,stringaV
435 jal cancella
436 la a1,stringaMessaggio3
437 li a0,4
438 ecall
439 li t0,13
440 add a1,t0,zero
441 li a0,2
442 ecall
443 la a1,myplaintext
444 li a0,4
445 ecall
446 li a1,13
447 li a0,2
448 ecall
449 addi s0,s0,1
450 j loop
```

DIZIONARIO

[Testo preso dalla spiegazione del progetto]

Ogni possibile simbolo ASCII viene mappato con un altro simbolo ASCII secondo una certa funzione, che riportiamo di seguito definita per casi.

- Se il carattere ci è una lettera minuscola (min), viene sostituito con l'equivalente maiuscolo dell'alfabeto in ordine inverso es. $Z = ct(a)$, $A = ct(z)$.
- Se il carattere ci è una lettera maiuscola (mai), viene sostituito con l'equivalente minuscolo dell'alfabeto in ordine inverso es. $z = ct(A)$, $y = ct(B)$, $a = ct(Z)$.
- Se il carattere ci è un numero (num), $ct(ci) = ASCII(cod(9)-num)$
- In tutti gli altri casi (sym), ci rimane invariato, ovvero $ct(ci) = ci$

Inizio:

```
455 algoritmo_D:
456
457 la a0,myplaintext
458 li s3,123 #z+1
459 li s4,97 #a
460 li s5,91 #Z+1
461 li s6,65 #a
462 li s7,58 #9+1
463 li s8,48 #0
```

Il codice seguente è molto simile all'algoritmo A, almeno per quanto riguarda la suddivisione di maiuscole, minuscole e numeri:

```
465 li t0,0
466 #inizio loop
467 loopD:
468 add t1,t0,a0 #indirizzo a0
469 lb a0,0(t1)
470 beq a0,zero,end_loopD
471 blt a0,s7,numero
472 blt a0,s5,maiuscolo
473 blt a0,s3,minuscolo
474 j aggiornaD
475 numero:
476 blt a0,s8,aggiornaD
477 jal numeroConversione
478 sb a0,0(t1)
479 j aggiornaD
480 maiuscolo:
481 blt a0,s6,aggiornaD
482 li a1,1
483 jal definisciDistanza
484 jal convertireCarattere
485 sb a0,0(t1)
486 j aggiornaD
487 minuscolo:
488 blt a0,s4,aggiornaD
489 li a1,0
490 jal definisciDistanza
491 jal convertireCarattere
492 sb a0,0(t1)
493 aggiornaD:
494 addi t0,t0,1
495 la a0,myplaintext
496 j loopD
```

Si nota, come una volta preso il carattere, avviene un controllo che definisce se è un “possibile” numero, “possibile” maiuscolo e un “possibile” minuscolo.

Se è un possibile numero allora esegue:

```
475 numero:
476 blt a0,s8,aggiornaD
477 jal numeroConversione
478 sb a0,0(t1)
479 j aggiornaD
```

Se il valore viene confermato come un numero allora esegue “numeroConversione”:

```
631 #a0 = numero
632 numeroConversione:
633 li t2,57
634 sub a0,t2,a0
635 add a0,s8,a0
636 jr ra
```

L'operazione che esegue è $\text{ASCII}(\text{cod}(9) - \text{num})$, dove $\text{cod}(9)$ è rappresentato direttamente da $t2$.

Se è un possibile minuscolo o una maiuscola allora esegue:

```
487 minuscolo:
488 blt a0,s4,aggiornaD
489 li a1,0
490 jal definisciDistanza
491 jal convertireCarattere
492 sb a0,0(t1)

480 maiuscolo:
481 blt a0,s6,aggiornaD
482 li a1,1
483 jal definisciDistanza
484 jal convertireCarattere
485 sb a0,0(t1)
486 j aggiornaD
```

Quello che hanno di diverso sono i parametri, ma entrambi chiamano gli stessi metodi, che sono “*definisciDistanza*” e “*convertireCarattere*”.

La funzione *definisciDistanza* serve appunto per calcolare la distanza che c'è tra la prima lettera dell'alfabeto e il carattere che appartiene al myplaintext.
L'idea infatti consiste nel calcolare la distanza e dopo convertirla con “*convertireCarattere*”.

Graficamente *definisciDistanza* possiamo immaginarla in questa maniera.

Esempio: Dato $a0 = D$ e $a1 = 1$ (valore booleano = maiuscola)

Partendo dalle Maiuscole si calcola la distanza:

65	A	01100001	97	a
66	B	01100010	98	b
67	C	01100011	99	c
68	D	01100100	100	d
69	E	01100101	101	e
70	F	01100110	102	f
71	G	01100111	103	g
72	H	01101000	104	h
73	I	01101001	105	i

Il codice risc-v e python:

```
638 definisciDistanza:
639 li t2,1
640 bne a1,t2,minuscolo2
641 addi a0,a0,-65
642 j salta_fuction
643 minuscolo2:
644 addi a0,a0,-97
645 salta_fuction:
646 jr ra

def definisciNumero(carattere,bool_maiuscolo):
    if bool_maiuscolo == 1:
        temp = carattere - 65
    else:
        temp = carattere - 97
    return temp
```

Ora manca soltanto da definire “*convertireCarattere*”. Una volta che ho calcolato distanza, devo definire quale sarà il mio nuovo carattere.
Continuando con l’esempio, il risultato sarà il seguente

85	U	01110101	117	u	
86	V	01110110	118	v	
87	W	01110111	119	w	3
88	X	01111000	120	x	2
89	Y	01111001	121	y	1
90	Z	01111010	122	z	

Codice risc-v:

```
649 #a0 = distanza, a1 = bool
650 convertireCarattere:
651 li t2,1 # serve per l'uguaglianza
652 li t3,122 # z
653 li t4,90 # Z
654 bne a1,t2,minuscolo3
655 sub a0,t3,a0
656 j salta2
657 minuscolo3:
658 sub a0,t4,a0
659 salta2:
660 jr ra
```

In python:

```
def convertireCarattere(distanza,maiuscolo):  
    if maiuscolo == 1:  
        temp = 122 - distanza  
    else:  
        temp = 90- distanza  
    return temp
```

L'algoritmo D, per quanto riguarda la decifratura, ripete lo stesso codice.

FASE DECIFRATURA

La fase di decifratura oltre ad essere molto piccola è anche molto semplice.

Il main per la decifratura è:

```
115 decifratura:  
116 #Per la decifratura  
117 la a1,stringa_Messaggio6  
118 li a0,4  
119 ecall  
120 li a1,13  
121 li a0,2  
122 ecall  
123 la a0,mychiper  
124 sub s1,zero,s1 #-k  
125 li s0,0 #l'indice viene ripristinato  
126 li s2,1 #decifratura true  
127 j algoritmo_E
```

Vediamo come inizia a cambiare i 3 registri che non sono stati toccati durante il codice s0,s1,s2 e soprattutto a0 = mychiper, perché sarà quello che verrà modificato dall'algoritmo E.

Il registro s0 che rappresenta l'indice usato per caricare le lettere del mychiper viene inizializzato.

Il registro s1 = -k per la decifratura dell'algoritmo A.

Infine, abbiamo il registro s2 = 1, che permette di eseguire la formula di decifratura della B e C oltre ad uscire dal main principale.

Nella fase di decifratura viene utilizzato solo un algoritmo che è l'INVERSIONE.

INVERSIONE

```
518 algoritmo_E:
519
520 jal lunghezzaArray
521 addi s3,a0,-1 # lunghezza array-1
522 la a0,mychip
523 li a1,0
524 add a2,zero,s3
525
526 loop_inversion:
527 bge a1,a2,end_inversion
528 jal swap
529 addi a1,a1,1
530 addi a2,a2,-1
531 j loop_inversion
532
533
534 end_inversion:
535 j loop
```

Il codice consiste nel prendere i due valori estremi nella stringa, ed eseguire un swap, dove il codice swap viene eseguito in questa maniera:

```
662 #Algoritmi per la E
663 # a0 = indirizzo, a1= primo elemento, a2 = ultimo valore
664 swap:
665 add t0,a0,a2
666 lb t1,0(t0) # carico in t1 l'ultimo carattere della stringa
667 add t2,a0,a1
668 lb t3,0(t2)
669 sb t1,0(t2)
670 sb t3,0(t0)
671 jr ra
```

Esempi

In questa ultima parte della relazione, sono presenti una buona dose di esempi che chiariscono i vari risultati del progetto.

1)

```
15 mychip: .string "A"
16 k: .word 1
17 key: .string ""
18 myplaintext: .string "AMO AssEMbLY!"
```

La tua stringa e':
AMO AssEMbLY!
ALG_A:
BNP BttFNcMZ!
ESECUZIONE DECIFRATURA
ALG_A:
AMO AssEMbLY!

2)

```
15 mychipер: .string "A"  
16 k: .word -1  
17 key: .string ""  
18 myplaintext: .string "AAAaaaAAA"
```

La tua stringa e':
AAAaaaAAA
ALG_A:
ZZZzzzZZZ
ESECUZIONE DECIFRATURA
ALG_A:
AAAaaaAAA

3)

```
15 mychipер: .string "A"  
16 k: .word 10  
17 key: .string "OLE"  
18 myplaintext: .string "Ciao, mi chiamo Emanuele matricola: 6147014"
```

La tua stringa e':
Ciao, mi chiamo Emanuele matricola: 6147014
ALG_A:
Msky, ws mrskwy Owkxeovo wkdbsmvkv: 6147014
ESECUZIONE DECIFRATURA
ALG_A:
Ciao, mi chiamo Emanuele matricola: 6147014

4)

```
15 mychipер: .string "B"  
16 k: .word 10  
17 key: .string "OLE"  
18 myplaintext: .string "LAUREATO_1"
```

La tua stringa e':
LAUREATO_1
ALG_B:
{mz!qf#{`
ESECUZIONE DECIFRATURA
ALG_B:
LAUREATO_1

5)

```
15 mychipер: .string "B"  
16 k: .word 10  
17 key: .string "OLa"  
18 myplaintext: .string "LAUREATO_1"
```

La tua stringa e':

LAUREATO_1

ALG_B:

{m6!q"#{ @`

ESECUZIONE DECIFRATURA

ALG_B:

LAUREATO_1

6)

```
15 mychipер: .string "B"  
16 k: .word 10  
17 key: .string "IAPPO ALOH"  
18 myplaintext: .string "LAUREATO_1"
```

La tua stringa e':

LAUREATO_1

ALG_B:

ub%"zAu{.Y

ESECUZIONE DECIFRATURA

ALG_B:

LAUREATO_1

7)

```
15 mychipер: .string "AAB"  
16 k: .word 50  
17 key: .string "OLE"  
18 myplaintext: .string "Ciao, mi chiamo Emanuele matricola: 6147014"
```

La tua stringa e':

Ciao, mi chiamo Emanuele matricola: 6147014

ALG_A:

Agym, kg afgykm Ckylscjc kyrpgamjy: 6147014

ALG_A:

Yewk, ie ydewik Aiwjqaha iwpneykhw: 6147014

ALG_B:

(1<:XE81EH0*F50Om.F6604&O5<?:*H7-FfEe]Yf\Vc

ESECUZIONE DECIFRATURA

ALG_B:

Yewk, ie ydewik Aiwjqaha iwpneykhw: 6147014

ALG_A:

Agym, kg afgykm Ckylscjc kyrpgamjy: 6147014

ALG_A:

Ciao, mi chiamo Emanuele matricola: 6147014

8)

```
15 mycipher: .string "C"  
16 k: .word 10  
17 key: .string ""  
18 myplaintext: .string "sempio di messaggio criptato -1"
```

La tua stringa e':

sempio di messaggio criptato -1

ALG_C:

s-1-13-14 e-2-12 m-3-11 p-4-24 i-5-9-18-23 o-6-19-28 -7-10-20-29 d-8 a-15-26 g-16-17 c-21 r-22 t-25-27 --
30 1-31

ESECUZIONE DECIFRATURA

ALG_C:

sempio di messaggio criptato -1

9)

```
15 mycipher: .string "CCCAC"  
16 k: .word 10  
17 key: .string ""  
18 myplaintext: .string "c"
```

La tua stringa e':

c

ALG_C:

c-1

ALG_C:

c-1 --2 1-3

ALG_C:

c-1 --2-5-6-10 1-3-9 -4-8 2-7 3-11

ALG_A:

m-1 --2-5-6-10 1-3-9 -4-8 2-7 3-11

ALG_C:

m-1 --2-5-6-8-10-12-17-19-23-25-29-33 1-3-13-16-34-35 -4-15-21-22-27-31 2-7-28 5-9 6-11 0-14 3-18-32
9-20 4-24 8-26 7-30

ESECUZIONE DECIFRATURA

ALG_C:

m-1 --2-5-6-10 1-3-9 -4-8 2-7 3-11

ALG_A:

c-1 --2-5-6-10 1-3-9 -4-8 2-7 3-11

ALG_C:

c-1 --2 1-3

ALG_C:

c-1

ALG_C:

c

10)

I caratteri nel myplaintext sono 99

```

15 mychipер: .string "C"
16 k: .word 160
17 key: .string "0"
18 myplaintext: .string "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

```

La tua stringa e':

aa

a

ALG_C:

a-1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-

36-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-54-55-56-57-58-59-60-61-62-63-64-65-66-67-

68-69-70-71-72-73-74-75-76-77-78-79-80-81-82-83-84-85-86-87-88-89-90-91-92-93-94-95-96-97-98-99

ESECUZIONE DECIFRATURA

ALG_C:

aa

a

11)

```

15 mychipер: .string "D"
16 k: .word 10
17 key: .string ""
18 myplaintext: .string "myStr0ng P4ssW_"

```

La tua stringa e':

myStr0ng P4ssW_

ALG_D:

NBhGI9MT k5HHd_

ESECUZIONE DECIFRATURA

ALG_D:

myStr0ng P4ssW_

12)

```

15 mychipер: .string "DD"
16 k: .word 10
17 key: .string ""
18 myplaintext: .string "myStr0ng P4ssW_"

```

La tua stringa e':

myStr0ng P4ssW_

ALG_D:

NBhGI9MT k5HHd_

ALG_D:

myStr0ng P4ssW_

ESECUZIONE DECIFRATURA

ALG_D:

NBhGI9MT k5HHd_

ALG_D:

myStr0ng P4ssW_

13)

```
15 mychipер: .string "ABCDА"
16 k: .word 160
17 key: .string "PiPPo"
18 myplaintext: .string "Progetto_completato19/20/2020"
```

La tua stringa e':

Progetto_completato19/20/2020

ALG_A:

Tvskixxs_gsqtPixexs19/20/2020

ALG_B:

\$_C;XHаC/VCZD@XHNHC ixb`~byb`

ALG_C:

\$-1_-2 C-3-8-11-19 ;-4 X-5-15 H-6-16-18 a-7 /-9 V-10 Z-12 D-13 @-14 N-17 -20 i-21 x-22 b-23-26-28 ` -
24-29 ~-25 y-27

ALG_D:

\$-8_-7 x-6-1-88-80 ;-5 c-4-84 s-3-83-81 Z-2 /-0 e-89 a-87 w-86 @-85 m-82 -79 R-78 C-77 Y-76-73-71 ` -
75-70 ~-74 B-72

ALG_A:

\$-8_-7 b-6-1-88-80 ;-5 g-4-84 w-3-83-81 D-2 /-0 i-89 e-87 a-86 @-85 q-82 -79 V-78 G-77 C-76-73-71 ` -75-
70 ~-74 F-72

ESECUZIONE DECIFRATURA

ALG_A:

\$-8_-7 x-6-1-88-80 ;-5 c-4-84 s-3-83-81 Z-2 /-0 e-89 a-87 w-86 @-85 m-82 -79 R-78 C-77 Y-76-73-71 ` -
75-70 ~-74 B-72

ALG_D:

\$-1_-2 C-3-8-11-19 ;-4 X-5-15 H-6-16-18 a-7 /-9 V-10 Z-12 D-13 @-14 N-17 -20 i-21 x-22 b-23-26-28 ` -
24-29 ~-25 y-27

ALG_C:

\$_C;XHаC/VCZD@XHNHC ixb`~byb`

ALG_B:

Tvskixxs_gsqtPixexs19/20/2020

ALG_A:

Progetto_completato19/20/2020

Codice

#Progetto Ade

#Autore-Barbieri Emanuele Luca

#Matricola-6147014

.data

stringa_errore1: .string "Errore Mychipер: ALG #-che?"

stringa_errore2: .string "Errore Mychipер: cifratura non eseguibile per troppe C"

stringa_errore3: .string "Errore Myplaintext: troppo lungo da cifrare"

stringa_errore4: .string "Errore Mychipер: sequenza non consentita, troppo lunga"

stringaMessaggio1: .string "ALG_A:"

stringaMessaggio2: .string "ALG_B:"

stringaMessaggio3: .string "ALG_C:"

stringaMessaggio4: .string "ALG_D:"

stringaMessaggio5: .string "La tua stringa e': "

stringa_Messaggio6: .string "ESECUZIONE DECIFRATURA"

```
mychipper: .string "ABCD A"  
k: .word 10  
key: .string "OLE"  
myplaintext: .string "Ciao, mi chiamo Emanuele matricola: 6147014"  
space: .string " "  
stringaV: .string ""  
.text  
#FASE DI BLOCCO ERRORI
```

```
#controllo mychipper massimo 5 caratteri
```

```
la a0,mychipper  
jal lunghezzaArray  
la a1,stringa_errore4  
li t0,6  
bge a0,t0,errore_controllo
```

```
#controllo mychipper: disponibilit? algoritmi
```

```
la a0,mychipper  
jal controllo1  
la a1,stringa_errore1  
beq a0,zero,errore_controllo
```

```
#controllare myplaintext lunghezza < 100
```

```
la a0,myplaintext  
jal lunghezzaArray  
la a1,stringa_errore3  
li t0,100  
bge a0,t0,errore_controllo
```

```
#controllo per eseguire un tot di C
```

```
la a0,mychipper  
#controllo_conteggioC(a0 = stringa)  
jal controllo_conteggioC  
beq a0,zero,finito_controllo  
add s0,a0,zero #salvo conteggio per confronto  
la a0,myplaintext  
#lunghezzaArray(a0 = stringa)  
jal lunghezzaArray  
#controllo2(a0 = int_lunghezzaArray)  
jal controllo2  
bge a0,s0,finito_controllo  
la a1,stringa_errore2  
j errore_controllo
```

```
finito_controllo:  
#FASE DI RICAPO SPAZIO  
la a0,space
```

```
jal lunghezzaArray
add s3,a0,zero # salvo la lunghezza,usata in cancella
la a0,space
jal cancella
```

#INIZIO CIFRATURA & DECIFRATURA

```
#Stampa Intro
```

```
la a1,stringaMessaggio5
```

```
li a0,4
```

```
ecall
```

```
li a1,13
```

```
li a0,2
```

```
ecall
```

```
la a1,myplaintext
```

```
li a0,4
```

```
ecall
```

```
li a1,13
```

```
li a0,2
```

```
ecall
```

```
main:
```

```
li s0,0 #indice di mychipper
```

```
lw s1,k
```

```
li s2,0 #segnale per cifratura e decifratura
```

```
loop:
```

```
la a0,mychipper
```

```
li t0,65 #A
```

```
li t1,66
```

```
li t2,67
```

```
li t3,68
```

```
add t5,s0,a0
```

```
lb t6,0(t5)
```

```
#fase di controllo
```

```
beq s2,zero,decifratura_false
```

```
beq t6,zero,end_main
```

```
decifratura_false:
```

```
beq t6,zero,decifratura
```

```
beq t6,t0,algoritmo_A
```

```
beq t6,t1,algoritmo_B
```

```
bne t6,t2,continua
```

```
beq s2,zero,cifraturaC
```

```
j algoritmo_C_decifratura
```

```
cifraturaC:
```

```
j algoritmo_C
```

```
continua:
```

```
beq t6,t3,algoritmo_D
```

```
decifratura:
```

```
#Per la decifratura
```

```

la a1,stringa_Messaggio6
li a0,4
ecall
li a1,13
li a0,2
ecall
la a0,mychiper
sub s1,zero,s1 #-k
li s0,0 #l'indice viene ripristinato
li s2,1 #decifratura true
j algoritmo_E
#-----ALGORITMO A-----#
algoritmo_A:

la s3,myplaintext
li s4,97
li s5,123
li s6,65
li s7,91

li t0,0

loop_A:
add t1,t0,s3
lb a1,0(t1)
beq a1,zero,end_loop_A
blt a1,s7,maiusc
blt a1,s5,minusc
j aggiorna

maiusc:
blt a1,s6,aggiorna
add a0,zero,s1 #lw a0,k sostituisce
add a2,s6,zero
add a3,s7,zero
jal miniciclo
sb a0,0(t1)
j aggiorna

minusc:
blt a1,s4,aggiorna
add a0,zero,s1 #lw a0,k
add a2,s4,zero
add a3,s5,zero
jal miniciclo
sb a0,0(t1)
j aggiorna

aggiorna:
addi t0,t0,1
j loop_A

```

```

end_loop_A:
la a1,stringaMessaggio1
li a0,4
ecall
li t0,13
add a1,t0,zero
li a0,2
ecall
add a1,s3,zero
li a0,4
ecall
add a1,t0,zero
li a0,2
ecall
addi s0,s0,1 #incremento indice
j loop

```

```

#-----ALGORITMO B-----#
algoritmo_B:

```

```

#inizio algoritmo
li s3,32
li s4,128
la s5,myplaintext
la a0,key
#a0 = indirizzo stringa
jal lunghezzaArray

```

```

add s6,a0,zero #salvo in s6 la lunghezza
li s7,0 #indice per key

```

```

#Fase Algoritmo
li t0,0 #per il loop

```

```

loop_algoritmo_blocchi:
add t1,t0,s5
lb t2,0(t1)
beq t2,zero,end_loopB
blt t2,s3,salta
bge t2,s4,salta
la a0,key
add a1,s7,zero #carico l'indice
jal ciclo_key
# a0 valore key[indice]
add s7,a1,zero
add a1,t2,zero #carico valore
#fase decisionale tra cifratura e decifratura

```



```

beq s2,zero,esegui_formula
jal formula_decifratura
j avanti
esegui_formula:
jal formula
avanti:
sb a0,0(t1)
addi s7,s7,1
salta:
addi t0,t0,1
j loop_algoritmo_blocchi

```

```

end_loopB:
la a1,stringaMessaggio2
li a0,4
ecall
li t0,13
add a1,t0,zero
li a0,2
ecall
add a1,s5,zero
li a0,4
ecall
li t0,13
add a1,t0,zero
li a0,2
ecall
addi s0,s0,1
j loop

```

#-----ALGORITMO C -----#

#CIFRATURA

```

algoritmo_C:
la a0,stringaV #indirizzo SV
la a1,myplaintext #indirizzo stringa
li s3,45 # -
li s4,32 # spazio
li s5,0 #contatore primo loop
li s6,0 #contatore stringa_vuota ( nel main)
li s7,31 #valore usato per marcare

```

```

loop1:
add t0,a1,s5 #indirizzo
lb a1,0(t0) #lettera_myplaintext
lb s8,1(t0) #lettera_myplaintext successiva
beq a1,zero,end_loop
beq a1,s7,aggiornamento
add a2,s6,zero #inserisco il contatore
jal inserisci_carattere
addi s6,s6,1 #aumento il contatore

```

```

add a2,a1,zero
la a1,myplaintext
#a0 stringaV #a1 stringa #a2 carattere da cercare
jal inserisci_posizione
add a2,zero,a0
#inserimento spazio
beq s8,zero,aggiornamento
la a0,stringaV
add a1,zero,s4
jal inserisci_carattere
addi s6,a2,1
aggiornamento:
addi s5,s5,1
la a0,stringaV
la a1,myplaintext
j loop1

```

```

end_loop:
#qui inizia il passaggio da stringaV a myplaintext
la a0,stringaV
la a1,myplaintext
jal conversione
la a0,stringaV
jal lunghezzaArray
add s3,a0,zero
la a0,stringaV
jal cancella
la a1,stringaMessaggio3
li a0,4
ecall
li t0,13
add a1,t0,zero
li a0,2
ecall
la a1,myplaintext
li a0,4
ecall
li a1,13
li a0,2
ecall
addi s0,s0,1
j loop

```

```

#DECIFRATURA C
algoritmo_C_decifratura:

```

```

la a0,myplaintext
li s3,0 #indice per l'intero loop
li s4,32
li s5,45

```

```

jal lunghezzaArray
add s6,a0,zero

loop_esterno:
la a0,myplaintext
add t0,s3,a0 #indirizzo
lb s7,0(t0)
beq s7,zero,end_decifraturaC
addi s3,s3,1
loop_interno:
jal definisci_numero
addi s3,s3,1
beq a0,s4,loop_esterno
beq a0,s5,update
#INIZIA A DEFINIRE IL NUMERO
jal dammi_il_numero
add s3,a1,zero
la t0,stringaV
add t1,a0,t0 #stringaV[valore]
sb s7,0(t1)
update:
beq s3,s6,loop_esterno #istruzione da aggiustare con len(stringa)
la a0,myplaintext
j loop_interno

#definisci_numero(a0 = indirizzo)
definisci_numero:
addi sp,sp,-8
sw ra,4(sp)
add t0,s3,zero #metto l'indice in una variabile temporanea
add t1,t0,a0 #indirizzo
lb a1,1(t1) #temp
lb a0,0(t1) #valore
beq a0,s5,passa_oltre
beq a0,s4,passa_oltre
sb a0,0(sp)
add a0,a1,zero
jal boolean_decimal
beq a0,zero,no_decimal
lb a0,0(sp)#ripristina valore di a0
j passa_oltre
no_decimal:
lb a1,0(sp) #ripristino a0
li a0,48 #dato che a0 e' il valore decimale, ho posto a0 = 48 cioe' zero
passa_oltre:
lw ra,4(sp)
addi sp,sp,8
jr ra

#a0 = valore
boolean_decimal:

```

```

li t0,1
li t1,45
li t2,32
beq a0,t1,cambia
beq a0,t2,cambia
beq a0,zero,cambia
j passa_avanti
cambia:
li t0,0
passa_avanti:
add a0,t0,zero
jr ra

```

```

#riceve i valori calcolati da definisci_numero che a sua volta sono passati
# a dammi_il_numero
stringa_numero:
li t0,48
sub a0,a0,t0
sub a1,a1,t0
li t1,10
mul t2,a0,t1
add a0,t2,a1
jr ra

```

```

#prende due valori: a0,a1
dammi_il_numero:
addi sp,sp,-8
add t0,s3,zero #indice
sw ra,4(sp)
sw t0,0(sp)
li t0,48
beq a0,t0,noDecimale
jal stringa_numero
lw t0,0(sp)
addi t0,t0,1
addi a0,a0,-1 #decrementi il numero principale
j fine_numero
noDecimale:
jal stringa_numero
addi a0,a0,-1
lw t0,0(sp)
fine_numero:
lw ra,4(sp)
addi sp,sp,8
add a1,t0,zero
jr ra

```

```

end_decifraturaC:
la a0,myplaintext
jal lunghezzaArray
add s3,a0,zero

```

```

la a0,myplaintext
jal cancella
#Adesso che myplaintext e' vuoto inserisco il nuovo myplaintext
la a0,stringaV
la a1,myplaintext
jal conversione
la a0,stringaV
jal lunghezzaArray
add s3,a0,zero
la a0,stringaV
jal cancella
la a1,stringaMessaggio3
li a0,4
ecall
li t0,13
add a1,t0,zero
li a0,2
ecall
la a1,myplaintext
li a0,4
ecall
li a1,13
li a0,2
ecall
addi s0,s0,1
j loop

```

#-----ALGORITMO D-----#

algoritmo_D:

```

la a0,myplaintext
li s3,123 #z+1
li s4,97 #a
li s5,91 #Z+1
li s6,65 #a
li s7,58 #9+1
li s8,48 #0

li t0,0
#inizio loop
loopD:
add t1,t0,a0 #indirizzo a0
lb a0,0(t1)
beq a0,zero,end_loopD
blt a0,s7,numero
blt a0,s5,maiuscolo
blt a0,s3,minuscolo
j aggiornaD
numero:

```

```

blt a0,s8,aggiornaD
jal numeroConversione
sb a0,0(t1)
j aggiornaD
maiuscolo:
blt a0,s6,aggiornaD
li a1,1 #variabile booleana
jal definisciDistanza
jal convertireCarattere
sb a0,0(t1)
j aggiornaD
minuscolo:
blt a0,s4,aggiornaD
li a1,0 # variabile booleana
jal definisciDistanza
jal convertireCarattere
sb a0,0(t1)
aggiornaD:
addi t0,t0,1
la a0,myplaintext
j loopD

```

```

end_loopD:
la a1,stringaMessaggio4
li a0,4
ecall
li t0,13
add a1,t0,zero
li a0,2
ecall
la a1,myplaintext
li a0,4
ecall
li t0,13
add a1,t0,zero
li a0,2
ecall
addi s0,s0,1
j loop

```

#-----ALGORITMO E-----#

algoritmo_E:

```

jal lunghezzaArray
addi s3,a0,-1 # lunghezza array-1
la a0,mychipper
li a1,0
add a2,zero,s3

```

loop_inversion:

```

bge a1,a2,end_inversion
jal swap
addi a1,a1,1
addi a2,a2,-1
j loop_inversion

```

```

end_inversion:
j loop

```

```

#-----METODI UTILIZZATI-----#

```

```

#metodo per la lunghezza di una stringa

```

```

#lunghezzaArray(a0 = stringa)

```

```

lunghezzaArray:

```

```

li t0,0 #indice=0

```

```

loop_lunghezzaArray:

```

```

add t1,a0,t0

```

```

lb t2,0(t1)

```

```

beq t2,zero,end_loop_LunghezzaArray

```

```

addi t0,t0,1

```

```

j loop_lunghezzaArray

```

```

end_loop_LunghezzaArray:

```

```

add a0,t0,zero

```

```

jr ra

```

```

#metodo per algortimo A

```

```

#a0 = k, a1 = valore_carattere, a2= estremo_inf, a3 = estremo_sup

```

```

miniciclo:

```

```

add t2,a1,a0

```

```

loop_miniciclo_inf:

```

```

bge t2,a2,loop_miniciclo_sup

```

```

sub t2,a2,t2

```

```

sub t2,a3,t2

```

```

j loop_miniciclo_inf

```

```

loop_miniciclo_sup:

```

```

blt t2,a3,fine

```

```

sub t2,t2,a3

```

```

add t2,a2,t2

```

```

j loop_miniciclo_sup

```

```

fine:

```

```

add a0,t2,zero

```

```

jr ra

```

```

#metodi per algortimi B

```

```

ciclo_key:
#passati come parametri a0 indirizzo key, a1 indice
bne a1,s6,salta_istruzione
li a1,0
salta_istruzione:
add t3,a0,a1
lb a0,0(t3)
jr ra

```

```

#a0 = valore_key, a1 = valore_lettera

```

```

formula:

```

```

addi sp,sp,-4

```

```

sw ra,0(sp)

```

```

sub t2,a0,s3

```

```

sub t3,a1,s3

```

```

add a0,t2,t3

```

```

jal modulo

```

```

add a0,a0,s3

```

```

lw ra,0(sp)

```

```

addi sp,sp,4

```

```

jr ra

```

```

#stessi parametri di formula

```

```

formula_decifratura:

```

```

addi sp,sp,-4

```

```

sw ra,0(sp)

```

```

add t2,a0,zero #conservo Valore Key

```

```

sub a0,a1,s3 #valore_stringa-32

```

```

jal modulo

```

```

add t5,zero,a0

```

```

sub a0,t2,s3

```

```

jal modulo

```

```

add t3,zero,a0

```

```

sub t4,t5,t3

```

```

bge t4,zero,positivo

```

```

addi t4,t4,128

```

```

j finish

```

```

positivo:

```

```

add t4,t4,s3

```

```

finish:

```

```

lw ra,0(sp)

```

```

addi sp,sp,4

```

```

add a0,t4,zero

```

```

jr ra

```

```

modulo:

```

```

li t3,96

```

```

div t4,a0,t3

```

```

mul t4,t4,t3

```

```

sub a0,a0,t4

```

```

jr ra

```



```

#algoritmi per la D
#a0 = numero
numeroConversione:
li t2,57
sub a0,t2,a0
add a0,s8,a0
jr ra

```

```

#a0=valore lettera, a1=boolean
definisciDistanza:
li t2,1
bne a1,t2,minuscolo2
addi a0,a0,-65
j salta_fuction
minuscolo2:
addi a0,a0,-97
salta_fuction:
jr ra

```

```

#a0 = distanza, a1 = bool
convertireCarattere:
li t2,1 # serve per l'uguaglianza
li t3,122 # z
li t4,90 # Z
bne a1,t2,minuscolo3
sub a0,t3,a0
j salta2
minuscolo3:
sub a0,t4,a0
salta2:
jr ra

```

```

#Algoritmi per la E
# a0 = indirizzo, a1= primo elemento, a2 = ultimo valore
swap:
add t0,a0,a2
lb t1,0(t0) # carico in t1 l'ultimo carattere della stringa
add t2,a0,a1
lb t3,0(t2)
sb t1,0(t2)
sb t3,0(t0)
jr ra

```

```

#Algoritmi C cifratura
#a0 = l'indirizzo, a2 = indice, a1 = carattere
inserisci_carattere:
add t3,a0,a2 #aggiornamento stringaV
sb a1,0(t3)
jr ra

```

```

inserisci_posizione:
addi sp,sp,-20 # 5 posizioni
sw ra,16(sp)
li t0,0 #contatore stringa
add t1,s6,zero #contatore stringaV
loop_posizione:
add t2,t0,a1 #stringa[i]
lb t3,0(t2)
beq t3,zero,end_loop2
beq t3,s7,aggiornamento_loop
bne t3,a2,aggiornamento_loop
sb s7,0(t2) #marcatura
add a1,s3,zero #trattino
sb a2,12(sp) #carico il carattere
add a2,t1,zero #contatore
#a0 stringaV #a1 trattino #a2 contatore
jal inserisci_carattere
addi t1,t1,1
sw t0,8(sp)
sw t1,4(sp)
sw a0,0(sp)
add a0,t0,zero
#a0 indice da trasformare in numero
addi a0,a0,1 #perche' vuole da 1
jal numero_stringa
add t0,a0,zero
add t1,a1,zero
lw a0,0(sp) #indirizzo stringa vuota
lw a2,4(sp) #contatore stringa vuota
li t2,48
beq t0,t2,no_decimale
add a1,t0,zero #valore decimale
#a0 stringaV #a1 valore intero decimale #a2 contatore stringa vuota
jal inserisci_carattere
addi a2,a2,1 #contatore stringa vuota
no_decimale:
#inserimento forse di a2 non serve
add a1,t1,zero
#a0 stringaV #a1 valore numerico #a2 contatore
jal inserisci_carattere
addi t1,a2,1
lw t0,8(sp)
lb a2,12(sp) #carattere
la a1,myplaintext
aggiornamento_loop:
addi t0,t0,1
j loop_posizione

end_loop2:
lw ra,16(sp)
addi sp,sp,20

```

```
add a0,t1,zero
jr ra
```

```
#a0 = posizione
numero_stringa:
li t0,48
li t1,48 #valore decimale
li t2,0 #count
li t3,9
loop_while:
bge t2,a0,end_while
bne t2,t3,else
li t2,0
addi t1,t1,1
li t0,48
addi a0,a0,-10
j loop_while
else:
addi t0,t0,1
addi t2,t2,1
j loop_while
end_while:
add a0,t1,zero #valore decimale
add a1,t0,zero
jr ra
```

```
#algoritmo usato per entrambi
#cancella(a0 = space,s3 = lunghezza salvata)
```

```
cancella:
li t0,0 #indice
add t1,zero,zero #valore che serve per cancellare
cancella_loop:
bge t0,s3,cancella_end_loop
add t2,t0,a0
sb t1,0(t2)
addi t0,t0,1
j cancella_loop
cancella_end_loop:
jr ra
```

```
conversione:
li t0,0
ciclo_conversione:
add t1,t0,a0 #indirizzo stringaV
add t2,t0,a1 #indirizzo myplaintext
lb t3,0(t1) #valore stringaV
beq t3,zero,end_ciclo_conversione
sb t3,0(t2)
addi t0,t0,1
```

```
j ciclo_conversione
end_ciclo_conversione:
jr ra
```

```
#-----METODI DI CONTROLLO-----#
#controllo1(a0 = stringa)
```

```
controllo1:
li t0,0 # contatore
li t1,65 # A
li t2,69 # E
li t3,1 #valore booleano TRUE
```

```
loop_controllo1:
add t4,t0,a0
lb t5,0(t4)
beq t5,zero,end_controllo1
blt t5,t1,trovato_errore1
bge t5,t2,trovato_errore1
addi t0,t0,1
j loop_controllo1
trovato_errore1:
li t3,0 #FALSE
end_controllo1:
add a0,t3,zero
jr ra
```

```
#controllo2 (a0 = int lunghezzaArray)
controllo2:
li t0,1 #contatore = 1, perch? almeno 1 pu? farlo
li t1,100 # i 100 caratteri che non devono essere superati
add t2,a0,zero
again:
slli t3,t2,2 #t2*4
addi t4,t2,-1 #(t2-1)
add t2,t3,t4
bge t2,t1,end_loop_controllo2
addi t0,t0,1
j again
end_loop_controllo2:
add a0,t0,zero
jr ra
```

```
#conteggio C = conta quanti C sono presenti nella stringa
#controllo_conteggioC (a0 = stringa)
```

```
controllo_conteggioC:
li t0,0
li t1,67
li t2,0 #count_C
controllo2_zero_loop:
```

```
add t3,a0,t0 #indirizzo
lb t4,0(t3)
beq t4,zero,end_loop_controllo2_zero
bne t4,t1,salta_controllo2_zero
addi t2,t2,1
salta_controllo2_zero:
addi t0,t0,1
j controllo2_zero_loop
end_loop_controllo2_zero:
add a0,t2,zero
jr ra
```

```
errore_controllo:
li a0,4
ecall
```

```
end_main:
```

Note informative

Ho scritto meno commenti per evitare di aumentare le righe e soprattutto ho evitato di spostare i metodi della decifratura di C, per timore che il ripes evidenziava problemi.

Esempi di errore che ha dato il ripes:

- 1) Dopo un tot di righe inizia a cambiare l'indirizzo della stringa, per questo motivo durante i controlli ho inserito gli indirizzi dei messaggi di errore, prima di andare in errore_controllo
- 2) Per un solo giorno mi ha considerato lb come lw
- 3) Nonostante abbia trovato una soluzione per risolvere il problema della Key, il codice non viene per nulla considerato dal ripes

Inoltre, gli errori, per quanto riguarda il multiplo di 4, nel mychiper vengono considerati.

