

Progetto Assembly MIPS A.A. 2018/2019

Relazione

Informazioni e data di consegna

Autore: Matteo Menichetti (matteo.menichetti@stud.unifi.it);

Data di consegna: 29 agosto 2019

Descrizione soluzione adottata

Sezione .data

Questa sezione è dedicata alla memoria statica del programma. Sono dichiarate delle procedure che permettono di allocare un determinato numero di byte, utilizzando la direttiva `.space` (dove verranno inseriti il messaggio e la chiave), e di raggiungere i file utili per l'esecuzione del programma. I file saranno utilizzati per estrarre il messaggio e la chiave e per inserire il messaggio criptato o decriptato. La direttiva `.space 256` analoga al messaggio è dovuta alla richiesta di spazio aggiuntivo quando è richiesta la cifratura con gli algoritmi A, B, C in quanto l'aumento del valore ASCII, senza il dovuto spazio, andrebbe a sovrascrivere la memoria statica dedicata all'utente.

Sezione .text

Nelle procedure a seguire alcuni registri hanno la caratteristica di contenere valori, in modo univoco per tutte le procedure, che variano il comportamento del programma e sono:

- `$s0` contiene l'indirizzo in memoria del messaggio;
- `$s1` contiene l'indirizzo in memoria della chiave;
- `$t4` contiene il numero di caratteri del messaggio (variano tra gli algoritmi A-B-C-D e l'algoritmo E);
- `$t7` contiene un valore (0/1) a seconda se il programma è in fase di crittografia (0) o decrittografia (1);

Procedura **main**: viene memorizzato l'indirizzo di memoria (`$ra`) all'interno della sezione di memoria preposta per l'utilizzo della struttura dati Stack. Vengono invocate due procedure che impostano l'apertura dei file a seconda della fase del programma (cifratura o decifratura).

Procedure **open_File-suff** e **read_-suff**: sono utilizzate per aprire i file "messaggio.txt" e "chiave.txt" inserendo il contenuto di questi in due buffer separati.

Procedura **selectAlg**: questa procedura è utilizzata per scorrere il contenuto del buffer contenente la chiave ed a seconda del valore del registro \$t7 viene scorso da sinistra verso destra o al contrario.

PSEUDOCODICE:

//in fase di cifratura T7 assume valore “false” (0)

S0 ← INIZIO MESSAGGIO

T2 = S0[i]

IF !(A < T2 < E) → Scrivi il risultato

IF (T7) → i-1 //per percorrere a ritroso il cifrario viene in/decrementato

ELSE i+1

IF (A < T2 < B) → ALGORITMOABC

IF (T2 == C) → ALGORITMOC

IF (T2 == D) → ALGORITMOD

IF (T2 == E) → ALGORITMOE

SALTO A PROC. selectAlg

Procedura **ALGORITMOABC**: procedura che determina, se in fase di cifratura, debba essere effettuata una somma, caricando l'immediato 4 nel registro \$t0 o invocare la procedura **algDecod**.

La procedura **ALGORITMOABC** utilizza procedure secondarie per effettuare le operazioni di somma e di scrittura dei caratteri cifrati commentate all'interno del sorgente. Una caratteristica degli algoritmi A, B, C e D è che non richiedono memoria ausiliaria per la cifratura o decifratura del messaggio.

PSEUDOCODICE:

WHILE (i < S0.lunghezza){

IF (S0[i] + 4 > 255) → S0[i] = (S0[i] + 4) / 256

ELSE S0[i] = S0[i] + 4

IF (S1[i] == A) → i + 1

IF (B < S1[i] < C) → i + 2

}

SALTO A PROC. selectAlg

Procedura **ALGORITMOD**: procedura che inverte i caratteri del messaggio utilizzando il numero di caratteri da cui è composto (\$t4). Sommando \$t4 all'indirizzo associato a \$s0 (somma nel registro \$t2) e copiando l'indirizzo del messaggio (\$s0) in un secondo registro (\$t3) si evita la modifica del registro da preservare \$s0. L'algoritmo utilizza una seconda procedura, “procedAD”, per scambiare effettivamente i caratteri.

Procedura **procedAD**: procedura che effettua gli scambi dei caratteri utilizzando due registri di appoggio (\$t1 e \$t5) dove vengono salvati i caratteri all'indirizzo base contenuto all'interno del registro utilizzato.

PSEUDOCODICE:

```
i = 0
j = S.lunghezza - 1
WHILE (T3 < T2) {
  T1 = S0[i]
  T5 = S0[j]
  S0[i] = T5
  S0[j] = T1
  i + 1
  j - 1
}
ELSE SALTO A PROC. lbkey
```

Procedura **ALGORITMOE**: la procedura ALGORITMOE applica, con procedure secondarie, l'algoritmo E.

In questa procedura, oltre lo stack, per memorizzare i dati generati dall'algoritmo riguardanti i registri da preservare, viene implementata una linkedlist, una per ogni applicazione dell'algoritmo E, che consente di memorizzare in quali posizioni sono allocati i caratteri del messaggio da cifrare. La linkedlist è costruita, utilizzando il registro \$t2 come testa e \$t3 come coda, in modo che per ogni elemento che la compone si ha che: 4 byte vengono utilizzati per memorizzare l'indirizzo successivo all'elemento inserito, 4 byte sono dedicati al carattere che viene inserito (il quale viene estratto dal messaggio da cifrare) e 4 byte per la posizione in cui è allocato il carattere all'interno del messaggio da cifrare. L'algoritmo utilizzerà, dopo il carattere precedentemente inserito, 8 byte invece che 12 byte dove i primi 4 byte sono utilizzati per inserire l'indirizzo dell'elemento successivo mentre i secondi 4 byte sono utilizzati per memorizzare la posizione a cui è allocato il carattere del messaggio precedentemente inserito. I 4 byte riguardanti l'indirizzo della memoria dell'elemento successivo sono portati a zero nel caso dell'inserimento in coda. La distinzione tra numero, che caratterizza la posizione del carattere, e carattere ASCII del messaggio cifrato è la sezione di memoria che compone la linkedlist. Quando viene analizzata una porzione di memoria con indirizzo di base che 'punta' all'indirizzo dell'elemento successivo (es. lw \$t0, 0(\$t2)) avremo che con offset 8 (es. lw \$t0, 8(\$t2)) il dato estratto si differenzia da un numero ridotto, nel caso si abbia un carattere ASCII del precedente messaggio, o dall'indirizzo di una porzione di memoria, quindi un numero molto alto, che caratterizza il formato da 8 byte. Il dato da estrarre è in posizione 4(\$t2) e sarà un numero intero che rappresenta la posizione all'interno della parola da cifrare di un carattere.

Il messaggio cifrato viene composto facendo la distinzione, tra porzioni di memoria, precedentemente descritta inserendo il messaggio cifrato in una nuova porzione di memoria. Questo per non corrompere la porzione di memoria del messaggio su cui viene applicato l'algoritmo. Per calcolare la memoria necessaria viene utilizzata la procedura **calcT4**.

I registri \$t8 e \$t9 sono utilizzati, nelle procedure per l'inserimento di un carattere all'interno della linkedlist, per stabilire a che posizione è un determinato carattere all'interno del messaggio da cifrare. Il registro \$t9 viene

incrementato ogni qualvolta si seleziona il carattere da cercare, all'interno della linkedlist, mentre \$t8 viene utilizzato per scorrere il messaggio da cifrare dalla posizione \$t9 fino alla sua fine, per cercare in quali posizioni sono le parole da inserire. La ricerca, tramite la procedura **search**, cerca all'interno della linkedlist il carattere del messaggio preso in considerazione per la cifratura. Questa ricerca viene effettuata per evitare di inserire due volte un carattere già presente all'interno della linkedlist. La ricerca ha inizio dalla 'testa' della linkedlist (\$t2) ed arriva alla 'coda' (\$t3) quando la ricerca è senza successo e quindi il carattere da cifrare, caricato nel registro \$t5, è da inserire all'interno della struttura dati. L'inserimento avviene tramite la procedura **addT5**. Questa procedura è caratterizzata dalla chiamata a sistema SBRK, ricerca 12 byte in memoria (che funzionerà come estensione alla linkedlist già esistente), e dall'inserimento del carattere all'interno della linkedlist, con annessa posizione all'interno del messaggio da cifrare. La procedura count, in seguito all'assegnazione dei registri \$s0 e \$t9 ai registri \$t6 e \$t8, scorre il messaggio per memorizzare all'interno della linkedlist, dopo una chiamata a sistema SBRK, ogni posizione in cui è presente il carattere da cifrare.

PSEUDOCODICE:

ALGORITMO E: IF (T7) —> compS0

initLL: T2 ← 0 //TESTA

T3 ← 0 //CODA

selectT5:

T5 ← S0[i]

ctrlT5E: IF (T5 == 0) —> instanceLL

ELSE SALTO A PROC. search

instanceLL:

Chiamata a sistema SBRK (A0 = 12)

IF (T3 == 0) —> newLL

ELSE SALTO A PROC. CHIAMANTE

search: T6 = T2

IF (T6[0] == 0) —> addT5

assT7: T7 = T6[4]

IF (T5 == T7) —> updateT6

IF (T6 == T3) —> updateT6

T7 = T6[8]

IF (T7 == 0 or T7 > 200000000) —> updateT6

T9 + 1

SALTO A PROC. selectT5

updateT6:

T6 = T6[0]

SALTO A PROC. assT7

calcT4: T6 = T2

ctrlT7C: IF (T7 == 0) → SALTO A PROC. CHIAMANTE

T7 == T6[8]

IF (T7 > 200000000) → T7 = T6[4] SALTO AGLI IF

T4 + 1 // una posizione per il carattere

IF (T7 > 99) → IF (T7 > 999) → T4 + 5 // 5 posizioni per il trattino e le quattro cifre che compongono numeri maggiori di 999

ELSE T4 + 3 ELSE T4 + 4 // 3 o 4 posizioni, a seconda della grandezza del numero per il trattino e le cifre che compongono il numero che caratterizza la posizione del carattere del messaggio da cifrare

T6 = T6[0]

SALTO A PROC. ctrlT7C

Procedura **reCompS0**: questa procedura è invocata dalla procedura **ALGORITMOE** quando il programma è in fase di cifratura del messaggio. Questa procedura mira a comporre i valori contenuti all'interno della linkedlist per creare il messaggio cifrato e utilizza procedure secondarie per comporre il messaggio cifrato quali **calcT4** e **initSBRK**, il quale utilizzo è già noto. Vengono memorizzati due valori, 45 e 32, in due registri, rispettivamente \$t6 e \$t7, che rappresentano i valori ASCII dei caratteri “-“ e “ “.

Procedura **decNum**: questa procedura permette di trasformare i numeri interi, che caratterizzano la posizione all'interno del messaggio da cifrare, in valori ASCII, per essere rappresentabili graficamente o viceversa. Tramite il frame pointer viene memorizzata l'indirizzo in cui sono memorizzati i dati che richiedono il mantenimento di una posizione nota. Tramite il registro \$t6 viene inserito un “-“.

Procedura **cycleDIV & swapValue**: queste procedure tramite la divisione del valore contenuto in \$t5 permettono di eseguire il modulo e di sommare 48 al risultato ottenuto dal modulo. L'operazione di modulo è sostenuta attraverso la divisione per 10 del numero associato al registro \$t5 e dal trasferimento della parte alta della divisione nel registro \$t3. Il valore associato ad HI, in questo caso, può essere definito come il numero che viene trascritto dopo la virgola, quando dividiamo per 10, oppure come risultato del modulo 10 di un numero. Il valore associato a LO è associato, nel caso di una divisione per 10, alle restanti cifre che compongono il numero diviso per 10.

PSEUDOCODICE (procedura **decNum**):

i = 0

IF (T7 == 1) → multD

WHILE (T5 > 0){

T3 = (T5 mod 10) + 48

T5 = T5 / 10

APP[i] ← T3

i + 1

} ELSE {

WHILE (i == 0){

VO[j] ← APP[i]

i - 1

j + 1

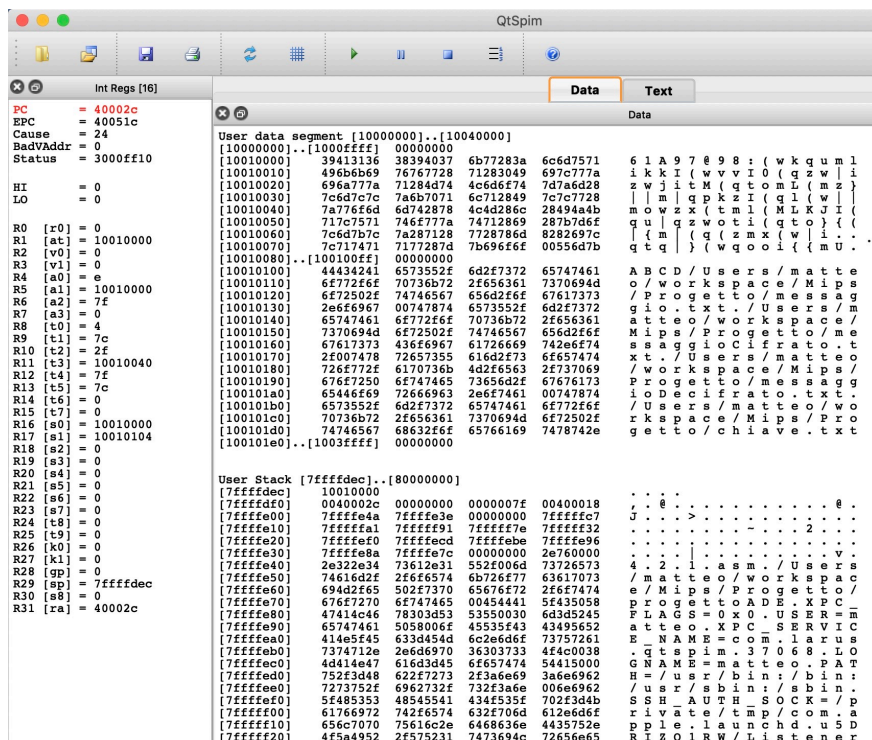
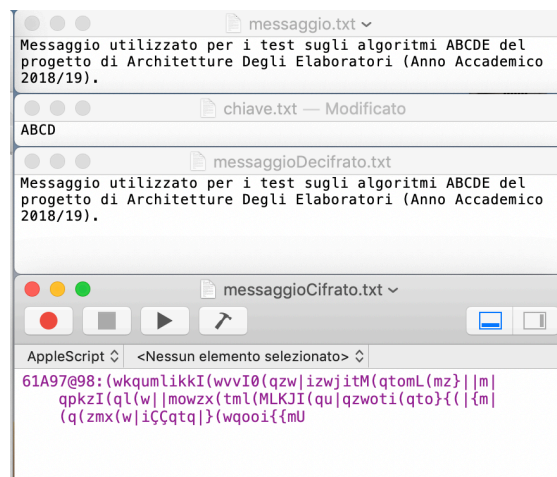
}

}

La rappresentazione tramite lo pseudocodice mostra che il vettore **APP** è, nel codice che compone il progetto, rappresentato dallo stack e l'indice i è rappresentato dall'indirizzo contenuto all'interno del registro \$sp. L'operazione di modulo viene effettuata tramite l'associazione del valore di HI (tramite istruzione specifica mfhi); il controllo del primo ciclo while è rappresentato dal salto condizionato beq \$t3, cycleDiv; infine il controllo sul secondo ciclo while è il salto condizionato bene \$fp, \$sp, swapValue.

Test di corretto funzionamento

Test con chiave "ABCD":



QtSpim screenshot showing registers, memory, and assembly code.

Test chiave E:

QtSpim screenshot showing registers, memory, and assembly code.

messaggio.txt

Messaggio utilizzato per i test sugli algoritmi ABCDE del progetto di Architetture Degli Elaboratori (Anno Accademico 2018/19).

chiave.txt

E

messaggioCifrato.txt

M-0 e-1-22-28-55-62-76-81-84-112 s-2-3-29-32
a-4-17-38-91-95-110 g-5-6-34-40-61-85
i-7-12-14-25-36-43-46-68-74-87-99-114
o-8-19-41-60-65-93-97-105-116
-9-20-24-26-31-37-47-53-57-66-69-82-88-100-106-117 u-10-33-79
t-11-18-27-30-44-63-64-75-77-78-96 l-13-35-39-56-86-90
z-15-16 p-21-58 r-23-42-59-71-80-94-98 m-45-113
A-48-70-102-107 B-49 C-50 D-51-83 E-52-89 d-54-67-111
C-72-108-109-115 h-73 b-92 (-101 n-103-104 2-118 0-119
1-120-123 8-121 -122 9-124)-125 -126

messaggioDecifrato.txt

Messaggio utilizzato per i test sugli algoritmi ABCDE del progetto di Architetture Degli Elaboratori (Anno Accademico 2018/19).

QtSpim screenshot showing registers, memory, and assembly code.

					Data	Text
					Data	
[10040000]	1004000c	0000004d	00000000	10040018	M
[10040010]	00000065	00000001	10040020	00000016	e
[10040020]	10040028	0000001c	10040030	00000037	(.	0 7
[10040030]	10040038	0000003e	10040040	0000004c	8	> @ L
[10040040]	10040048	00000051	10040050	00000054	H	Q P T
[10040050]	10040058	00000070	10040064	00000073	X	p d s
[10040060]	00000002	1004006c	00000003	10040074	l t
[10040070]	0000001d	1004007c	00000020	10040088
[10040080]	00000061	00000004	10040090	00000011	a
[10040090]	10040098	00000026	100400a0	0000005b	& [.
[100400a0]	100400a8	0000005f	100400b0	0000006e n
[100400b0]	100400bc	00000067	00000005	100400c4	- g
[100400c0]	00000006	100400cc	00000022	100400d4 "
[100400d0]	00000028	100400dc	0000003d	100400e4	(.	=
[100400e0]	00000055	100400f0	00000069	00000007	U	i
[100400f0]	100400f8	0000000c	10040100	0000000e
[10040100]	10040108	00000019	10040110	00000024 \$
[10040110]	10040118	0000002b	10040120	0000002e	+
[10040120]	10040128	00000044	10040130	0000004a	(.	D 0 J
[10040130]	10040138	00000057	10040140	00000063	8	W @ c
[10040140]	10040148	00000072	10040154	0000006f	H	r T o
[10040150]	00000008	1004015c	00000013	10040164	\ d
[10040160]	00000029	1004016c	0000003c	10040174)	l < t
[10040170]	00000041	1004017c	0000005d	10040184	A]
[10040180]	00000061	1004018c	00000069	10040194	a i
[10040190]	00000074	100401a0	00000020	00000009	t
[100401a0]	100401a8	00000014	100401b0	00000018
[100401b0]	100401b8	0000001a	100401c0	0000001f
[100401c0]	100401c8	00000025	100401d0	0000002f	% /
[100401d0]	100401d8	00000035	100401e0	00000039	5 9
[100401e0]	100401e8	00000042	100401f0	00000045	B E
[100401f0]	100401f8	00000052	10040200	00000058	R X
[10040200]	10040208	00000064	10040210	0000006a	d j
[10040210]	10040218	00000075	10040224	00000075	u \$ u
[10040220]	0000000a	1004022c	00000021	10040234 ! 4
[10040230]	0000004f	10040240	00000074	0000000b	O	@ t
[10040240]	10040248	00000012	10040250	0000001b	H P
[10040250]	10040258	0000001e	10040260	0000002c	X
[10040260]	10040268	0000003f	10040270	00000040	h	? p @
[10040270]	10040278	0000004b	10040280	0000004d	x	K M
[10040280]	10040288	0000004e	10040290	00000060	N
[10040290]	1004029c	0000006c	0000000d	100402a4	l
[100402a0]	00000023	100402ac	00000027	100402b4	#
[100402b0]	00000038	100402bc	00000056	100402c4	8 V h
[100402c0]	0000005a	100402d0	0000007a	0000000f	Z z
[100402d0]	100402d8	00000010	100402e4	00000070 p
[100402e0]	00000015	100402ec	0000003a	100402f8 t
[100402f0]	00000072	00000017	10040300	0000002a	r *
[10040300]	10040308	0000003b	10040310	00000047 ; G
[10040310]	10040318	00000050	10040320	0000005e P
[10040320]	10040328	00000062	10040334	0000006d	(.	b 4 m

					Data	Text
					Data	
[10040320]	10040328	00000062	10040334	0000006d	(.	b 4 m
[10040330]	0000002d	1004033c	00000071	10040348	-	< q H
[10040340]	00000041	00000030	10040350	00000046	A	0 P F
[10040350]	10040358	00000066	10040360	0000006b	x	f \ k
[10040360]	1004036c	00000042	00000031	10040378	l	B 1 x
[10040370]	00000043	00000032	10040384	00000044	C	2 D
[10040380]	00000033	1004038c	00000053	10040398	3 S
[10040390]	00000045	00000034	100403a0	00000059	E	4 Y
[100403a0]	100403ac	00000064	00000036	100403b4	d 6
[100403b0]	00000043	100403bc	0000006f	100403c8	C o
[100403c0]	00000063	00000048	100403d0	0000006c	c	H l
[100403d0]	100403d8	0000006d	100403e0	00000073	m s
[100403e0]	100403ec	00000068	00000049	100403f8	h I
[100403f0]	00000062	0000005c	10040404	00000028	b \ (.
[10040400]	00000065	10040410	0000006e	00000067	e n
[10040410]	10040418	00000068	10040424	00000032	h \$ 2
[10040420]	00000076	10040430	00000030	00000077	v	0 w
[10040430]	1004043c	00000031	00000078	10040444	<	1 x D
[10040440]	0000007b	10040450	00000038	00000079	{	P 8 y
[10040450]	1004045c	0000002f	0000007a	10040468	\ / z h
[10040460]	00000039	0000007c	10040474	00000029	9 t
[10040470]	0000007d	00000000	0000002e	0000007e	}

				Data	Text
				Data	
039	0000007c	10040474	00000029	9 t)
07d	00000000	0000002e	0000007e	}
44d	2d312d65	322d3232	35352d38	-	0 . . e - 1 - 2 2 - 2 8 - 5 5
22d	382d3637	34382d31	3231312d	M	6 2 - 7 6 - 8 1 - 8 4 - 1 1 2
22d	322d332d	32332d39	342d6120	-	s - 2 - 3 - 2 9 - 3 2 a - 4
20d	392d3833	35392d31	3031312d	-	1 7 - 3 8 - 9 1 - 9 5 - 1 1 0
32d	332d362d	30342d34	2d31362d	g	5 - 6 - 3 4 - 4 0 - 6 1 -
320	312d372d	34312d32	2d35322d	8	5 i - 7 - 1 2 - 1 4 - 2 5 -
333	36342d33	2d38362d	382d3437	3	6 - 4 3 - 4 6 - 6 8 - 7 4 - 8
337	3431312d	382d6f20	2d39312d	7	- 9 9 - 1 1 4 o - 8 - 1 9 -
334	35362d30	2d33392d	312d3739	4	1 - 6 0 - 6 5 - 9 3 - 9 7 - 1
330	20203631	322d392d	34322d30	0	5 - 1 1 6 - 9 - 2 0 - 2 4
32d	332d3133	37342d37	2d33352d	-	2 6 - 3 1 - 3 7 - 4 7 - 5 3 -
335	39362d36	2d32382d	312d3838	5	7 - 6 6 - 6 9 - 8 2 - 8 8 - 1
330	312d3630	75203731	2d30312d	0	0 - 1 0 6 - 1 1 7 u - 1 0 -
333	2d742039	312d3131	37322d38	3	3 - 7 9 t - 1 1 1 - 1 8 - 2 7
32d	362d3434	34362d33	2d35372d	-	3 0 - 4 4 - 6 3 - 6 4 - 7 5 -
737	36392d38	312d6c20	35332d33	7	7 - 7 8 - 9 6 l - 1 3 - 3 5
327	382d3635	30392d36	312d7a20	-	3 9 - 5 6 - 8 6 - 9 0 z - 1
435	322d7020	38352d31	322d7220	5	- 1 6 p - 2 1 - 5 8 r - 2
433	2d39352d	382d3137	34392d30	3	- 4 2 - 5 9 - 7 1 - 8 0 - 9 4
42d	35342d6d	3331312d	342d4120	-	9 8 m - 4 5 - 1 1 3 A - 4
438	3230312d	3730312d	342d4220	8	- 7 0 - 1 0 2 - 1 0 7 B - 4
039	44203035	2d31352d	45203338	9	C - 5 0 D - 5 1 - 8 3 E
03d	64203938	2d34352d	312d3736	-	5 2 - 8 9 d - 5 4 - 6 7 - 1
331	2d32372d	2d383031	2d393031	1	1 c - 7 2 - 1 0 8 - 1 0 9 -
331	33372d68	392d6220	2d282032	1	1 5 h - 7 3 b - 9 2 (-
331	30312d6e	30312d33	2d322034	1	0 1 n - 1 0 3 - 1 0 4 2 -
331	31312d30	2d312039	2d303231	1	1 8 0 - 1 1 9 1 - 1 2 0 -
331	32312d38	2d2f2031	20323231	1	2 3 8 - 1 2 1 / - 1 2 2
439	2d292034	20353231	32312d2e	9	- 1 2 4) - 1 2 5 . - 1 2
036	00000000	00000000	00000000	6
27j	00000000				
44d	7a696c69				
06f	79696c69	6f74617a	72657020	M	e s s a g g i
207	20747365	6c677573	6c612069	u	t i l i z z a t o p e r
620	20696d74	44434241	65642045	i	t e s t s u g l i a l
06c	7465676f	64206f74	72412069	g	o r i t m i A B C D E d e
363	75747465	44206572	696c6765	l	p r o g e t t o r d i A r
260	61726f62	69726f74	6e412820	h	i t e t t u r e D e g l i
62e	64616363	63696d65	3032206f	E	l a b o r a t o r i o (A n
031	000e2939	00000000	00000000	n	o A c c a d e m i c o 2 0
fbj	00000000			1	8 / 1 9)
User Stack [7ffffe00]..[80000000]					
44a	7fffffe3e	00000000	7fffffc7	J >
fa1	7fffff91	7fffff7e	7fffff32 ~ 2
2f0	7ffffecd	7ffffe96	7ffffe96
83a	7ffffefc	00000000	2e7f0000 v .
434	73612e31	552f006d	73726573	4	. 2 . 1 . a s m . / U s e r s

CODICE

```
.data
bufferMsg: .space 256 #direttiva utilizzata per allocare 256 byte per il messaggio
bufferKey: .space 4 #direttiva utilizzata per allocare 4 byte per la chiave
#percorso dei file utilizzati per la cifratura dei messaggi
nameFileInput: .asciiz "/Users/matteo/workspace/Mips/Progetto/messaggio.txt"
nameFileOutput: .asciiz "/Users/matteo/workspace/Mips/Progetto/messaggioCifrato.txt"
nameFileOutputReverse: .asciiz "/Users/matteo/workspace/Mips/Progetto/messaggioDecifrato.txt"
nameFileKey: .asciiz "/Users/matteo/workspace/Mips/Progetto/chiave.txt"

.text
main:  sw $ra, 0($sp)  #inserimento dell'indirizzo di ritorno dell'exception handler
      jal open_FileMsg
      j setRegReverseProcess
open_FileMsg:
      li $v0, 13      #carico il numero 13 nel registro $v0
      la $a0, nameFileInput  #carico l'indirizzo in memoria 'nameFileInput'
      move $a1, $zero      #imposto flag e modalità per poter solamente leggere dopo l'apertura del file
      move $a2, $zero
      syscall          #chiamata a sistema e passaggio del descrittore del file
      move $a0, $v0
read_Msg:
      li $v0, 14      #carico il numero 14 nel registro $v0
      la $a1, bufferMsg  #carico l'indirizzo in memoria 'bufferMsg'
      li $a2, 256      #imposto la lunghezza del buffer (256 byte)
      syscall          #chiamata a sistema
      la $s0, bufferMsg #carico l'indirizzo del messaggio in $s0 e verra' utilizzato come registro di base applicare gli
algoritmi
      sub $sp, $sp, 4 #preservo l'indirizzo del registro $ra (indirizzo dell'istr. j setRegReverseProcess) per poter
invocare la procedura 'close'senza perdere il valore del registro
      sw $ra, 0($sp)
      move $t4, $v0 #al registro $v0 è associato il numero di caratteri contenenti all'interno del file aperto.
      jal close
```

open_FileKey: #'open_fileKey' e 'read_Key' sono procedure analoghe a quelle riguardanti l'apertura del file "messaggio.txt"

```
li $v0, 13
la $a0, nameFileKey
move $a1, $zero
move $a2, $zero
syscall
move $a0, $v0
```

read_Key:

```
li $v0, 14
la $a1, bufferKey
li $a2, 4
syscall
la $s1, bufferKey
jal close
```

lw \$ra, 0(\$sp) #estrazione dell'indirizzo di ritorno precedente alla chiusura del file

storeSP: sub \$sp, \$sp, 12 #sottrazione di 3 parole per preservare l'indirizzo del messaggio da de/cifrare, l'indirizzo di ritorno e il registro \$t7. Il registro \$ra contiene l'indirizzo successivo al salto incondizionato contraddistinto dall'utilizzo della jal; il registro \$t7 contiene un selettore, così da poter selezionare la fase in cui un algoritmo deve operare (fase in cui viene cifrato, valore 0, e fase in cui viene decifrato il messaggio, valore 1) ed il registro \$t4 è utilizzato per tenere traccia del numero dei caratteri da cui è composta una parola

```
sw $s0, 0($sp) #dati da preservare
sw $ra, 4($sp)
sw $t7, 8($sp)
sw $t4, 12($sp)
```

j lbKey #non vi e' necessita' di ricaricare gli indirizzi del messaggio e del ritorno perche' gia' corretti

selectAlg:

```
lw $s0, 0($sp)
lw $ra, 4($sp)
```

lbKey: lb \$t2, 0(\$s1) #carichiamo byte per byte il contenuto della chiave per poi applicare al messaggio ogni algoritmo di cifratura

ctrl: blt \$t2, 65, decisionFileOut # i valori compresi tra 65 e 69 (codice ASCII in base 10) rappresentano le lettere A-B-C-D-E e quando il valore di \$t2 e' diverso gli algoritmi da applicare sono terminati ed e' possibile scrivere il messaggio nel file apposito

```
bgt $t2, 69, decisionFileOut
```

beq \$t7, 1, subS2 # se il programma è in fase di decifratura viene decrementato di un byte l'indirizzo della chiave

```

    addi $s1, $s1, 1
select: ble $t2, 66, ALGORITMOABC#scelta dell'algoritmo da applicare
        beq $t2, 67, ALGORITMOC
        beq $t2, 68, ALGORITMOD
        beq $t2, 69, ALGORITMOE
        j lbKey
decisionFileOut:
        beqz $t7, open_Fileout # quando $t7 e' asserito scriviamo in "messaggioCifrato.txt" oppure in
"messaggioDecifrato.txt"
        j open_FileMsgCripted
subS2: sub $s1, $s1, 1
        j select
ALGORITMOC:
        addi $s0, $s0, 1 #l'algoritmo C richiede di cifrare i caratteri contenuti in posizioni dispari e come indirizzo di
parte verra' incrementato l'indirizzo di base di un byte
ALGORITMOABC:
        beq $t7, 1, assT0
        li $t0, 4 #valore utilizzato per la somma di 4 posizioni del valore ASCII del carattere
assT5ABC:
        lb $t5, 0($s0) #lettura del primo carattere all'indirizzo 0($s0)
ctrlT5: beqz $t5, selectAlg #arrivato al termine del buffer del messaggio viene richiamata la procedura selectAlg
        add $t5, $t5, $t0#somma di 4 o -4
div256: bgt $t5, 255, divMod #modulo 256 del valore del carattere per poter dare una rappresentazione anche agli
ultimi 15 caratteri ASCII quando vengono applicate i cifrari A-B-C una o piu' volte
store: sb $t5, 0($s0) #scrittura del carattere cifrato
        beq $t2, 65, ALGORITMOA #selezione dell'incremento dell'indirizzo, a seconda delle richieste
dell'algoritmo
        ble $t2, 67, ALGORITMOBC
        j assT5ABC
assT0: li $t0, -4 #in fase di decifratura si assegna il valore -4 per procedere al contrario rispetto alla cifratura
        j assT5ABC
ALGORITMOA:
        addi $s0, $s0, 1 #avanzamento di un byte / carattere
        j ALGORITMOABC
ALGORITMOBC:
        addi $s0, $s0, 2 #avanzamento di due byte / carattere

```

j assT5ABC

ALGORITMOD:

add \$t2, \$s0, \$t4 #somma all'indirizzo in memoria del messaggio del numero di caratteri -1

move \$t3, \$s0 #assegnazione di \$s0 a \$t3 per evitare di modificare \$s0

procedAD:

sub \$t2, \$t2, 1

bgt \$t3, \$t2, lbKey #appena i due valori si incrociano \$t3 diventa maggiore di \$t2, viene invocata la procedura

lbKey lb \$t1, 0(\$t3)

lb \$t5, 0(\$t2)

sb \$t1, 0(\$t2)

sb \$t5, 0(\$t3)

addi \$t3, \$t3, 1

j procedAD

ALGORITMOE:

beq \$t7, 1, compS0 #fase di decifratura

initLL: move \$t2, \$zero #inizializzazione dei registri usati nelle procedure seguenti \$t2 = testa LINKEDLIST

move \$t3, \$zero #coda LINKEDLIST

move \$t9, \$zero # \$t9 e' il registro utilizzato come contatore ed e' tenuto aggiornato per sapere, passo passo, il numero della posizione del carattere rispetto all'inizio della parola durante l'esecuzione dell'algoritmo

selectT5:

lb \$t5, 0(\$s0) #estrazione del carattere

ctrlT5E:

beqz \$t5, reCompS0 #se \$t5 e' uguale a 0 invoca la procedura che ricompone il messaggio

add \$s0, \$s0, 1

ctrlLL: beqz \$t2, instanceLL #quando \$t2 e' uguale a 0 (l'elemento in testa non esiste) senno' cerca il carattere (\$t5)

j search

instanceLL: #chiamata sbrk. Istanza uno spazio di 3 words (12 byte)

li \$v0, 9

li \$a0, 12

syscall

beqz \$t3, newLL #se la coda non è stata inizializzata viene invocata la procedura newLL

jr \$ra

newLL: move \$t2, \$v0 #inizializzazione della LinkedList

move \$t3, \$v0 #testa (\$t2) = coda (\$t3)

sw \$zero, 0(\$t2) # 0-3 byte caratterizzati dall'indirizzo all'elemento successivo contenuto nella linkedlist

```

sw $t5, 4($t2) #4-7 byte caratterizzati dal contenere il carattere contenuto nel messaggio
sw $zero, 8($t2) #8-11 byte, posizione in cui è allocato il carattere nel messaggio da cifrare
j count
addT5: add $t9, $t9, 1 #viene incrementato di 1 perche' altrimenti rimarrebbe al valore della precedente parola
jal instanceLL
assV0A:sw $v0, 0($t3) #inserisco nuovo indirizzo della coda
move $t3, $v0 #aggiorno coda della linkedlist con il nuovo indirizzo
sw $zero, 0($t3) #inserisco 0 perche' e' l'ultimo elemento della linkedlist (0-3 byte)
sw $t5, 4($t3) #inserisco il carattere da cifrare (4-7 byte)
sw $t9, 8($t3) #inserisco posizione del carattere da cifrare (8-11 byte)
count: move $t6, $s0 #inizializzazione del registro $t6 con l'indirizzo della parola per evitare di modificare $s0
move $t8, $t9 # $t8 utilizzato dalla posizione contenuta in $t9 alla fine del messaggio
ASST7E:
lb $t7, 0($t6) #estraggo il carattere da confrontare con il carattere da cifrare
add $t8, $t8, 1 #aggiorno la posizione del carattere estratto
ctrlT7: beqz $t7, selectT5 #se $t7 e' uguale a zero la procedura e' a fine messaggio da cifrare
beq $t7, $t5, trueT7e5 #se i caratteri, quello estratto dal messaggio da cifrare e quello da cifrare, hanno lo
stesso valore ASCII viene inserito il valore del registro $t8 all'interno della linkedlist
incrT6: add $t6, $t6, 1 #avanzamento di un carattere del messaggio
j ASST7E
trueT7e5:
li $v0, 9 #allocazione spazio per inserire il valore associato a $t8 (4 byte) e l'indirizzo dell'elemento
successivo (4 byte)
li $a0, 8
syscall
assV0C:sw $v0, 0($t3) #inserimento del nuovo indirizzo della coda
move $t3, $v0 #aggiorno l'indirizzo della 'coda' della linkedlist
sw $zero, 0($t3) #inserisco 0 nei primi 4 byte perche' e' l'ultimo elemento della linkedlist
sw $t8, 4($t3) #inserisco $t8 all'interno della linked list
j incrT6
reCompS0:
jal calcT4 #calcolo del numero di caratteri del messaggio cifrato
li $t6, 45 #assegnazione dei valori "-" e " " con il valore ASCII in base 10
li $t7, 32
jal initSBRK #allocazione della memoria per il messaggio cifrato
sw $v0, 0($sp) #aggiornamento del indirizzo del messaggio cifrato

```

firstComp:

```
lw $t5, 4($t2) #estrazione primo carattere
sw $t5, 0($v0) #inserimento del primo carattere
addi $v0, $v0, 1 #scorrimento di un carattere l'indirizzo base del messaggio cifrato
move $t5, $zero #trasferimento di zero in $t5
j decNun
```

calcT4: move \$t6, \$t2 #duplico l'indirizzo contenuto in \$t2

```
li $t7, 1 #per evitare il salto condizionato assegnazione di 1 al registro $t7
```

cfrT7C: beqz \$t7, jump #controllo sul valore contenuto all'indirizzo 0(\$t6). la seguente istruzione solleverebbe un'eccezione se non esiste l'elemento ad indirizzo 8(\$t6)

```
lw $t7, 8($t6) #caricamento del valore all'indirizzo 8($t6)
```

bgt \$t7, 200000000, assT7 #quando persiste un valore alto nella porzione di memoria 8(\$t6) il contenuto da utilizzare nel messaggio e' solo un numero (contenuto all'indirizzo 4(\$t6))

bgt \$t7, 99, assT5 #quando non si verifica la condizione precedente i valori validi da inserire nel messaggio cifrato sono contenuti agli indirizzi con offset 4 e 8(\$t6) ed a seconda della grandezza dei loro valori verra' incrementato il valore associato a \$t4 (utilizzato poi dalla procedura initSBRK quando viene eseguito il salto condizionato beqz \$t7, jump)

addi \$t4, \$t4, 3 #3 posizioni per il carattere del messaggio cifrato, la posizione ed il trattino da posizionare tra carattere e numero

seT50: move \$t5, \$zero # \$t5 e' utilizzato come selettore per il conteggio dei caratteri del messaggio cifrato

```
addi $t4, $t4, 1
```

assT7: beq \$t5, 1, seT50

```
lw $t7, 4($t6)
```

bgt \$t7, 99, t7gt99 #3 o 4 posizioni, a seconda della grandezza del numero per il trattino e le cifre che compongono il numero che caratterizza la posizione del carattere del messaggio da cifrare

```
addi $t4, $t4, 3
```

```
j incrT6C
```

assT5: li \$t5, 1

```
bgt $t7, 999, t7gt999
```

t7gt99: addi \$t4, \$t4, 4

cfrT5.1: beq \$t5, 1, assT7

incrT6C:

```
lw $t6, 0($t6) #aggiornamento all'elemento successivo della linkedlist
```

```
beqz $t6, jump #controllo sulla posizione della linkedlist
```

```
lw $t7, 0($t6) #estrazione del
```

```
j cfrT7C
```

t7gt999: addi \$t4, \$t4, 5 # 5 posizioni per il trattino e le quattro cifre che compongono i numeri maggiori di 999

j cfrT5.1

cfrT2: beqz \$t2, stackPointerVar #salto condizionato, se viene eseguito e' stata composto completamente il messaggio cifrato

assT58: lw \$t5, 8(\$t2) #assegnazione del valore all'indirizzo 8(\$t2)

bgt \$t5, 200000000, T5eq0R#se il valore \$t5 il contenuto da prendere in considerazione è quello all'indirizzo 4(\$t2) e rappresenta la posizione di un carattere all'interno del messaggio da cifrare (organizzazione a 8 byte)

beqz \$t5, T5eq0R #in ultima posizione della linkedlist \$t5 assume valore 0 e quindi il valore da estrarre e' il numero in posizione 4(\$t2)

lw \$t8, 4(\$t2)#il valore contenuto in \$t5 e' un intero che rappresenta il carattere all'indirizzo 4(\$t2) (organizzazione a 12 byte)

storeT75:

sb \$t7, 0(\$v0)#347 viene inserito uno spazio per distanziare gli interni (le posizioni del precedente carattere) dal carattere da inserire (con analoghe posizioni)

sb \$t8, 1(\$v0)#inserimento del carattere estratto come ultimo carattere del messaggio cifrato

incrV0: addi \$v0, \$v0, 2# avanzamento di due posizioni a causa dell'inserimento dello spazio e del carattere

j decNum

T5eq0R:

lw \$t5, 4(\$t2)

j decNum

cfrT5: beqz \$t5, stackPointerVar #nel caso che anche l'indirizzo all'elemento

j assT58

decNum:

beq \$t7, 1, multD

move \$fp, \$sp

sb \$t6, 0(\$v0)#scrittura del carattere "-"

addi \$v0, \$v0, 1 #incremento dovuto all'aggiunta del trattino

cycleDIV:

div \$t5, \$t5, 10

mfhi \$t3

addi \$t3, \$t3, 48 #somma che permette di rappresentare le cifre (0-9)dei numeri, quindi scomposti cifra per cifra

sub \$sp, \$sp, 1

sb \$t3, 0(\$sp)#inserimento del numero all'interno dello stack

mflo \$t3

bnez \$t3, cycleDIV #ciclo fin quando non saranno inserite tutte le cifre all'interno dello stack, utilizzato come appoggio per evitare di scrivere al contrario (posizione 125 diventerebbe 521) le lettere nel messaggio cifrato (non conoscendo la quantita' precisa di cifre)

swapValue:

```
lb $t3, 0($sp)
sb $t3, 0($v0)
addi $v0, $v0, 1
addi $sp, $sp, 1
```

bne \$fp, \$sp, swapValue#ciclo per inserire correttamente le cifre del che compongono la posizione in cui e' presente il carattere

incrT2: lw \$t2, 0(\$t2) #incrementa \$t2 effettuare un controllo sulla 'testa' della linkedlist

```
move $t5, $zero
j cfrT2
```

multD: move \$t6, \$zero#in fase di decifratura del messaggio i numeri interi che rappresentano la posizione in cui il carattere era allocato prima della precedente applicazione dell'algoritmo e devono essere riportati a numeri utilizzabili dall'algoritmo

li \$t3, 1 #caricamento necessario per inizializzare il registro \$t3 (utilizzato per moltiplicare per multipli di 10 i numeri derivati dalla conversione del messaggio cifrato)

cycleMULT:

```
lb $t5, 0($s0)
beq $t5, 45, refrSp#appena incontra il valore "-" o con $t3>1000 salta alla procedura refrSp (restituisce il numero derivato dalla sequenza trasformata in precedenza come sequenza di valori, in base 10, compresi tra 48 e 57)
bgt $t3, 1000, refrSp
sub $t5, $t5, 48
mul $t5, $t5, $t3
mul $t3, $t3, 10
add $t6, $t6, $t5
sub $s0, $s0, 1
j cycleMULT
```

search: move \$t6, \$t2#inizializzazione \$t6 con l'indirizzo contenuto in \$t2 per evitare di modificare \$t2

firstSearch:

```
j asst7#salto incondizionato al confronto tra caratteri
```

updateT6:

```
lw $t6, 0($t6) #incremento di $t6 all'indirizzo dell'elemento successivo della linkedlist
```

assT7S: beqz \$t6, addT5 #se \$t6 e' uguale a 0 non e' presente il carattere all'interno della linkedlist e va inserito con le posizioni in cui e' presente all'interno del messaggio da cifrare

asst7: lw \$t7, 4(\$t6) #caricamento del valore da confrontare con quello da inserire, se sono diversi aggiorna \$t6

cfrT57: bne \$t5, \$t7, updateT6

ctrlS: sub \$t7, \$t6, \$t3#421 sottrazione e controllo dovuti alla verifica sull'indirizzo in \$t6. se uguale a \$t3 la ricerca è arrivata a fine linkedlist e quindi il carattere associato a \$t5 deve essere inserito nella linkedlist

```

    beqz $t7, addT5
    lw $t7, 8($t6)
    bgt $t7, 200000000, updateT6
    beqz $t7, updateT6
addT9: addi $t9, $t9, 1
        j selectT5
compS0:
    lw $t4, 12($sp)#carica il numero di caratteri preventivamente calcolato
    jal initSBRK#chiamata SBRK per comporre il messaggio decifrato
    j compSP
assT2: lb $t2, 0($s0)#carica il carattere che verra' inserito nelle posizioni calcolate in base al messaggio cifrato
incrS0: addi $s0, $s0, 2#incremento di 2 posizioni dovuto dal valore "-" che divide il carattere dall'intero che
rappresenta la posizione nella quale deve essere inserito. (M-0 e-1-ecc.)
assT8: lb $t8, 1($s0)#estrazione del valore in posizione 1($s0)
        beq $t8, 45, trueT5eq4532#se il valore estratto e' un "-" o " " significa che all'indirizzo $s0 e' presente l'ultima
cifra del numero intero che contraddistingue la posizione che deve assumere il carattere associato al registro $t2
        beq $t8, 32, trueT5eq4532
        beqz $t8, trueT5eq4532
        addi $s0, $s0, 1
        j assT8#fin quando non viene associato al registro $t8 il valore "-" o " " $s0 viene incrementato di uno
compSP:      addi $sp, $sp, 4 #aggiornamento dello stack (diminuisce di una parola lo stack perche' non e' piu'
necessario il valore all'indirizzo 12($sp)
        sw $v0, 0($sp)
        sw $ra, 4($sp)
        sw $t7, 8($sp)
        sw $t4, 12($sp)
        j assT2
trueT5eq4532:
    sub $sp, $sp, 8#$t8 e' uguale a "-" o " " e quindi vengono preservati i valori dei registri $s0 e $ra
    sw $s0, 0($sp)
    sw $ra, 4($sp)
    j decNum
refrSp: lw $s0, 0($sp)#ripristina i registri da preservare
        lw $ra, 4($sp)
        addi $sp, $sp, 8

```

posT2: add \$v0, \$v0, \$t6 #utilizzando il valore restituito dalla porcedura multD avanza delle posizioni necessarie all'inserimento del carattere nella messaggio da decifrare

sb \$t2, 0(\$v0)

sub \$v0, \$v0, \$t6

beq \$t8, 32, trueT8#quando il valore successivo alla posizione nella quale si inserisce il carattere e' uno spazio si incrementa di due l'indirizzo di \$s0

addi \$s0, \$s0, 1

beqz \$t8, T8eq0

j assT8

T8eq0: jal count_CharMsg#procedura utilizzata per aggiornare il valore \$t4 all'interno dello stack

sw \$t4, 12(\$sp)

j selectAlg

trueT8: addi \$s0, \$s0, 2

j assT2

initSBRK: #chiamata a sistema SBRK

li \$v0, 9

move \$a0, \$t4

syscall

jr \$ra

divMod: div \$t5, \$t5, 256 #calcolo del modulo 256

mfhi \$t5

j store

stackPointerVar:

jal count_CharMsg#ripristina i valori nello stack e ricalcola il numero di caratteri a causa della variabilita' dei caratteri generati dall'applicazione dell'algoritmo e

lw \$s0, 0(\$sp)

lw \$ra, 4(\$sp)

lw \$t7, 8(\$sp)

sub \$sp, \$sp, 4

sw \$s0, 0(\$sp)

sw \$ra, 4(\$sp)

sw \$t7, 8(\$sp)

sw \$t4, 12(\$sp)

j lbKey

open_Fileout:#apertura del file di scrittura del messaggio cifrato

lw \$t4, 12(\$sp)#estrazione numero di caratteri da scrivere all'interno del file

```

    li $v0, 13
    la $a0, nameFileOutput
    li $a1, 1
    li $a2, 1
    syscall
    move $a0, $v0
    lw $ra, 4($sp)
    j write_File
setRegReverseProcess:
    sub $s1, $s1, 1
    li $t7, 1 #caricamento 1, inserimento in stack. inizio fase di decifratura
    sw $t7, 8($sp) #modifica del valore nello stack
    j lbKey
open_FileMsgCripted:#apertura del file "messaggioDecifrato.txt"
    li $v0, 13
    la $a0, nameFileOutputReverse
    li $a1, 1
    move $a2, $zero
    syscall
    move $a0, $v0
    addi $sp, $sp, 16#aggiornamento dello stack
    lw $ra, 0($sp)
    addi $sp, $sp, 4
write_File:
    li $v0, 15
    la $a1, ($s0)
    move $a2, $t4 #numero di caratteri che compone il messaggio da scrivere su file
    syscall
close:  li $v0, 16      #procedura creata per risolvere eventuali errori derivati dal mantenere "aperti" file
        syscall
jump:   jr $ra          #procedura utilizzata per eseguire salti incondizioni all'indirizzo contenuto in $ra
count_CharMsg:
    move $t4, $zero
    lw $s0, 0($sp)
lbT5:   lb $t5, 0($s0)

```

beq \$t5, \$zero, jrRa

addi \$s0, \$s0, 1

addi \$t4, \$t4, 1

j lbT5

jrRa: lw \$s0, 0(\$sp)#permette di effettuare un salto incondizionato e ripristinare \$s0 dopo aver eseguito il conteggio dei caratteri

jr \$ra