Progetto Assembly RISC-V per il Corso di Architetture degli Elaboratori – A.A. 2019/2020 –

Messaggi Cifrati

Informazioni

Verione RIPES: v2.1.0-6-gac5e783

Descrizione

Descrizione ad alto livello

Il main si occupa di caricare le stringhe myplaintext e mycypher ed eseguire quattro procedure.
La prima procedura si occupa di controlla la correttezza delle stringhe caricate. Se il risultato di questa procedura da esito negativo verrà stampato a video una stringa che ci informa che non è andato a buon fine il controllo, in caso di esito positivo il main lancia la procedura che stampa a video il plain text, dopo di che viene eseguita la procedura di cifratura. Essa si occupa di leggere il carattere della stringa mycypher e applicare di conseguenza l'algoritmo di cifratura idoneo. Quando l'algoritmo di cifratura selezionato termina viene lanciato la procedura che stampa a video. Essa ha il compito di stampare su una nuova riga la stringa myplaintext (che adesso viene anche identificata con il temine cypthertext). Riprende la procedura di cifratura che legge il carattere successivo della stringa mycypher ed esegue i successivi algoritmi sul testo cifrato dall'algoritmo precedente. Una volta terminata la stringa riprende l'esecuzione del main per eseguire la procedura che si occupa della decifratura della stringa myplaintext (cypthertext) che è stata criptata dagli algoritmi contrassegnati nella stringa mycyher. Nella decifratura la stringa mycypher viene letta al contrario, e i relativi algoritmi vengono quindi applicati in ordine invero. Il comportamento di questa procedura è simile a quello della cifratura, quindi una volta selezionata l'algoritmo ed eseguito, si ritorna alla decifratura che lancia la procedura per la stampa del risultato. Questo procedimento si ripete fino a quando non è stato completamente attraversata la stringa mycypher, che quindi corrisponde alla completa decifrazione del messaggio. Per concludere si torna al main che si occupa di saltare alla fine del programma, terminando l'esecuzione.

In dettaglio
La procedura di controllo analizza per prima cosa la stringa mycypher accertandosi che ogni carattere sia uno tra 'A', 'B', 'C', 'D' e contestualmente viene controllato che la stringa non sia più lunga di 5, tramite lo stesso indice utilizzato per l'iterazione. La procedura termina nel caso in cui fallisca un controllo o se la stringa risulti vuota, stampando a video un messaggio di errore.
In caso contrari il programma passa al controllo della stringa myplaintext. Su di essa i passaggi sono similare ai precedenti. Viene controllato che i caratteri siano compresi tra il valore ASCII 32 e 127 che corrispondono a 'Spc' e 'Del', contestualmente si controlla che la lunghezza della stringa non superi i 100 caratteri. Anche in questo caso il controllo termina con un messaggio di errore se si riscuotano violazioni oppure se la stringa è vuota. Nel caso in cui tutti i controlli siano rispettati la procedura restituisce il controllo al main.
La procedura che stampa a video si occupa di stampare il plain text e di ritornare al main.
La procedure di cifratura come prima cosa salva nello stack il registro per ritornare al main. Il compito di questa procedura è appunto quello di estrarre il carattere che specifica l'algoritmo di cifratura da applicare al plain text. Viene quindi iterata la stringa mycypher e il carattere estratto viene passato da una procedura che si occupa si identificare a quale algoritmo il carattere estratto corrisponda. Dopo che viene terminato l'algoritmo selezionato si riprende l'esecuzione da dove era stata interrotta, per eseguire un ulteriore passaggio che si occupa di stampare su una nuova riga la stringa appena cifrata. Riprende il ciclo sulla stinga mycypher che porta alla ripetizione dei passi descritti, oppure se risulta terminata viene definita una variabile con la lunghezza della stringa e ritorna al main.
Nel main viene lanciata la procedura di decifratura che si comporta in modo simile alla cifratura. Viene eseguita la lettura della stringa mycypher in ordine inverso grazie alla variabile calcolata durante la cifratura, che definisce la lunghezza della stringa. Condivido con la cifratura la procedura per la selezione del algoritmo da eseguire e per la stampa del risultato. Si differenzia nell'inizializzare un flag che permette di identificare all'interno degli algoritmi il caso deve essere applicata la decifratura. Una volta attraversato

completamente la stringa mycypher e di conseguenza applicati tutti gli algoritmi, viene ritornato al main che quindi può terminare il programma.

Algoritmo di Cesare
Nel caso che la lettera estratta dalla stringa mycypher sia 'A' viene eseguito l'algoritmo di cesare che si compone in una fase in cui viene calcolato il modulo dello shif alfabetico da applicare successivamente successivamente alla stringa myplaintext/cyphertext. Questo calcolo si effettuato poiché la formula viene applicata solo alle lettere dell'alfabeto quindi lo scorrimento è al massimo 26 posizioni. Nel caso nostro poiché il modulo non è presente come istruzione, si sostituisce con l'operazione di resto per i valori dello shif positivi, mentre per valori dello shift negativo il risultato viene sottratto a 26, che equivale al modulo di un valore negativo. Successivamente si occupa di iterare sulla stringa in modo tale da estrarre a ogni ciclo un carattere. Una volta ottenuto viene eseguita una procedura per calcolare l'offset. Tale procedura nel caso non ritorni una lettera lascia immutato tale carattere e continua l'iterazione sulla stringa, nel caso in cui sia una lettera può ritornare il valore 97 se minuscola o 65 se maiuscola. L'algoritmo a questo punto si preoccupa di controllare se siamo nella cifratura o nella decifratura tramite un flag. Nello stato di cifratura viene applicata la seguente formula {[(l - o) + k] % 26} + o, 'l' corrisponde al codice ASCII della lettera, 'o' rappresenta il valore dell'offset riportato dalla precedente procedura e 'k' è lo shift alfabetico, il risultato della formula viene memorizzato e si passa al carattere successivo ripetendo i passi precedentemente descritti. Nella decifratura si va ad applicare la formula inversa ma con delle modifiche dovute alla mancanza del istruzione modulo. Conseguentemente se il seguente calcolo [(l - o) – k] risulta positivo, si applica la formula {[(l - o) - k] + o} rimossa del modulo poiché non necessario, nel caso in cui invece sia negativo la formula da applicare è {[((l - o) - k) + 26] + o} alla quale è stato aggiunto 26 poiché impossibilitati ad applicare il modulo di un valore negativo. L'algoritmo termina con il ripristino dei valori dallo stack e il ritorno alla procedura di cifratura o decifratura a seconda della situazione in cui si trova.

Algoritmo a Blocchi
Nel caso che la lettera estratta dalla stringa mycypher sia 'B' viene eseguito l'algoritmo a blocchi.
La stringa myplaintext/cyphertext viene iterata per estrarre il carattere sul quale applicare l'algoritmo, il qui codice ASCII viene diminuito di 32 applicando preventivamente parte della formula dell'algoritmo a blocchi. Conseguentemente viene iterata la stringa contente la chiave da utilizzare durante la formula, se la chiave risulta più corta della stringa sulla quale viene applicata, il ciclo di iterazione della chiave riparte dall'inizio, lasciando inalterato quello della stringa. Anche la chiave viene diminuita di 32 e successivamente viene controllato se l'algoritmo sia di cifratura o decifratura. Nel caso della cifratura viene applicata la seguente formula {[(cod(b ij) – 32) + (cod(key j) – 32)] % 96} + 32, di cui (cod(b ij) – 32) e (cod(key j) – 32) calcolati precedentemente, viene utilizzata l'istruzione resto di 96 per lo stesso motivo spiegato nel algoritmo di cesare. Il risultato della formula viene memorizzato e si riprende il ciclo con i passi precedentemente descritti. Nel caso della decifratura viene controllato che ((cod(bij) –32) - (cod(blocKeyj) –32)) risulti positivo per applicare la formula [(cod(b ij) – 32) - (cod(blocKey j) – 32)] + 32, in caso contrari viene applicata la formula {[(cod(b ij) – 32) - (cod(blocKey j) – 32)] + 96} + 32, alla quale viene sommata 96, per motivazioni analoghe legate al modulo. Il risultato viene poi memorizzato e si passa ai successivi caratteri. Quando sul intera stringa è stato applicato l'algoritmo a blocchi viene ripristinato lo stack e si riprende la procedura di cifratura o decifratura a seconda della situazione.

Algoritmo delle Occorenze
Nel caso che la lettera estratta dalla stringa mycypher sia 'C' viene eseguito l'algoritmo delle occorrenze. La prima azione che viene eseguita in questo algoritmo a differenza di quelli decritti in precedenza consiste nel controllare se deve essere eseguita la cifratura o la decifratura.
Per la cifratura ci occupiamo di determinare la lunghezza della stringa myplaintext/cyphertext. Dopo di che andiamo a iterare sulla medesima stringa, caricando il carattere di cui vogliamo trovare le occorrenze. Per evitare che lo stesso carattere abbia molteplici cifrature durante l'algoritmo verranno rimosse le occorrenze dopo essere state cifrate. Questo ci spinge ad adottare una variabile contente la lunghezza della stringa per determinare la sua fine. Dunque una volta estratto il carattere si itera nuovamente sulla stringa cercando le sue occorrenze. Quando viene trovato un risconto, viene eseguita una procedura che si occupa di effettuare i calcoli per la cifratura. In primis vengono salvate nello stack alcune variabili utilizzate nelle iterazioni precedenti. Seguono tre controlli, per distinguere in quale situazione si trova la procedura e come si deve comportare per la memorizzazione dei caratteri e delle posizioni. Infatti tutti i risultati della cifratura vengono caricati in una array temporaneo sul quale si devono seguire le seguenti convenzioni, memorizzare

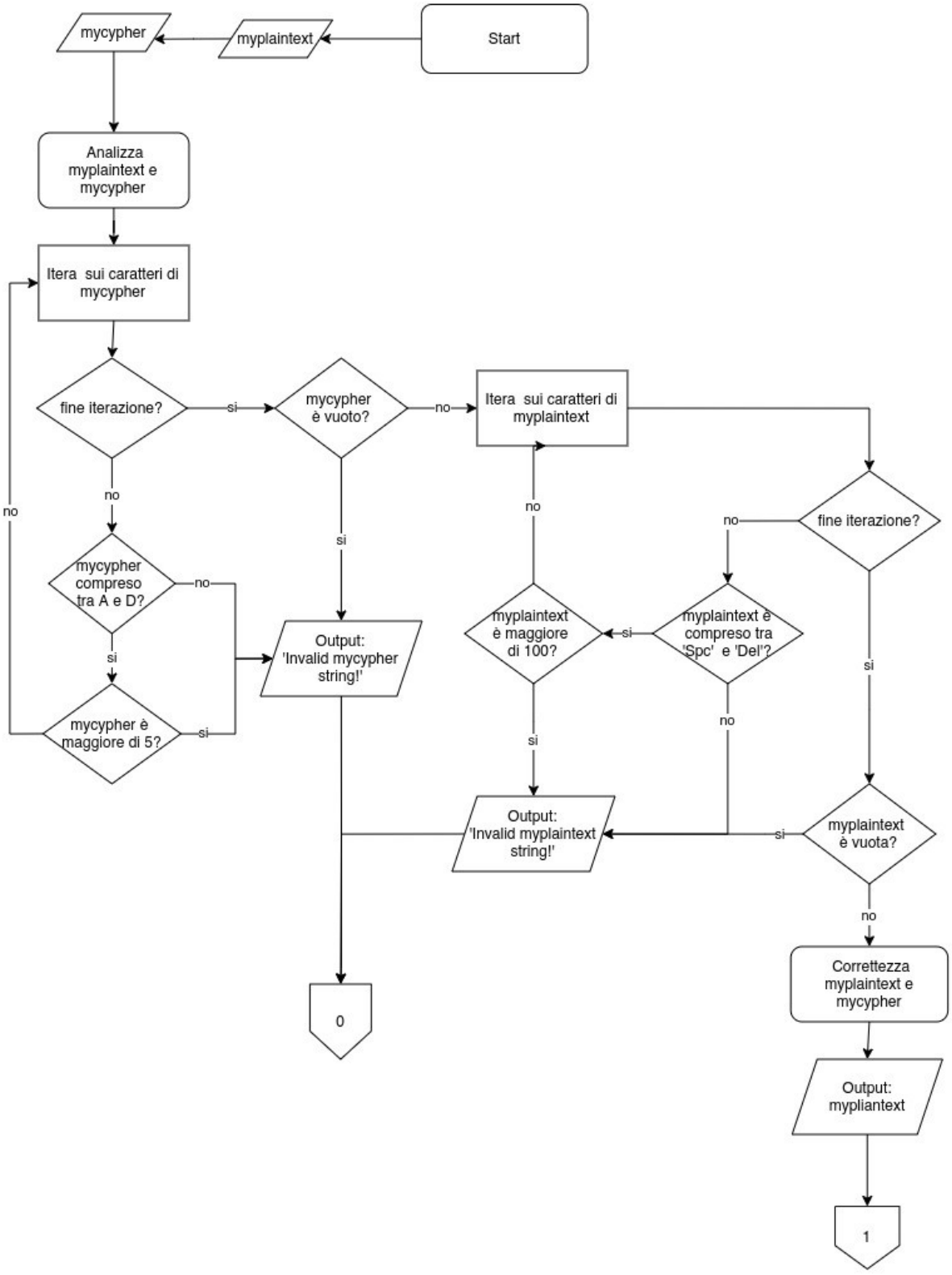
prima il carattere e successivamente al carattere le posizioni delle sue occorrenze divise dalla ‘-’ e ogni volta che si sono trovate tutte le occorrenze si deve passare al carattere successivo separandoli da uno spazio ‘a-1 b-2’. Dunque seguendo queste convezioni la procedure controlla se è la prima esecuzione per memorizza il carattere all’inizio della stringa, dato che nelle cifrature dei caratteri successivi al primo deve essere preceduto da uno spazio che li separi. Se il carattere è stato già memorizzato questa parte viene salta e si passa alla conversione della posizione in un carattere ASCII. Per fare ciò viene preso l’indice del iterazione che corrisponde alla posizione esatta dell’occorrenza, aumentata di uno poiché i caratteri vengo memorizzati a partire da zero negli array. La conversione della posizione deve essere fatta per ogni cifra del numero. Nel caso di valori a una sola cifra quindi minori di dieci si sottrae 48 e poi si memorizza, mentre nel caso di più cifre dunque tutti i numeri maggiori di nove utilizziamo un array per caricare il numero scomposto in singole cifre. Il numero di partenza viene scomposto tramite il resto di dieci per restituire la cifra più a destra, che viene caricata sul array, poi il numero viene diviso sempre per dieci e si ripetono i passaggi precedenti fino a quando la divisione non da come risultato zero, che corrisponde alla completa divisione del numero nelle sue singole cifre. A questo punto ogni cifra a partire ad quella più a sinistra viene estratta dal array e convertita in ASCII tramite la somma di 48 per essere memorizzata su un array temporaneo, questo procedimento viene ripetuto per tutte le cifre. Una volta terminato viene rimosso il carattere dell’occorrenza dalla stringa di partenza e viene ripristinato lo stack. Il controllo ritorna al ciclo che si occupa di trovare la successiva occorrenze. La ricerca termina quando no ci sono più corrispondenze nella stringa riprendendo il primo ciclo per identificare un nuovo carattere del quale trovare le occorrenze. Una volta che anche tale stringa che stata completamente iterata, si procede al trasferimento inverso dei dati dall’array temporaneo alla stringa di partenza, per evitare problemi durante l’applicazione dello stesso algoritmo più di una volta. Viene anche ripristino lo stack.
Per la decifratura viene iterato il cyphertext dal quale viene caricato il carattere da decifrare, di seguito viene eliminato dalla stringa insieme al carattere separatore ‘-’. Successivamente viene caricata il codice ASCII contenente la cifra che compone la posizione del carattere (es. ...c-→ 4..-, →cod(52)), essa viene convertita in un numero decimale sottraendo 48. La prima cifra della posizione viene memorizzata in un buffer(su una word). La successiva cifra viene convertita per essere poi sommata al prima cifra moltiplicata per dieci, il risultato viene nuovamente caricato sul intero buffer e la cifra cancellata dalla stringa di partenza. Questo procedimento viene effettuato per ricomporre il valore della posizione dell’occorrenza. Tale iterazione viene sospesa se il carattere estratto è un separatore ‘-’ segnalando che la posizione dell’occorrenza è stata completamente decodificata, portando al salvataggio del carattere in un array temporaneo nella posizione decodificata. Viene ripresa l’iterazione per la decodifica delle restanti occorrenze per lo stesso carattere. Quando durante l’iterazione invece viene caricato uno spazio segnala la completa decifratura del carattere in esame e quindi si ritorna al ciclo precedente per passare al successivo carattere sul quale applicare l’algoritmo. Una volta che anche tale stringa che stata completamente iterata, si procede con la stessa operazione effettuata durante la cifratura.

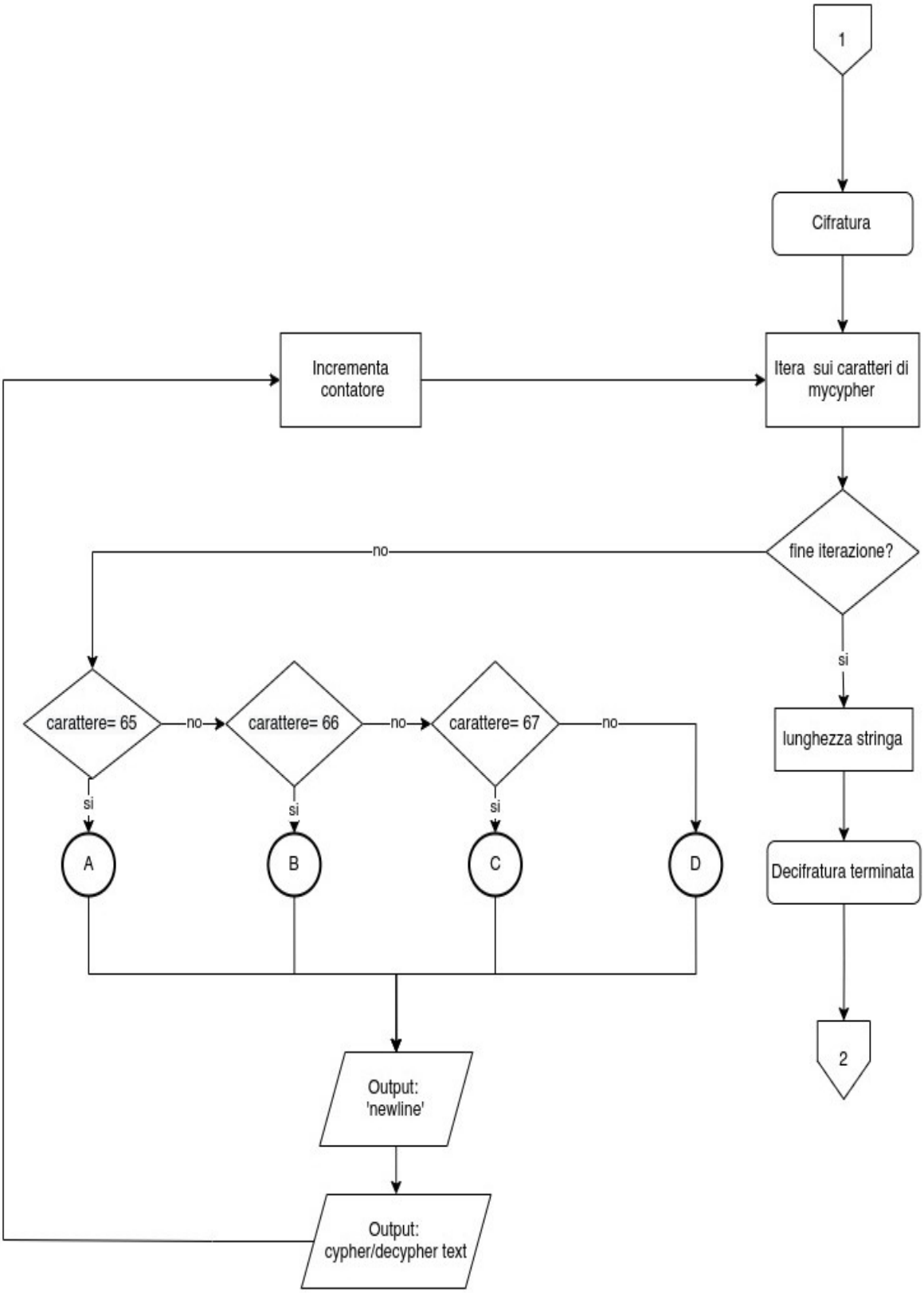
Dizionario
Nel caso che la lettera estratta dalla stringa mycypher sia ‘D’ viene eseguito l’algoritmo dizionario. Tale algoritmo si occupa di cifrare e decifrare utilizzando le stesse procedure. La stringa viene quindi iterata per estrarre il carattere. Viene chiama la procedure utilizzata anche nell’algoritmo di cesare. Tale procedura nel caso non ritorni una lettera controlla se il carattere è un numero, se il controllo risulta positivo viene sottratto 48 al codice ASCII di tale carattere e conseguentemente viene applicata la formula cod(57)-numero, in caso contrari il carattere viene memorizzato senza alcuna alterazione. Nel caso in cui sia una lettera può ritornare il valore di offset 97 se minuscola e quindi viene applicata la formula 90 - ASCII(lettera) + offset oppure 65 se maiuscola e viene applicata la formula 122 - ASCII(letter) + offset. Viene quindi memorizzato il risultato e ripresa l’iterazione, che termina con la fine della stringa. Lo stack viene ripristinato e si ritorna alla procedura di cifratura o decifratura.

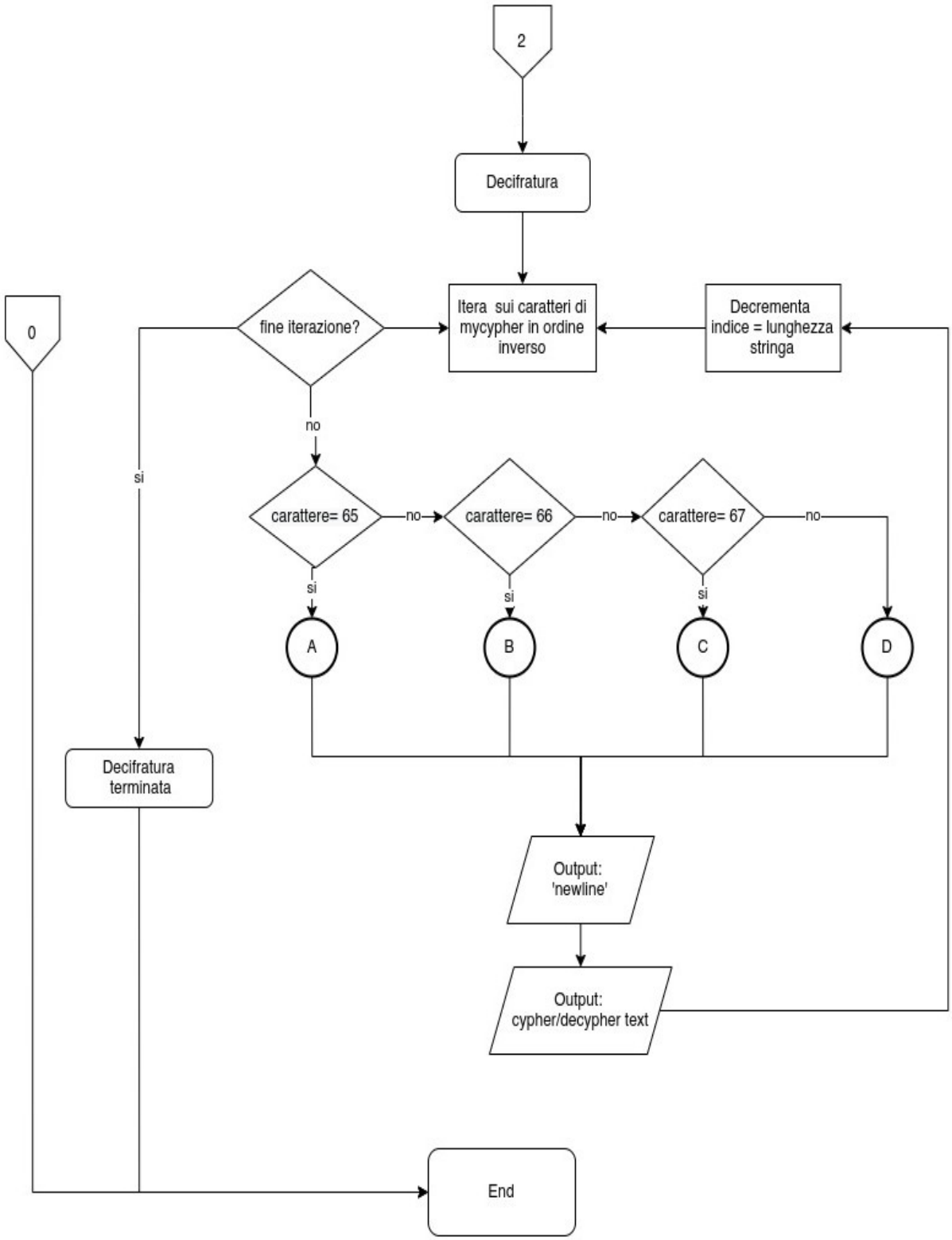
GetCharOffset:
Procedura che si occupa di calcolare l’offset delle lettere minuscole e maiuscole, nel caso di caratteri che non sono lettere viene ritornata un flag. La procedura come primo passo carica i valori nello stack, poi lancia un ulteriore procedura che si preoccupa di controllare che sia una lettera minuscola tramite l’utilizzo dei valori ASCII, nel caso affermativo viene ritornato il valore 1, in caso contrari controlla che sia una lettera maiuscola e restituisce 0 oppure ripristina lo stack e restituisce -1. Nei casi in cui sia una lettera torna al chiamante per ripristinare lo stack ed assegnare il valore 97 (ASCII 'a') in caso di lettera

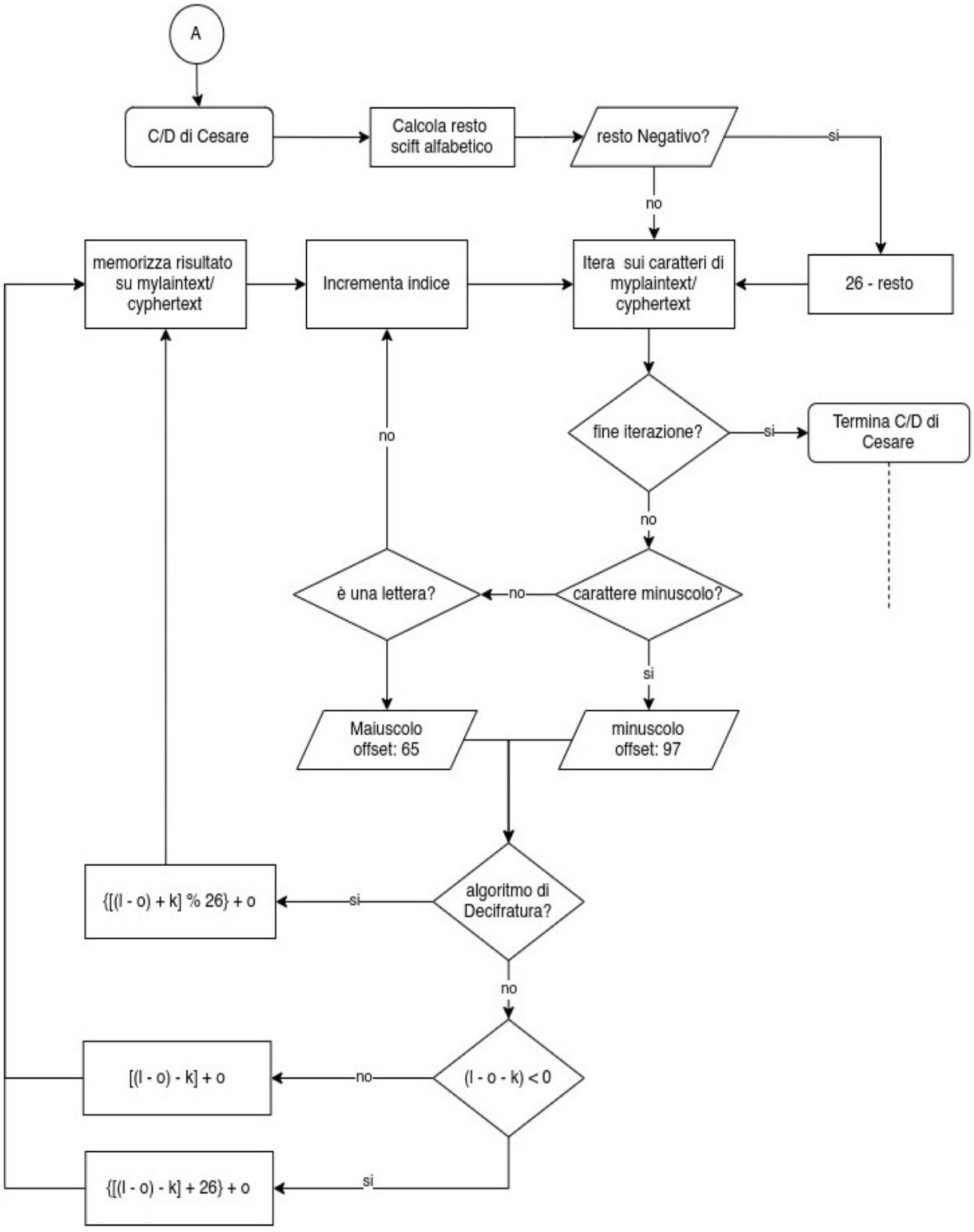
minuscola oppure 65 (ASCII 'A') in caso di lettera maiuscola. E infine ritornare all'algoritmo con il valore risultate da tali operazioni.

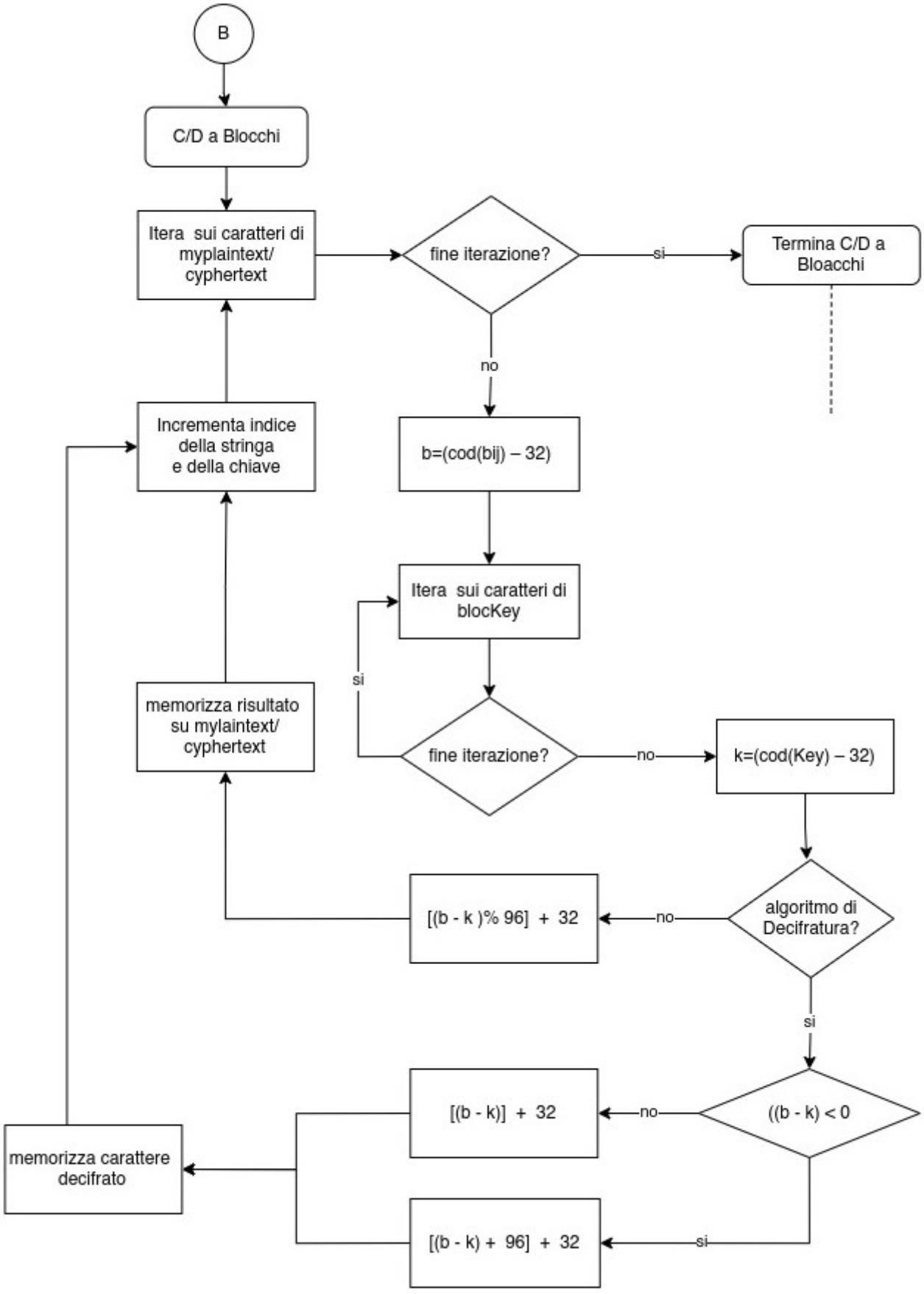
Flow-chart

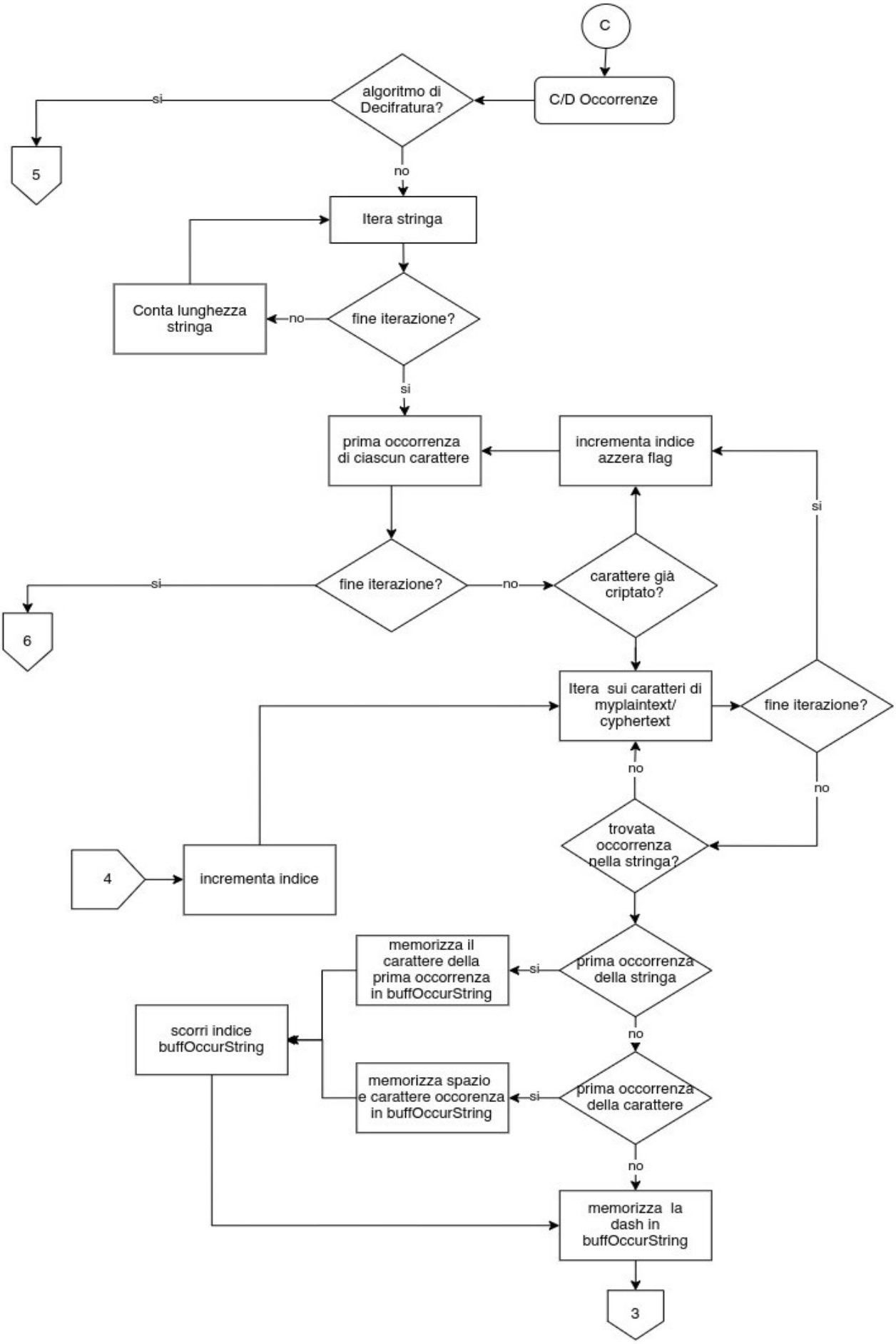


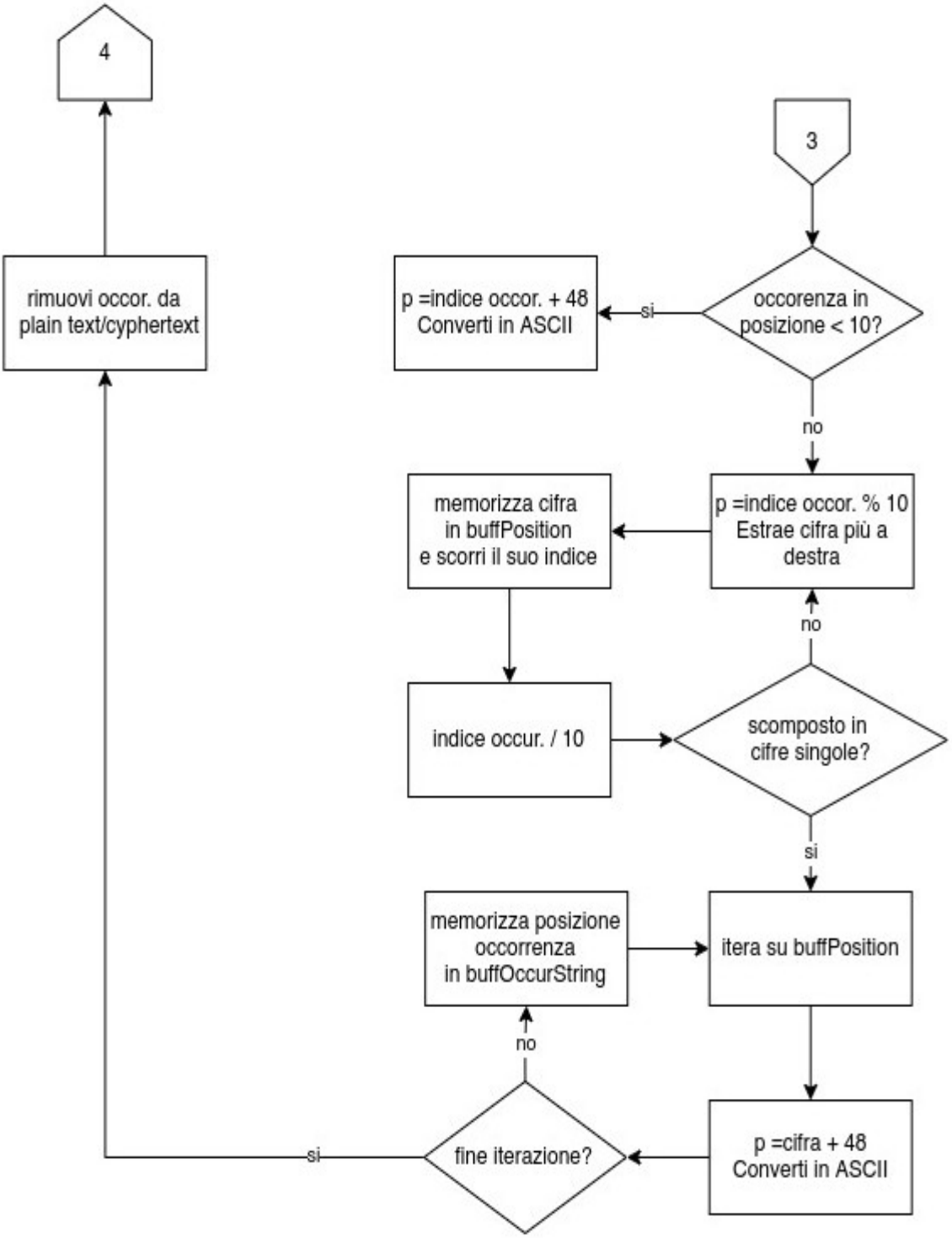

1
Cifratura
Itera sui caratteri di mycypher
Incrementa contatore
fine iterazione?
no
si
carattere= 65
no
carattere= 66
no
carattere= 67
no
si
si
si
A
B
C
D
lunghezza stringa
Decifratura terminata
2
Output: 'newline'
Output: cypher/decypher text

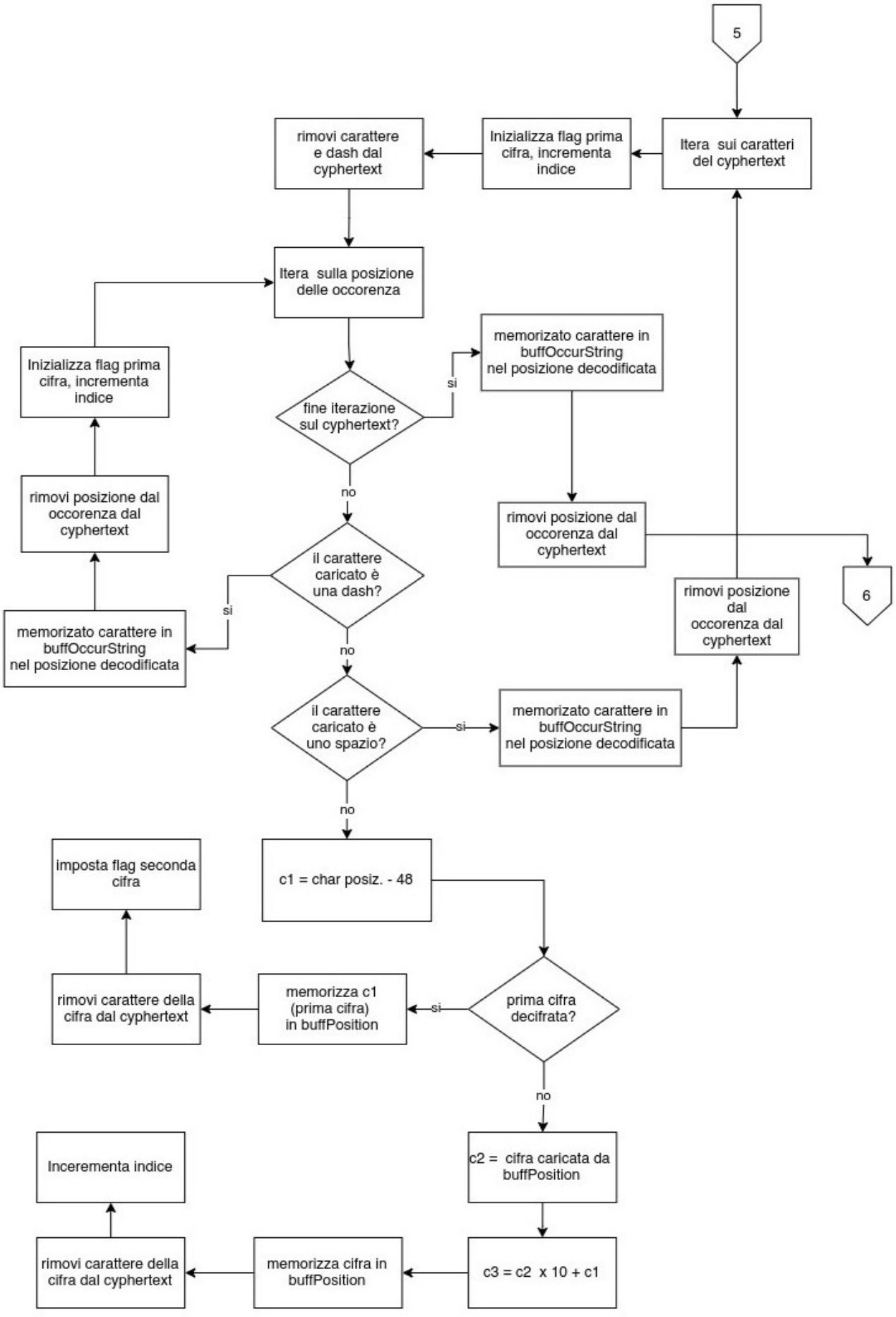

2
Decifratura
0
fine iterazione?
Itera sui caratteri di mycypher in ordine inverso
Decrementa indice = lunghezza stringa
no
si
carattere= 65
no
carattere= 66
no
carattere= 67
no
si
si
si
A
B
C
D
Decifratura terminata
Output: 'newline'
Output: cypher/decypher text
End

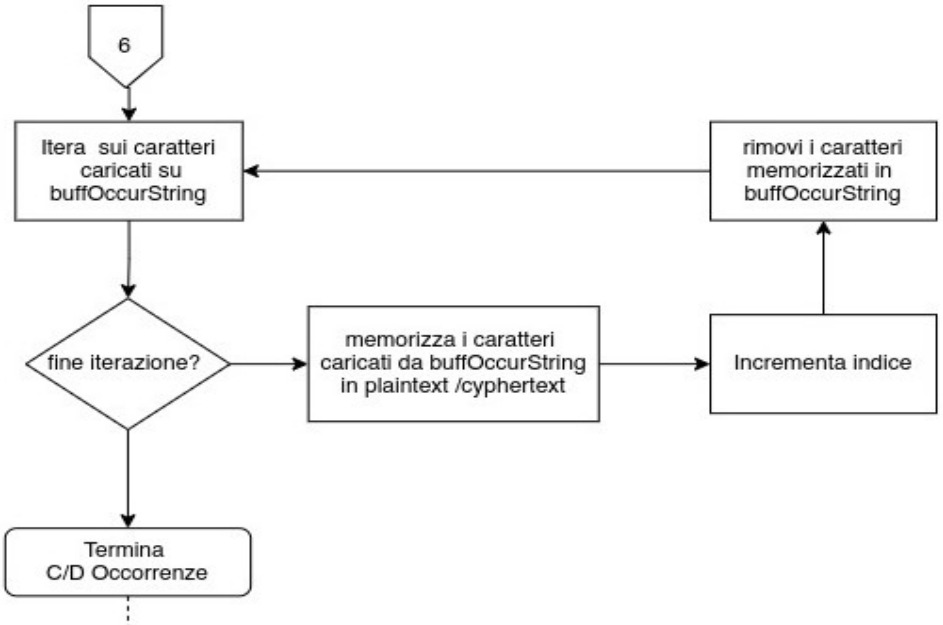

A
C/D di Cesare
Calcola resto scift alfabetico
resto Negativo?
si
no
memorizza risultato su mylaintext/ cyphertext
Incrementa indice
Itera sui caratteri di myplaintext/ cyphertext
26 - resto
fine iterazione?
si
Termina C/D di Cesare
no
no
è una lettera?
no
carattere minuscolo?
si
Maiuscolo offset: 65
minuscolo offset: 97
{[(l - o) + k] % 26} + o
si
algoritmo di Decifratura?
no
[(l - o) - k] + o
no
(l - o - k) < 0
{[(l - o) - k] + 26} + o
si

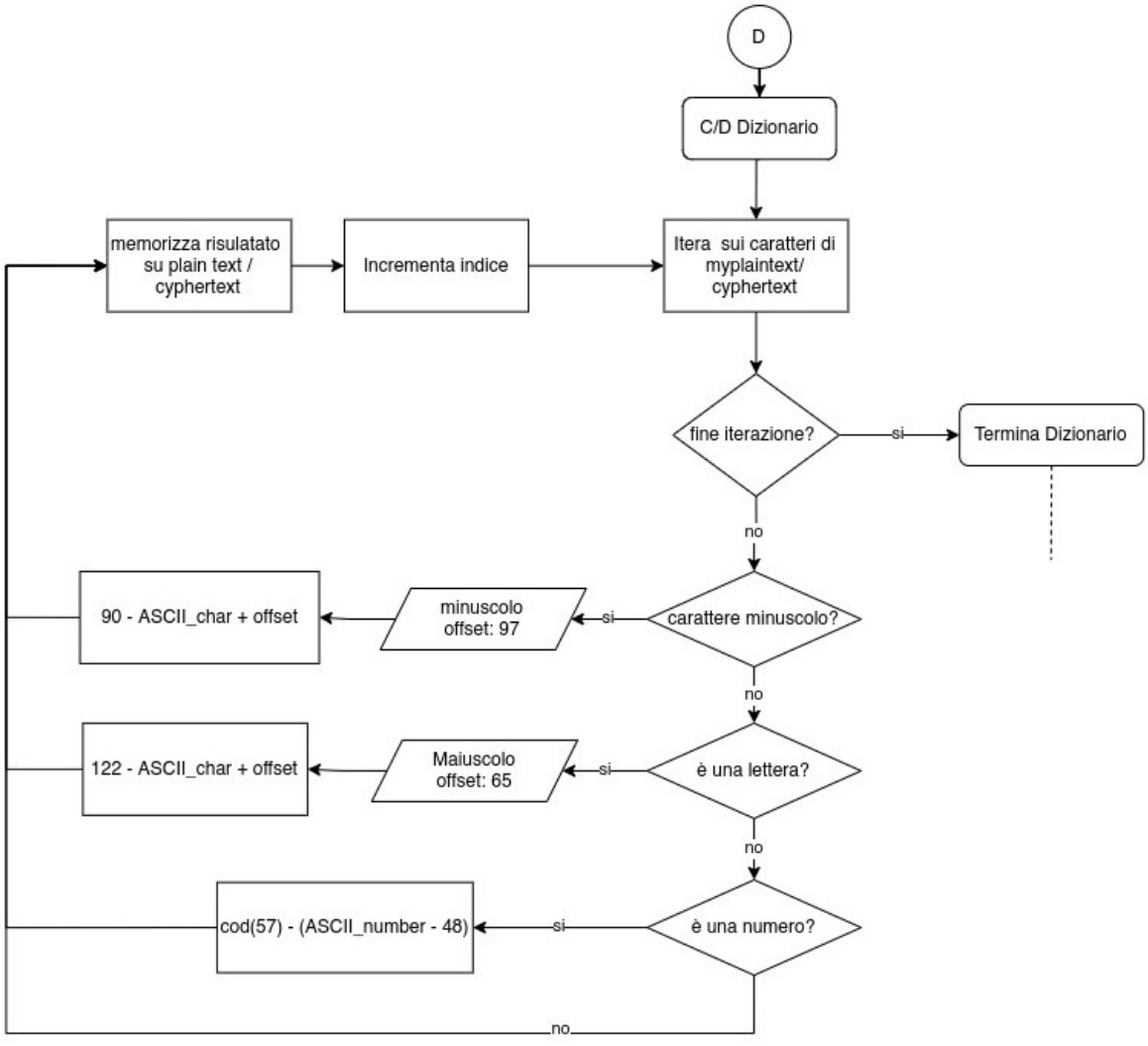

B
C/D a Blocchi
Itera sui caratteri di myplaintext/ cyphertext
fine iterazione?
si
Termina C/D a Bloacchi
no
Incrementa indice della stringa e della chiave
b=(cod(bij) – 32)
Itera sui caratteri di blocKey
si
memorizza risultato su mylaintext/ cyphertext
fine iterazione?
no
k=(cod(Key) – 32)
[(b - k)% 96] + 32
no
algoritmo di Decifratura?
si
[(b - k)] + 32
no
((b - k) < 0
memorizza carattere decifrato
[(b - k) + 96] + 32
si


C
C/D Occorrenze
algoritmo di Decifratura?
si
5
no
Itera stringa
Conta lunghezza stringa
no
fine iterazione?
si
prima occorrenza di ciascun carattere
incrementa indice azzera flag
si
fine iterazione?
si
6
no
carattere già criptato?
Itera sui caratteri di myplaintext/ cyphertext
fine iterazione?
no
no
trovata occorrenza nella stringa?
4
incrementa indice
memorizza il carattere della prima occorrenza in buffOccurString
si
prima occorrenza della stringa
scorri indice buffOccurString
no
memorizza spazio e carattere occorenza in buffOccurString
si
prima occorrenza della carattere
no
memorizza la dash in buffOccurString
3


4
3
rimuovi occor. da
plain text/cyphertext
p =indice occor. + 48
Converti in ASCII
si
occorenza in
posizione < 10?
no
memorizza cifra
in buffPosition
e scorri il suo indice
p =indice occor. % 10
Estrae cifra più a
destra
no
indice occur. / 10
scomposto in
cifre singole?
si
memorizza posizione
occorrenza
in buffOccurString
itera su buffPosition
no
fine iterazione?
si
p =cifra + 48
Converti in ASCII


5
Itera sui caratteri del cyphertext
Inizializza flag prima cifra, incrementa indice
rimovi carattere e dash dal cyphertext
Itera sulla posizione delle occorenza
fine iterazione sul cyphertext?
si
memorizato carattere in buffOccurString nel posizione decodificata
rimovi posizione dal occorenza dal cyphertext
no
il carattere caricato è una dash?
si
memorizato carattere in buffOccurString nel posizione decodificata
rimovi posizione dal occorenza dal cyphertext
Inizializza flag prima cifra, incrementa indice
no
il carattere caricato è uno spazio?
si
memorizato carattere in buffOccurString nel posizione decodificata
rimovi posizione dal occorenza dal cyphertext
6
no
c1 = char posiz. - 48
prima cifra decifrata?
si
memorizza c1 (prima cifra) in buffPosition
rimovi carattere della cifra dal cyphertext
imposta flag seconda cifra
no
c2 = cifra caricata da buffPosition
c3 = c2 x 10 + c1
memorizza cifra in buffPosition
rimovi carattere della cifra dal cyphertext
Inceremenda indice


6
Itera sui caratteri caricati su buffOccurString
rimovi i caratteri memorizzati in buffOccurString
fine iterazione?
memorizza i caratteri caricati da buffOccurString in plaintext /cyphertext
Incrementa indice
Termina C/D Occorrenze


D
C/D Dizionario
memorizza risulatato su plain text / cyphertext
Incrementa indice
Itera sui caratteri di myplaintext/ cyphertext
fine iterazione?
si
Termina Dizionario
no
90 - ASCII_char + offset
minuscolo offset: 97
si
carattere minuscolo?
no
122 - ASCII_char + offset
Maiuscolo offset: 65
si
è una lettera?
no
cod(57) - (ASCII_number - 48)
si
è una numero?
no

Uso dei registri e memoria

In .data vengono caricate nella memoria statica "Invalid myplaintext string!" e "Invalid mycypher string!" che rappresentano due stringhe utilizzare come output nel caso in cui si riscontrino errori nel controllo della stringa myplaintext e mycypher. Viene poi definito l'indirizzo alla testa del array che verrà creato nel algoritmo delle occorrenze, contestualmente ad un buffer sul quale verranno caricati valori intermedi per lo stesso algoritmo (memoria dinamica). Seguo le variabili per lo spostamento alfabetico e quella la chiave per l'algoritmo a blocchi. Per finire nella memoria statica viene caricata la stringa mycpyher usata per identificare gli algoritmi da applicare al plain text. Nella memoria dinamica viene invece caricato la stinga myplaintext che rappresenta il messaggio da cifrare e decifrare.

main:
s0 = myplaintext s1 = mycypher
a2 ← s0 e a3 ← s1 per invocare la procedura plaiCypCheck per il controllo delle stringhe
a0 ← s0 per essere usato da tutte le successive procedure che si occupano di andare a modificare la stringa myplaintext per l'applicazione degli algoritmi di cifratura e decifratura.

plaiCypCheck:
t0 ← contatore ciclo usato in cpyCheckLoop
t4 ← contatore ciclo usato in strCheckLoop
t5 ← 100 limite di caratteri consentiti sulla stringa myplaintext, usato in cpyCheckLoop
t6 ← 5 limite di caratteri consentiti sulla stringa mycypher, usato in strCheckLoop

cpyCheckLoop:
a4 ← 65 codice ASCII del carattere 'A', usato per controlli
a5 ← 68 codice ASCII del carattere 'D', usato per controlli
a3 ← stringa mycypher
t0 ← contatore per scorrere sulla stringa mycypher
t1 ← carattere corrente caricato dalla stringa mycypher
t3 ← indirizzo di base che punta alla stringa mycypher sommato al contatore
t6 ← 5, limite sulla grandezza della stringa

strCheckLoop:
a4 ← 32 codice ASCII del carattere 'Spc', usato per controlli
a5 ← 127 codice ASCII del carattere 'Del' usato per controlli
a2 ← stringa myplaintext
t4 ← contatore per scorrere sulla stringa myplaintext
t2 ← indirizzo di base che punta alla stringa myplaintext sommato al contatore
t1 ← carattere corrente caricato dalla stringa myplaintext
t0 ← valore contatore usato in cpyCheckLoop, usato per controllare che la stringa mycypher, non sia vuota

strCheckExit:
t4 ← valore contatore usato in strCheckLoop, usato per controllare che la stringa myplaintext, non sia vuota
t5 ← 100, limite sulla grandezza della stringa
ra ← ritorna al main, all'istruzione successiva 'jal print_plaintext'

cryCheckError:
a0 ← chiamate di sistema per stamapare 'Invalid mycypher string!'
a7 ← 4 valore per la stampa di stringhe

strCheckError:
a0 ← chiamate di sistema per stamapare
a7 ← 4 valore per la stampa di stringhe

cipher:
Memorizza nello stack il registro ra per ritornare al main
t0 ← inizializza il contatore ciclo usato in loopCipher

loopCipher:
a3 ← stringa mycypher
t2 ← indirizzo di base che punta alla stringa mycypher sommato al contatore
t0 ← contatore ciclo
t1 ← carattere corrente caricato dalla stringa mycypher
updateLoopCipher:
t0 ← viene incrementato il contatore usato in loopCipher

endLoopCipher:
Ripristina lo stack contente l'istruzione 'ra' per tornare al main.

a1 ← rappresenta il valore della lunghezza della stringa mycypher
ra ← ritorna al main, all'istruzione successiva 'jal decipher'

decipher:
Memorizza nello stack il registro ra per ritornare al main.
t0 ← contatore del ciclo inizializzato alla lunghezza di mycypher per essere decementato e quindi iterare sulla stringa in ordine inverso
a1 ← lunghezza della stringa mycypher
a5 ← -1, flag usato per identificare che l'algoritmo che verrà eseguito è in fase di decifratura

loopDecipher:
a3 ← stringa mycypher
t0 ← contatore ciclo
t1 ← carattere corrente caricato dalla stringa mycypher
t2 ← indirizzo di base che punta alla stringa mycypher sommato al contatore

updateLoopDecipher:
t0 ← viene decrementato il contatore usato in *loopDecipher*

endLoopDecipher:
Ripristina lo stack contente l'istruzione 'ra' per tornare al main.
ra ← ritorna al main, all'istruzione successiva *'j endMain', la quale salta alla fine del programma e lo termina*

algChoice:
t3 ← 65 codice ASCII del carattere 'A' usato per controllare che quale algoritmo eseguire
t4 ← 66 codice ASCII del carattere 'B' usato per controllare che quale algoritmo eseguire
t5 ← 67 codice ASCII del carattere 'C' usato per controllare che quale algoritmo eseguire

caesarCipherDecipher:
Memorizza nello stack il registro ra e t0 sul quale è memorizzato il contatore per attraversare la stringa mycypher.
Durante la cifratura l'istruzione 'ra' ritorna a loopCipher, successivamente nella decifratura l'istruzione 'ra' ritrona a loopDecipher.

t1 ← inizzializza contatore ciclo usato in coreCaesarAlgorithm
t3 ← -1, flag per controllare se siamo nella decifratura
t6 ← 26 costante per il modulo dello shift alfabetico
a4 ← viene caricato il valore dello shift alffabetico

moduleSostK:
a4 ← viene memorizzato il risultato del modulo di 26 sullo shift

negModuleSostK:
a4 ← viene memorizzato il risultato del resto di 26 sullo shift nel caso in cui lo sia negativo

coreCaesarAlgorithm:
a0 ← stringa myplaintext
t1 ← contatore ciclo
t2 ← indirizzo di base che punta alla stringa *myplaintext* sommato al contatore
a6 ← *carattere corrente caricato dalla stringa myplaintext*
t5 ← prende il valore del carattere corrente
t3 ← -1, flag per controllare che non sia una lettera
a1 ← risultato della procedura getCharOffset
a5 ← -1/0, flag per controllare se siamo in decifrare

cipherCaesarAlgorithm:
t0 ← viene salvato il risultati dei calcoli per l'algoritmo di cesare nella cifratura
t5 ← codice ASCII della lettera da cifrare
t4 ← offset passato da a0
a4 ← shift alfabetico sul quale sono stati applicati dei calcoli
t6 ← 26, per calcolare il modulo

decipherCaesarAlgorithm:
t0 ← viene salvato il risultati dei calcoli per l'algoritmo di cesare della decifratura
t5 ← codice ASCII della lettera da cifrare
t4 ← offset passato da a0
a4 ← shift alfabetico sul quale sono stati applicati dei calcoli

negSostKDecipCaesarAlgorithm:
t0 ← viene salvato il risultati dei calcoli per l'algoritmo di cesare della decifratura, nel caso (letter ASCII - offset - key) < 0
a4 ← shift alfabetico sul quale sono stati applicati

dei calcoli
t6 ← 26

caesarNextChar:
t1 ← contatore ciclo incrementato, usato in coreCaesarAlgorithm

caesarCipDecipEnd:
Ripsristina i valori memorizzati sullo stack ‘ra’ e ‘t0’.
ra ← ritorna a loopCipher nel caso di cifratura oppure a loopDecipher durante la decifratura

blockCipherDecipher:
Viene memorizzato nello stack t0, contatore di loopCipher e loopDecipher a seconda della situazione. Non viene caricato nessun ‘ra’ poiché non ci sono istruzioni jal.
a1 ← carica la stringa contente la chiave per l’algoritmo a blocchi(blocKey)
a6 ← -1, flag per controllare se siamo nella decifratura
t0 ← contatore ciclo per la stringa myplaintext
t1 ← contatore ciclo per la stringa chiave (blocKey)
t6 ← 96, valore per il calcolo del modulo usato nella formula dell’algoritmo

coreBlockAlgorithm:
a0 ← stringa myplaintext
t1 ← contatore ciclo per iterare su myplaintext
t2 ← indirizzo di base che punta alla stringa myplaintext sommato al contatore
t4 ← carattere corrente caricato dalla stringa myplaintext, sul quale viene poi salvato il risultato di cod(char) - 32
a1 ← stringa della chiave (blocKey)
t0 ← contatore ciclo per iterare su blocKey
t3 ← indirizzo di base che punta alla stringa blocKey sommato al contatore
t5 ← carattere corrente caricato dalla stringa blocKey

coreBlockKeyAlgorithm:
t5 ← viene poi salvato il risultato di cod(key) – 32
a5 ← -1/ 0, flag per controllare se siamo in decifrare
a6 ← -1, flag per controllare se siamo in decifrare

cipherBlockAlgorithm:
t4 ← salvato il risultato della formula per la cifratura a blocchi e precedentemente risultato di (cod(char) - 32)
t5 ← risultato di (cod(key) – 32)

t6 ← 96, valore per calcolare il modulo durante la formula

cipherBlockAlgorithm:
t4 ← salvato il risultato della formula per la cifratura a blocchi e precedentemente risultato di (cod(char) - 32)
t5 ← risultato di (cod(key) – 32)

negModuloDecipBlockAlgorithm:
t4 ← salvato il risultato della formula per la cifratura a blocchi e precedentemente risultato di (cod(char) - 32)

blockNextChar:
t0 ← contatore ciclo per la stringa blocKey, incrementato e usato in coreBlockAlgorithm
t1 ← contatore ciclo per la stringa myplaintext, incrementato e usato in coreBlockAlgorithm

blocKeyLoop:
t0 ← 0, resetta il contatore per la stringa blocKey
a1 ← stringa della chiave (blocKey)
t3 ← indirizzo di base che punta alla stringa blocKey sommato al contatore
t5 ← carattere corrente caricato dalla stringa blocKey

blockCipDecEnd:
Ripristina il valore memorizzato nello stack, t0 contatore ciclo per la cifratura e la decifrare.
ra ← ritorna a loopCipher nel caso di cifratura oppure a loopDecipher durante la decifratura

encrDecrOccurrence:
t0 ← -1, flag per controllare se siamo nella decifratura
t5 ← inizializza *contatore ciclo usato in charPositionOccur*
t6 ← inizializza contatore ciclo usato in charEncodeOccur
a4 ← 45, codice ASCII per il carattere ‘-’
a6 ← 32, codice ASCII per il carattere ‘ ’, spazio
a1 ← 2000, l’indirizzo per l’array temporaneo bufferOccurString (situato in .text)
a3 ← inizializza il buffer, buffPosition, per immagazzinare valori intermedi durante la cifratura e la decifratura
a5 ← -1/0, *flag per controllare se siamo in decifrare*

couterStringOccur:
a0 ← stringa myplaintext
t0 ← contatore lunghezza della stringa myplaintext

t2 ← indirizzo di base che punta alla stringa myplaintext sommato al contatore
t4 ← carattere corrente caricato dalla stringa myplaintext

encrOccur:
t4 ← inizializza contatore ciclo usato in occurSeparDash
a7 ← 1, flag per identifica la prima occorrenza del carattere da cifrare, eccetto il primo carattere della stringa myplaintext

charEncodeOccur:
a0 ← stringa myplaintext
a2 ← carattere corrente caricato dalla stringa, di cui deve essere trovate le occorrenze nella medesima stringa
t3 ← indirizzo di base che punta alla stringa myplaintext sommato al contatore
t6 ← contatore ciclo per iterare su myplaintext
t0 ← valore lunghezza stringa

charPositionOccur:
a0 ← stringa myplaintext
a5 ← carattere corrente caricato dalla stringa, per essere confronto con a2 in cerca di occorenze
t2 ← indirizzo di base che punta alla stringa myplaintext sommato al contatore
t0 ← valore lunghezza stringa
a2 ← carattere di cui trovare le occorrenze all'interno della stringa

charPositionOccurLoop:
t5 ← contatore ciclo incrementato, usato in charPositionOccur

nextcharEncodeOccur:
a7 ← 0, impostato quando sono state trovate tutte le occorrenze di un carattere
t5 ← resetta il contatore del ciclo per charPositionOccur e successivamente viene inizializzato a t6 per saltare tutti i valori che a già analizzato nel cicli precedenti
t6 ← incrementa il contatore del ciclo per charEncodeOccur
coreOccurAlgor:
Viene memorizzato nello stack t0, che contiene la lunghezza della stringa myplaintext, t5 che contiene il contatore usato in charPositionOccur e t6 contente il contatore usato in charEncodeOccur.

t2 ← 10, valore usato per il calcolo del modulo e nella divisione
t3 ← 3, valore indice per iterare su buffPosition (a3)
t5 ← contatore del ciclo charPositionOccur
a7 ← 0/1, valore per controllare se è la prima occorrenza del carattere oppure le successive

occurSeparDash:
a1 ← stringa buffOccurString
a4 ← 45, carattere ASCII '-'
t4 ← contatore ciclo
t6 ← indirizzo di base che punta alla stringa buffOccurString sommato al contatore
t5 ← posizione dell'occorrenza
t2 ← 10, valore per il controllo di posizioni a singola cifra

positionEncodInASCII:
t0 ← caricato risultato del calcolo
t5 ← posizione dell'occorrenza
t2 ← 10, valore per il calcolo del modulo e la divisione
t1 ← indirizzo di base che punta all'array buffPosition sommato al contatore
a3 ← array buffPosition
t3 ← contatore per iterare su buffPosition

charPositionEncode:
t3 ← contatore ciclo per buffPosition incrementato
a3 ← array buffPosition
t1 ← indirizzo di base che punta alla array buffPosition sommato al contatore
t0 ← carica il valore memorizzato su a3, su esso vengono effettuate operazioni
a1 ← stringa buffOccurString
t4 ← contatore ciclo per la stringa buffOccurString, successivamente viene incremento
t6 ← indirizzo di base alla stringa buffOccurString

singleDigitPositEncInASCII:
t0 ← memorizza risulto operazioni
t5 ← posizione occorrenza
a1 ← stringa buffOccurString
t4 ← contatore ciclo per la stringa buffOccurString, successivamente viene incremento
t6 ← indirizzo di base alla stringa buffOccurString

firstCharEncode:
a1 ← stringa buffOccurString

t4 ← contatore ciclo per la stringa buffOccurString, successivamente viene incremento
t6 ← indirizzo di base alla stringa buffOccurString
a5 ← carattere dell'occorrenza ricercata, estratto durante charPositionOccur

followSpaceCharEncode:
a1 ← stringa buffOccurString
t4 ← contatore ciclo per la stringa buffOccurString, successivamente viene incremento
t6 ← indirizzo di base alla stringa buffOccurString
a6 ← 32, valore ASCII corrispondente allo spazio ' '
a5 ← carattere dell'occorrenza ricercata, estratto durante charPositionOccur
a7 ← 1, cambia il flag per identificare è il secondo ciclo dell'occorrenza e quindi non è necessario memorizzare il carattere ricercato

nextCharPositionOccur:
Carica lo stack memorizzato in *coreOccurAlgor. Ripristina quindi t0 che contiene il valore della lunghezza della stringa myplaintext, t5 e t6 che contiene il contatore del ciclo usato per charPositionOccur e charEncodeOccur.*
a0 ← stringa myplaintext
t5 ← contatore ciclo per la stringa myplaintext e successivamente incementato
t2 ← indirizzo di base per la stessa stringa
zero ← usato per rimuove il valore del carattere all'interno della stringa myplaintext

decrOccur:
Memorizza nello stack il valore di a5 che rappresenta il flag per l'identificazione dell'algoritmo di decifrare.
t0 ← inizializza il contatore del ciclo di charDencodeOccur
a7 ← 10, usato per la moltiplicazione

charDencodeOccur:
a0 ← stringa myplaintext (anche identificato come cyphertext)
t0 ← contatore ciclo
t3 ← indirizzo di base per la stringa
t1 ← carattere dell'occorrenza per la decriptazione delle sue occorrenze
t5 ← flag per identificare la prima iterazione sulle cifre che compongono le posizioni delle occorrenze
t4 ← contatore ciclo usato sulla stringa myplaintext (cyphertext)
zero ← usata per rimuovere il corrente carattere e la dash della stringa corrente

decryptOccurAlgorithm:
a0 ← stringa myplaintext (cyphertext)
t4 ← contatore ciclo
t3 ← indirizzo di base per la stringa
t2 ← carattere contente la cifra della posizione di un occorrenza, una dash oppure uno spazio; su di essa vengono anche effettuati calcoli
a4 ← 45, codice ASCII per il carattere '-'
a6 ← 32, codice ASCII per il carattere spazio ' '
t5 ← flag per controllare se è la cifra più a sinistra del valore della posizione, passa da 0 a 1 dopo che è stata processata la cifra più a sinistra
a3 ← array buffPostion per memorizzare i valori intermedi della valore posizione
t6 ← indirizzo di base della arry buffPostion
zero ← usata per rimuovere il corrente carattere dalla stringa myplaintext

storePositionOccurChar:
a3 ← array buffPostion contente un valore
a5 ← viene caricato il valore contenuto in buffPosition, vengono poi eseguiti su di essa i successivi calcoli
a7 ← 10, valore per il calcolo della moltiplicazione per decifrare il valore della posizione
t2 ← carattere estratto dalla stringa myplaintext contente la successiva cifra del valore della posizione
a3 ← array buffPostion per memorizzare i valori intermedi della valore posizione
t6 ← indirizzo di base della arry buffPostion
zero ← usata per rimuovere il corrente carattere dalla stringa myplaintext

nextPositDecrOccurAlg:
t4 ← contatore incrementato del ciclo per la stringa myplaintext

nextCharDecOccurSpace:
t0 ← contatore incrementato del numero di iterazioni fatte per decifrare le posizioni delle occorrenze

nextPositOccurDash:
t5 ← reimposta il flag per identificare la cifra più a sinistra, poiché il ciclo passa alla prossima posizione dell'occorrenza da decifrare

t4 ← contatore incrementato del ciclo per la stringa myplaintext

storeDecrChar:
a3 ← array buffPosition
a5 ← valore decifrato della posizione dell'occorrenza del carattere, decrementato poi di uno; rappresenta l'indice per scorrere in buffOccurString
a1 ← stringa buffOccurString
a2 ← indirizzo di base per la stringa buffOccurString aggiunta della posizione dell'occorrenza del carattere da decifrare
t1 ← carattere corrente da decifrare
zero ← usata per rimuovere il corrente carattere dalla stringa myplaintext
ra ← per ritornare a nextCharDecOccurSpace, nextPositOccurDash oppure endDecrOccur, dipende di ha usato jal

endDecrOccur:
Ripristina lo stack restituendo da a5 il suo ruolo di flag

encrDecrOccurEnd:
t5 ← inizializza il contatore per il ciclo su buffOccurString e myplaintext

encrDecrOccurEndByValue:
a0 ← stringa buffOccurString con il risultato della cif./decif.
a0 ← stringa myplaintext sul quale viene trasferito il risultato della cif./decif.
t5 ← contatore per il ciclo su buffOccurString e myplaintext
t2 ← indirizzo di base per buffOccurString
t1 ← viene caricato il carattere per poi essere memorizzato sul myplaintext
t3 ← indirizzo di base per myplaintext

loopEndOccur:
t5 ← contatore incrementato per per il ciclo su buffOccurString e myplaintext
zero ← zero ← usata per rimuovere il corrente carattere dalla stringa buffOccurString

restoreStakOccurEnd:
Carica lo stack memorizzato encrDecrOccurrence. Carica t0 con il valore del contatore usato in loopCipher e loopDecipher, a3 sul quale è presente la stringa mycypher e infine ra.
ra ← ritorna a loopCipher nel caso di cifratura oppure a loopDecipher durante la decifratura

dictionary:
Viene memorizzato nello stack t0 ed ra.
t1 ← inizializza il contatore per il ciclo sulla stringa myplaintext
t3 ←-1, flag per controllare se il carattere è una non è una lettera
t4 ←97, carattere ASCII 'a' usata come flag per le lettere minuscole
t6 ← 57, carattere ASCII '9'
a7 ← 48, carattere ASCII '0'

coreDictionaryAlgorithm:
a0 ← stringa myplaintext
t1 ← contatore per il ciclo sulla stringa myplaintext
t2 ← indirizzo di base per la stringa
a6 ← carattere corrente su cui applicare l'algoritmo
t5 ← assume il valore del carattere corrente
zero ← utilizzato per controllo
a1 ← valore restituito da getCharOffset
t3 ←-1, flag per controllare se il carattere è una non è una lettera
t4 ←97, carattere ASCII 'a' usata come flag per le lettere minuscole

dictUpperCaseAlgorithm:
t5 ← contente il carattere su cui applicare l'algoritmo, rappresenta anche dove salvare i risultati dei calcoli
a1 ← 65, offset calcolato da getCharOffset; carattere ASCII 'A'

dictLowercaseAlgorithm:
t5 ← contente il carattere su cui applicare l'algoritmo, rappresenta anche dove salvare i risultati dei calcoli
a1 ← 97, offset calcolato da getCharOffset; carattere ASCII 'a'

dictionaryIsNotALetter:
a7 ← 48, carattere ASCII '0'; per controllare che il carattere in t5 sia un numero
t6 ← 57, carattere ASCII '9'; per controllare che il carattere in t5 sia un numero
t5 ← carattere estratto dalla stringa dove applicare l'algoritmo, se risulta un numero viene usato per salvare il risultato della formula

nextCharDictionary:
t5 ← carattere dopo l'applicazione della formula più appropriata
t2 ← indirizzo di base per la stringa myplaintext
t1 ← contatore incrementato per il ciclo sulla stringa myplaintext

endDictionary:
Carica i valori dallo stack di t0 e ra. t0 contiene il valore del contatore usato in loopCipher e loopDecipher.
ra ← ritorna a loopCipher nel caso di cifratura oppure a loopDecipher durante la decifratura

getCharOffset:
Viene memorizzato t1 e ra sullo stack. t1 contatore ciclo usato in coreCaesarAlgorithm e ra ritorna al coreDictionaryAlgorithm oppure coreCaesarAlgorithm. Dopo aver eseguito la procedura isLowerCase ripristina subito i valori dallo stack.
a1 ← flag ritornato dalla procedura, per controllare se è la lettera sia maiuscolo o minuscolo
zero ← combacia ad a1 nel caso la lettera sia maiuscolo

offsetCipherLowerCase:
a1 ← 97, carattere ASCII 'a', nel caso di lettera misucola
ra ← ritorna al chaimante

offsetCipherUpperCase
a1 ← 65, carattere ASCII 'A', nel caso di lettera misucola
ra ← ritorna al chaimante

isLowerCase:
t0 ← 97, carattere ASCII 'a'; per effettuare dei controlli
t1 ← 122, carattere ASCII 'z'; per effettuare dei controlli
a6 ← carattere corrente della stringa myplaintext (cyphertext)
a1 ← 1, valore assunto nel caso il carattere a6 sia una lettera minuscola

isNotLowerCase:
t0 ← 91, carattere ASCII '['; per effettuare dei controlli
t1 ← 65, carattere ASCII 'A'; per effettuare dei controlli
a6 ← carattere corrente della stringa myplaintext (cyphertext)
a1 ← 0, valore assunto nel caso il carattere a6 sia una lettera maiuscola

isNotALetter:
Ripristina lo stack memorizzato in *getCharOffset poiché non ritorna in tale procedura.*
a1 ← -1, valore assunto nel caso il carattere a6 sia non sia una lettera

print_plaintext:
a2 ← stringa contente il plain text
a0 ← puntatore alla stringa myplaintext
a7 ← 4 valore per la stampa di stringhe
ra ← ritorna la main, all'istruzione successiva 'add a0, s0, zero'

printCipherDecipher:
a1 ← viene spostata la stringa della cifratura e della decifratura da a0 perché viene prima usato per stampare uncarattere
a0 ← carica 10, carattere ASCII 'newline' stringa, successivamente viene puntata la stringa per la stampa d myplaintext(cyphertext)
a7 ← assumere valore 11 per la stampa di caratteri, poi assume valore 4 per la stampa di stringhe
ra ← ritorna a loopCipher nel caso di cifratura oppure a loopDecipher durante la decifratura

Test di corretto funzionamento

myplaintext

- dimensione massima 100 caratteri

- 32 ≤ cod(c) ≤ 127

Test 1

myplaintext:

‘ # $%&'()*+,-./0123456789:<=>?@abcdefghijklmnopqrstuvwxyz[\]^_ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~0123456!’

stringa di 100 caratteri

mycypher: ‘C’

```
# ============================================================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ============================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "C"                           # mycypher string where to indicate which encryption algorithms apply to myplaintext
myplaintext: .string " # $%&'()*+,-./0123456789:<=>?@abcdefghijklmnopqrstuvwxyz[\]^_ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~0123456!"    # r

```

```
 # $%&'()*+,-./0123456789:<=>?@abcdefghijklmnopqrstuvwxyz[\]^_ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~0123456!
 -1-3 #-2 $-4 %-5 &-6 '-7 (-8 )-9 *-10 +-11 ,-12 --13 .-14 /-15 0-16-93 1-17-94 2-18-95 3-19-96 4-20-97 5-21-98 6-22-99 7-23 8-24 9-25 :-26 <-27 =-28 >-29
?-30 @-31 a-32 b-33 c-34 d-35 e-36 f-37 g-38 h-39 i-40 j-41 k-42 l-43 m-44 n-45 o-46 p-47 q-48 r-49 s-50 t-51 u-52 v-53 w-54 x-55 y-56 z-57 [-58 \-59 ]-60
^-61 _-62 A-63 B-64 C-65 D-66 E-67 F-68 G-69 H-70 I-71 J-72 K-73 L-74 M-75 N-76 O-77 P-78 Q-79 R-80 S-81 T-82 U-83 V-84 W-85 X-86 Y-87 Z-88 {-89 |-90 }-91
~-92 !-100
 # $%&'()*+,-./0123456789:<=>?@abcdefghijklmnopqrstuvwxyz[\]^_ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~0123456!
```

Test 2

myplaintext:

‘ # $%&'()*+,-./0123456789:<=>?@abcdefghijklmnopqrstuvwxyz[\]^_ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~0123456!!’

stringa di 101 caratteri

```
# ============================================================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ============================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "C"                           # mycypher string where to indicate which encryption algorithms apply to myplaintext
myplaintext: .string " # $%&'()*+,-./0123456789:<=>?@abcdefghijklmnopqrstuvwxyz[\]^_ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~0123456!!"    #

```

```
Invalid myplaintext string!
```

Test 3

myplaintext: ‘’ , stringa vuota

```
# ============================================================================================================================
#          Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                          Messaggi Cifrati
#
#  Name: Gianni Magherini
#  Serial Number: 6341492
# ============================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "C"                           # mycypher string where to indicate which encryption algorithms apply to myplaintext s
myplaintext: .string ""    # myplaintext is the string which is encrypt/decrypt (Decided by the user)
```

```
Invalid myplaintext string!
```

Test 4

myplaintext: ‘Carattere çæ’

carattere non compreso tra 32 ≤ cod(c) ≤ 127, ç = ASCII 135 e æ = ASCII 145

```
# ============================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ============================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incc
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorre
buffPosition: .word 0                            # Store the position of each char for Decryp
buffOccurString: .word 2000                      # Address where store the string of the Encr
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decide
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "C"                           # mycypher string where to indicate which encry
myplaintext: .string "Carattere çæ"    # myplaintext is the string which is encrypt/decrypt (
```

```
Invalid myplaintext string!
```

mycypher

- massimo 5 caratteri (quindi con 1≤n≤5)
- Si (con 1≤i≤n) corrisponde ad uno fra i caratteri ‘A’, ‘B’, ‘C’, ‘D’

Test 1

mycypher: ‘ABCDA’ , stringa di 5 caratteri

myplaintext: ‘Prova’ sostK: 1 blocKey: ‘OLE’

```
#
#                          Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ============================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "ABCDA"                        # mycypher string where to indicate which encryption algorithms apply to myplai
myplaintext: .string "Prova"    # myplaintext is the string which is encrypt/decrypt (Decided by the user)
```

```
Prova
Qspwb
 ?5F.
 -1 ?-2 5-3 F-4 .-5
 -8 ?-7 4-6 u-5 .-4
 -8 ?-7 4-6 v-5 .-4
 -8 ?-7 4-6 u-5 .-4
 -1 ?-2 5-3 F-4 .-5
 ?5F.
Qspwb
Prova
```

Test 2

mycypher: ‘ABCDAD’ , stringa di 6 caratteri

```
# ============================================================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                            Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ============================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "ABCDAD"                         # mycypher string where to indicate which encryption algorithms apply to mypl
myplaintext: .string "Prova"    # myplaintext is the string which is encrypt/decrypt (Decided by the user)


```

```
Invalid mycypher string!
```

Test 3

mycypher: ‘’ , stringa vuota

```
# ============================================================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                            Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ============================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string ""                           # mycypher string where to indicate which encryption algorithms apply to mypl
myplaintext: .string "Prova"    # myplaintext is the string which is encrypt/decrypt (Decided by the user)


```

```
Invalid mycypher string!
```

Test 4

mycypher: ‘ABCfA’ , carattere invalido ‘f’

```
# ================================================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "ABCfA"                         # mycypher string where to indicate which encryption algorithms appl
myplaintext: .string "Prova"    # myplaintext is the string which is encrypt/decrypt (Decided by the user)

```

```
Invalid mycypher string!
```

Cifrario di Cesare:

Test 1

myplaintext: ‘AMO AssEMbLY’ mycypher: ‘A’ shift alfabetico(sostK): 1

Risultato atteso: ‘BNP BttFNcMZ’

```
# ================================================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word   1                                # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "A"                           # mycypher string where to indicate which encryption algorithms ap
myplaintext: .string "AMO AssEMbLY"    # myplaintext is the string which is encrypt/decrypt (Decided by the user


```

```
AMO AssEMbLY
BNP BttFNcMZ
AMO AssEMbLY
```

Test 2

myplaintext: ‘1_AMO AssEMbLY’ mycypher: ‘A’ shift alfabetico(sostK): -1

Risultato atteso: ‘1_ZLN ZrrDLaKX’

```
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"     # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"        # Output for mycypher string when is incorrect
buffPosition: .word 0                             # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                       # Address where store the string of the Encryption Occurrences
sostK: .word  -1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                            # Key for Block cipher (Decided by user)
mycypher: .string "A"                           # mycypher string where to indicate which encryption algorithms ap
myplaintext: .string "1_AMO AssEMbLY"     # myplaintext is the string which is encrypt/decrypt (Decided by the us

```

```
1_AMO AssEMbLY
1_ZLN ZrrDLaKX
1_AMO AssEMbLY
```

Test 3

myplaintext: ‘1234_%&AMO AssEMbLY’ shift alfabetico(sostK): 28
Risultato atteso: ‘1234_%&COQ CuuGOdNA’

```
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"     # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"        # Output for mycypher string when is incorrect
buffPosition: .word 0                             # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                       # Address where store the string of the Encryption Occurrences
sostK: .word  28                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                            # Key for Block cipher (Decided by user)
mycypher: .string "A"                           # mycypher string where to indicate which encryption algorithms ap
myplaintext: .string "1234_%&AMO AssEMbLY"     # myplaintext is the string which is encrypt/decrypt (Decided by

```

```
1234_%&AMO AssEMbLY
1234_%&COQ CuuGOdNA
1234_%&AMO AssEMbLY
```

Test 4

myplaintext: ‘AMO AssEMbLY-1’ shift alfabetico(sostK): 1 mycypher: ‘AAAAA’
Risultato atteso: ‘BNP BttFNcMZ-1’ → ‘COQ CuuGOdNA-1’ → ‘DPR DvvHPeOB-1’ → ‘EQS EwwIQfPC-1’ → ‘FRT FxxJRgQD-1’

A	M	O	A	s	s	E	M	b	L	Y	-	1
B	N	P	B	t	t	F	N	c	M	Z	-	1
C	O	Q	C	u	u	G	O	d	N	A	-	1
D	P	R	D	v	v	H	P	e	O	B	-	1
E	Q	S	E	w	w	I	Q	f	P	C	-	1
F	R	T	F	x	x	J	R	g	Q	D	-	1

Alfabeto : a-b-c-d-e-f-g-h-i-j-k-l-m-n-o-p-q-r-s-t-u-v-w-x-y-z

```
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"     # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"        # Output for mycypher string when is incorrect
buffPosition: .word 0                             # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                       # Address where store the string of the Encryption Occurrences
sostK: .word  1                                  # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                            # Key for Block cipher (Decided by user)
mycypher: .string "AAAAA"                          # mycypher string where to indicate which encryption algorithm
myplaintext: .string "AMO AssEMbLY-1"     # myplaintext is the string which is encrypt/decrypt (Decided by the us


```

```
AMO AssEMbLY-1
BNP BttFNcMZ-1
COQ CuuGOdNA-1
DPR DvvHPeOB-1
EQS EwwIQfPC-1
FRT FxxJRgQD-1
EQS EwwIQfPC-1
DPR DvvHPeOB-1
COQ CuuGOdNA-1
BNP BttFNcMZ-1
AMO AssEMbLY-1
```

Cifrario a Blocchi:

Test 1

myplaintext: ‘LAUREATO_1’ mycypher: ‘B’ blocKey: ‘OLE’

Risultato atteso: ‘{mz!qf#{$`’

Pt	L	A	U	R	E	A	T	O	_	1
Cod(pt)	76	65	85	82	69	65	84	79	95	49
Key	O	L	E	O	L	E	O	L	E	O
Cod(key)	79	76	69	79	76	69	79	76	69	79
Cod(ct)	**123**	**109**	**122**	**33**	**113**	**102**	**35**	**123**	**36**	**96**
ct	**{**	**m**	**z**	**!**	**q**	**f**	**#**	**{**	**$**	**'**

```
#
#                          Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ============================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "B"                           # mycypher string where to indicate which encryption algorithms apply to myplaint
myplaintext: .string "LAUREATO_1"    # myplaintext is the string which is encrypt/decrypt (Decided by the user)

```

```
LAUREATO_1
{mz!qf#{$`
LAUREATO_1
```

Test 2

myplaintext: ‘LAUREATO_1’ mycypher: ‘B’ blocKey: ‘TeS1’

Risultato atteso: ‘ &(cy&'`3v’

Pt	L	A	U	R	E	A	T	O	_	1
Cod(pt)	76	65	85	82	69	65	84	79	95	49
Key	T	e	S	1	T	e	S	1	T	e
Cod(key)	84	101	83	49	84	101	83	49	84	101
Cod(ct)	**32**	**38**	**40**	**99**	**121**	**38**	**39**	**96**	**51**	**118**
ct		**&**	**(**	**c**	**y**	**&**	**'**	**`**	**3**	**v**

```
# ================================================================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ================================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "TeS1"                          # Key for Block cipher (Decided by user)
mycypher: .string "B"                           # mycypher string where to indicate which encryption algorithms apply to myplaintex
myplaintext: .string "LAUREATO_1"    # myplaintext is the string which is encrypt/decrypt (Decided by the user)

```

```
LAUREATO_1
 &(cy&'`3v
LAUREATO_1
```

Test 3

myplaintext: ‘Prova’ blocKey: ‘12’ mycypher: ‘BBBBB’
Risultato atteso: ‘a$ (r’ → ‘r61:#’ → ‘#HBL4’ → ‘4ZS^E’ → ‘EldpV’

Pt	**P**	**r**	**o**	**v**	**a**
Cod(pt)	80	114	111	118	97
Key	1	2	1	2	1
Cod(key)	49	50	49	50	49
ct	**a**	**$**		**(**	**r**
Cod(ct)	**97**	**36**	**32**	**40**	**114**
Key	1	2	1	2	1
Cod(key)	49	50	49	50	49
ct	**r**	**6**	**1**	**:**	**#**
Cod(ct)	**114**	**54**	**49**	**58**	**35**
Key	1	2	1	2	1
Cod(key)	49	50	49	50	49
ct	**#**	**H**	**B**	**L**	**4**
Cod(ct)	**35**	**72**	**66**	**76**	**52**
Key	1	2	1	2	1
Cod(key)	49	50	49	50	49
ct	**4**	**Z**	**S**	**^**	**E**
Cod(ct)	**52**	**90**	**83**	**94**	**69**
Key	1	2	1	2	1
Cod(key)	49	50	49	50	49

```
Prova
a$ (r
r61:#
#HBL4
4ZS^E
EldpV
4ZS^E
#HBL4
r61:#
a$ (r
Prova
```

```
# ================================================================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ================================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "12"                            # Key for Block cipher (Decided by user)
mycypher: .string "BBBBB"                       # mycypher string where to indicate which encryption algorithms apply to mypla
myplaintext: .string "Prova"    # myplaintext is the string which is encrypt/decrypt (Decided by the user)

```

Cifratura Occorrenze:

Test 1

myplaintext: ‘esempio di messaggio criptato -1’ mycypher: ‘C’

Risultato atteso:

‘e-1-3-13 s-2-14-15 m-4-12 p-5-25 i-6-10-19-24 o-7-20-29 -8-11-21-30 d-9 a-16-27 g-17-18 c-22 r-23 t-26-28 --31 1-32’

```
# ================================================================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ================================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "C"                           # mycypher string where to indicate which encryption algorithms apply to myplaintext
myplaintext: .string "esempio di messaggio criptato -1"    # myplaintext is the string which is encrypt/decrypt (Decided by the us

```

```
esempio di messaggio criptato -1
e-1-3-13 s-2-14-15 m-4-12 p-5-25 i-6-10-19-24 o-7-20-29  -8-11-21-30 d-9 a-16-27 g-17-18 c-22 r-23 t-26-28 --31 1-32
esempio di messaggio criptato -1
```

Test 2

myplaintext:

‘# $%&'()*+,-./0123456789:<=>@abcdefghijklmnopqrstuvwxyz[\]^_ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~0123456!’

mycypher: ‘C’

Risultato atteso:

‘ -1-3 #-2 $-4 %-5 &-6 '-7 (-8)-9 *-10 +-11 ,-12 --13 .-14 /-15 0-16-93 1-17-94 2-18-95 3-19-96 4-20-97 5-21-98 6-22-99 7-23 8-24 9-25 :-26 <-27 =-28 >-29 ?-30 @-31 a-32 b-33 c-34 d-35 e-36 f-37 g-38 h-39 i-40 j-41 k-42 l-43 m-44 n-45 o-46 p-47 q-48 r-49 s-50 t-51 u-52 v-53 w-54 x-55 y-56 z-57 [-58 \-59]-60 ^-61 _-62 A-63 B-64 C-65 D-66 E-67 F-68 G-69 H-70 I-71 J-72 K-73 L-74 M-75 N-76 O-77 P-78 Q-79 R-80 S-81 T-82 U-83 V-84 W-85 X-86 Y-87 Z-88 {-89 |-90 }-91 ~-92 !-100’

```
# ================================================================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ================================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "C"                           # mycypher string where to indicate which encryption algorithms apply to myplaintext
myplaintext: .string " # $%&'()*+,-./0123456789:<=>?@abcdefghijklmnopqrstuvwxyz[\]^_ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~01234561"    # m
```

```
 # $%&'()*+,-./0123456789:<=>?@abcdefghijklmnopqrstuvwxyz[\]^_ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~0123456!
 -1-3 #-2 $-4 %-5 &-6 '-7 (-8 )-9 *-10 +-11 ,-12 --13 .-14 /-15 0-16-93 1-17-94 2-18-95 3-19-96 4-20-97 5-21-98 6-22-99 7-23 8-24 9-25 :-26 <-27 =-28 >-29 ?-30
@-31 a-32 b-33 c-34 d-35 e-36 f-37 g-38 h-39 i-40 j-41 k-42 l-43 m-44 n-45 o-46 p-47 q-48 r-49 s-50 t-51 u-52 v-53 w-54 x-55 y-56 z-57 [-58 \-59 ]-60 ^-61 _-62
A-63 B-64 C-65 D-66 E-67 F-68 G-69 H-70 I-71 J-72 K-73 L-74 M-75 N-76 O-77 P-78 Q-79 R-80 S-81 T-82 U-83 V-84 W-85 X-86 Y-87 Z-88 {-89 |-90 }-91 ~-92 !-100
 # $%&'()*+,-./0123456789:<=>?@abcdefghijklmnopqrstuvwxyz[\]^_ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~0123456!
```

Test 3

myplaintext: ‘a -’ mycypher: ‘CCCCC’

Risultato atteso:

→ ‘A-1 -2 --3’

→ ‘A-1 --2-6-9-10 1-3 -4-5-8 2-7 3-11’

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

→ ‘A-1 --2-5-6-8-10-12-14-17-20-23-26-29-32-35-40-42-45-48-51-56-58-61-64-67-70-75-77-82-86-91-96-99-104-109-114-119 1-3-15-18-21-25-39-43-46-59-62-72-87-88-92-97 -4-38-54-55-73-80-84-89-94-102-107-112-117 2-7-19-24-27-30-33-65-68-74-78-101-105-106-110-115 5-9-31-53-60-108 6-11-47-81-116 8-13-79-98-113 0-16-66-90-121 7-22-69-76-118 3-28-36-37-41-44-49 -52-71-95-100-120 9-34-63-83-85 4-50-57-93-103-111’

→ ‘A-1 --2-5-6-8-10-12-14-17-20-23-26-29-32-35-38-41-44-47-50-53-56-59-62-65-68-71-74-77-80-83-86-89-92-95-98-102-106-110-116-118-121-124-127-130-133-136-139-142-145-148-151-154-157-162-164-167-170-173-176-179-182-185-188-192-196-200-206-208-211-214-217-220-223-226-229-232-235-239-243-247-251-257-259-262-265-268-274-277-280-283-289-292-295-298-304-307-310-313-319-322-325-328-334-337-340-343-346-349-352-355-358-361-365-371-374-377-380-385-388-391-394-398 1-3-15-18-21-24-58-67-91-99-103-107-108-111-112-115-119-122-126-189-193-197-198-201-202-209-236-238-240-244-248-249-252-253-261-269-275-276-282-284-285-290-299-300-305-314-316-329-330-345-357-362-366-395-399-400-401 -4-114-160-161-204-255-272-287-302-317-332-369-383 2-7-19-27-30-33-36-40-49-85-125-128-144-147-156-191-199-205-212-215-315-320-321-335-354-367 5-9-43-52-57-60-63-79-120-129-140-168-171-172-225-242-254-256-263-353-360-382-386-389 6-11-34-61-66-69-72-88-94-138-143-224-227-246-266-273-286-306-308-309-323-327-339-375 8-13-55-64-84-87-123-149-152-153-166-177-180-183-228-234-271-281-288-297-331-336-378-381 0-16-28-46-76-100-104-178-190-194-219-237-241-245-250-267-270-303-312-363-364-368-387-396 4-22-45-48-51-54-70-101-109-134-137-163-169-181-187-213-231-278-344-347-348-350-373-384 7-25-73-75-78-81-82-146-150-159-174-195-203-207-216-230-233-279-293-318-326-342-356-390 3-31-39-42-117-131-135-165-175-218-221-222-260-264-291-301-333-338-341-372-376-379-393-397 9-37-90-93-96-97-105-113-132-141-155-158-184-186-210-258-294-296-311-324-351-359-370-392’

```
# ============================================================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ============================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "CCCCC"                         # mycypher string where to indicate which encryption algorithms apply to myplair
myplaintext: .string "A -"    # myplaintext is the string which is encrypt/decrypt (Decided by the user)

```

```
A -
A-1  -2 --3
A-1 --2-6-9-10 1-3  -4-5-8 2-7 3-11
A-1 --2-5-6-8-10-12-17-21-23-25-29-33 1-3-13-16-34-35  -4-15-19-20-27-31 2-7-28 6-9 9-11 0-14 3-18-32 4-22 5-24 8-26 7-30
A-1 --2-5-6-8-10-12-14-17-20-23-26-29-32-35-40-42-45-48-51-56-58-61-64-67-70-75-77-82-86-91-96-99-104-109-114-119 1-3-15-18-21-25-39-43-46-59-62-72-87-88-92-97
-4-38-54-55-73-80-84-89-94-102-107-112-117 2-7-19-24-27-30-33-65-68-74-78-101-105-106-110-115 5-9-31-53-60-108 6-11-47-81-116 8-13-79-98-113 0-16-66-90-121
7-22-69-76-118 3-28-36-37-41-44-49-52-71-95-100-120 9-34-63-83-85 4-50-57-93-103-111
A-1
--2-5-6-8-10-12-14-17-20-23-26-29-32-35-38-41-44-47-50-53-56-59-62-65-68-71-74-77-80-83-86-89-92-95-98-102-106-110-116-118-121-124-127-130-133-136-139-142-145-
148-151-154-157-162-164-167-170-173-176-179-182-185-188-192-196-200-206-208-211-214-217-220-223-226-229-232-235-239-243-247-251-257-259-262-265-268-274-277-280
-283-289-292-295-298-304-307-310-313-319-322-325-328-334-337-340-343-346-349-352-355-358-361-365-371-374-377-380-385-388-391-394-398
1-3-15-18-21-24-58-67-91-99-103-107-108-111-112-115-119-122-126-189-193-197-198-201-202-209-236-238-240-244-248-249-252-253-261-269-275-276-282-284-285-290-299
-300-305-314-316-329-330-345-357-362-366-395-399-400-401  -4-114-160-161-204-255-272-287-302-317-332-369-383
2-7-19-27-30-33-36-40-49-85-125-128-144-147-156-191-199-205-212-215-315-320-321-335-354-367
5-9-43-52-57-60-63-79-120-129-140-168-171-172-225-242-254-256-263-353-360-382-386-389
6-11-34-61-66-69-72-88-94-138-143-224-227-246-266-273-286-306-308-309-323-327-339-375
8-13-55-64-84-87-123-149-152-153-166-177-180-183-228-234-271-281-288-297-331-336-378-381
0-16-28-46-76-100-104-178-190-194-219-237-241-245-250-267-270-303-312-363-364-368-387-396
4-22-45-48-51-54-70-101-109-134-137-163-169-181-187-213-231-278-344-347-348-350-373-384
7-25-73-75-78-81-82-146-150-159-174-195-203-207-216-230-233-279-293-318-326-342-356-390
3-31-39-42-117-131-135-165-175-218-221-222-260-264-291-301-333-338-341-372-376-379-393-397
9-37-90-93-96-97-105-113-132-141-155-158-184-186-210-258-294-296-311-324-351-359-370-392
A-1 --2-5-6-8-10-12-14-17-20-23-26-29-32-35-40-42-45-48-51-56-58-61-64-67-70-75-77-82-86-91-96-99-104-109-114-119 1-3-15-18-21-25-39-43-46-59-62-72-87-88-92-97
-4-38-54-55-73-80-84-89-94-102-107-112-117 2-7-19-24-27-30-33-65-68-74-78-101-105-106-110-115 5-9-31-53-60-108 6-11-47-81-116 8-13-79-98-113 0-16-66-90-121
7-22-69-76-118 3-28-36-37-41-44-49-52-71-95-100-120 9-34-63-83-85 4-50-57-93-103-111
A-1 --2-5-6-8-10-12-17-21-23-25-29-33 1-3-13-16-34-35  -4-15-19-20-27-31 2-7-28 6-9 9-11 0-14 3-18-32 4-22 5-24 8-26 7-30
A-1 --2-6-9-10 1-3  -4-5-8 2-7 3-11
A-1  -2 --3
A -
```

Dizionario

Test 1

myplaintext: ‘myStr0ng P4ssW_’ mycypher: ‘D’

Risultato atteso: ‘NBhGI9MT k5HHd_’

Pt	m	y	S	t	r	0	n	g		P	4	s	s	W	_
Tipo ci	min	min	mai	min	min	num	min	min	sym	mai	num	min	min	mai	sym
ct	**N**	**B**	**h**	**G**	**I**	**9**	**M**	**T**		**k**	**5**	**H**	**H**	**d**	**_**

```
# ================================================================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ================================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "D"                           # mycypher string where to indicate which encryption algorithms apply to myplaintex
myplaintext: .string "myStr0ng P4ssW_"    # myplaintext is the string which is encrypt/decrypt (Decided by the user)

```

```
myStr0ng P4ssW_
NBhGI9MT k5HHd_
myStr0ng P4ssW_
```

Test 2

myplaintext: ‘PRova9&TeST-2’ mycypher: ‘D’

Risultato atteso: ‘kiLEZ0&gVhg-7’

Pt	P	R	o	v	a	9	&	T	e	S	T	-	2
Tipi ci	mai	mai	min	min	min	num	sym	mai	min	mai	mai	sym	num
ct	k	i	L	E	Z	0	&	g	V	h	g	-	7

```
# ================================================================================================================
#            Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                              Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ================================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "D"                           # mycypher string where to indicate which encryption algorithms apply to myplain
myplaintext: .string "PRova9&TeST-2"    # myplaintext is the string which is encrypt/decrypt (Decided by the user)

```

```
PRova9&TeST-2
kiLEZ0&gVhg-7
PRova9&TeST-2
```

Test con molteplici Algoritmi

myplaintext: ‘myStr0ng P4ssW_’ mycypher: ‘DABC’

sostK: 1 blocKey: ‘OLE’

Pt	m	y	S	t	r	0	n	g		P	4	s	s	W	_
Tipo ci	min	min	mai	min	min	num	min	min	sym	mai	num	min	min	mai	sym
ct	**N**	**B**	**h**	**G**	**I**	**9**	**M**	**T**		**k**	**5**	**H**	**H**	**d**	**_**

Dizionario → ‘NBhGI9MT k5HHd_’

N	B	h	G	I	9	M	T		k	5	H	H	d	_
O	**C**	**i**	**H**	**J**	**9**	**N**	**U**		**l**	**5**	**I**	**I**	**e**	**_**

Cifrario di Cesare → ‘OCiHJ9NU l5IIe_’

Pt	O	C	i	H	J	9	N	U		l	5	I	I	e	_
Cod(pt)	79	67	105	72	74	57	78	85	32	108	53	73	73	101	45
Key	O	L	E	O	L	E	O	L	E	O	L	E	O	L	E
Cod(Key)	79	76	69	79	76	69	79	76	69	79	76	69	79	76	69
Cod(ct)	**126**	**111**	**46**	**119**	**118**	**94**	**125**	**33**	**69**	**59**	**97**	**110**	**120**	**49**	**36**
ct	**~**	**o**	**.**	**w**	**v**	**^**	**}**	**!**	**E**	**;**	**a**	**n**	**x**	**1**	**$**

Cifrario a Blocchi → ‘~o.wv^}!E;anx1$’

Cifratura Occorrenze → ‘~-1 o-2 .-3 w-4 v-5 ^-6 }-7 !-8 E-9 ;-10 a-11 n-12 x-13 1-14 $-15’

```
# ============================================================================================================
#           Progetto Assembly RISC-V per il Corso di AdE 19/20
#
#                          Messaggi Cifrati
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
# ============================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption Occurrences
buffOccurString: .word 2000                      # Address where store the string of the Encryption Occurrences
sostK: .word  1                                 # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"                           # Key for Block cipher (Decided by user)
mycypher: .string "DABC"                         # mycypher string where to indicate which encryption algorithms apply to myplaint
myplaintext: .string "myStr0ng P4ssW_"    # myplaintext is the string which is encrypt/decrypt (Decided by the user)

```

```
myStr0ng P4ssW_
NBhGI9MT k5HHd_
OCiHJ9NU l5IIe_
~o.wv^}!E;anx1$
~-1 o-2 .-3 w-4 v-5 ^-6 }-7 !-8 E-9 ;-10 a-11 n-12 x-13 1-14 $-15
~o.wv^}!E;anx1$
OCiHJ9NU l5IIe_
NBhGI9MT k5HHd_
myStr0ng P4ssW_
```

Codice assembly

```
#=============================================================================================================
#             RISC-V Assembly Project for the AdE 19/20 Course
#
#                               Cipher Message
#
#   Name: Gianni Magherini
#   Serial Number: 6341492
#=============================================================================================================
.data
errStr: .string "Invalid myplaintext string!"    # Output for myplaintext string when is incorrect
errCyp: .string "Invalid mycypher string!"       # Output for mycypher string when is incorrect
buffPosition: .word 0                            # Store the position of each char for Decryption
Occurrences
buffOccurString: .word 2000          # Address where store the string of the Encryption Occurrences
sostK: .word   -1                    # Alphabetical shift for Caesar cipher (Decided by user)
blocKey: .string "OLE"               # Key for Block cipher (Decided by user)
# mycypher string where to indicate which encryption algorithms apply to myplaintext string
# (Decided by user)
mycypher: .string "ABCD"
# myplaintext is the string which is encrypt/decrypt (Decided by the user)
myplaintext: .string "~ #$%&'(56789:;<=>?@abcdefghijklmnopqrstuvwxyz[\]^_ABCDEFGHIJKLMNOPQRST"



.text
main:
la s0, myplaintext   # Load head of myplaintext
la s1, mycypher      # Load head of mycypher


add a2, s0, zero     # a2 <- myplaintext
add a3, s1, zero     # a3 <- mycypher
jal plaiCypCheck      # Jump to the procedure that checks the fairness of myplaintext and mycypher
```

```
jal print_plaintext  # Jump to the procedure which prints the string myplaintext

add a0, s0, zero     # a0 <- myplaintext the string to be encrypted/decrypted
jal cipher           # Jump to the procedure for the encryption of the myplaintext string

jal decipher         # Jump to the procedure for the dencryption of the cyphertext

j endMain            # Jump to the ends of the progam


# ================================================
#              plaiCypCheck
#
# Note: Procedure that checks the validity of myplaintext
#       string and mycypher string
#
# ================================================
plaiCypCheck:
li t0, 0       # Loop Counter
li t4, 0       # Loop Counter
li t5, 100     # myplaintext limit number of characters allowed
li t6, 5       # mycypher limit number of characters allowed


# ========================================================
#                  cpyCheckLoop
# Note: Maximum string length is 5 characters.
#       Each character can only be one of 'A','B','C','D'.
#
# Parameters:
#   a3 <- string mycypher                t1 <- current character
#   a4 <- load character 'A'             t3 <- base address
#   a5 <- load character 'D'             t6 <- max string length (5)
#   t0 <- loop counter for cpyCheckLoop
#
# Possible result:
#   1. End of cpyCheckLoop with string mycypher which passes
#      all the tests
#   2. All tests fail, jump to cryCheckError procedure
# ========================================================
cpyCheckLoop:
li a4, 65                    # ASCII code for character 'A'
li a5, 68                    # ASCII code for character 'D'

add t3, t0, a3               # Pass through mycypher string
lb t1, 0(t3)                 # Current character to analyze

beq t1, zero, strCheckLoop    # Check if the string has been fully viewed, if is true jump to
'strCheckLoop'

# ========================================================
# Check if the character is not contained between A and D
# ========================================================
blt t1, a4, cryCheckError
bgt t1, a5, cryCheckError

addi t0, t0, 1               # Increment the counter
bgt t0, t6, cryCheckError   # Check if the string is more than 5, if is true jump to 'cryCheckError'
j cpyCheckLoop


# ============================================================================
#                   strCheckLoop
# Note: Maximum string length is 100 characters.
#       Each character can be between 32 and 127 (ASCII code).
#
# Parameters:
#   a2 <- string myplaintext             t2 <- base address
#   a4 <- load character 'Spc'           t4 <- loop counter for strCheckLoop
#   a5 <- load character 'Del'           t5 <- max string length (100)
#   t1 <- current character
#
# Possible result:
#    1. If mycypher is an empty string jump to cryCheckError,
#       without checking myplaintext
#    2. End of strCheckLoop with string myplaintext which passes
#       all the tests
```

```
#    3. All tests fail, jump to strCheckError procedure
# ============================================================================
strCheckLoop:
beq t0, zero cryCheckError   # Check if mycypher is an empty string
li a4, 32                    # ASCII code for character 'Spc'
li a5, 127                   # ASCII code for character 'Del'

add t2, t4, a2               # Pass through myplaintext string
lb t1, 0(t2)                 # Current character to analyze

beq t1, zero, strCheckExit   # Check if the string has been fully viewed, if is true jump to
strCheckExit

# ============================================================================
# Check if the character is not contained between  'Spc' and 'Del'
# ============================================================================
blt t1, a4, strCheckError    # Character less then 32
bgt t1, a5, strCheckError    # Character greater then 127

addi t4, t4, 1               # Increment the counter
bgt t4, t5, strCheckError # Check if the string is more than 100, if is true jump to 'strCheckError'
j strCheckLoop

# ============================================================================
# Check if myplaintext is an empty string, if is true jump to
# strCheckError otherwise return to the main in the expected position
# ============================================================================
strCheckExit:
beq t4, zero, strCheckError
jr ra

# ============================================================================
# Mycypher tests fail, print 'Invalid mycypher string!' and ends the program
# ============================================================================
cryCheckError:
la  a0, errCyp
li a7, 4
ecall           # Print error message
j endMain       # Jump to end

# ============================================================================================
# Myplaintext tests fail, print 'Invalid myplaintext string!' and ends the program
# ============================================================================================
strCheckError:
la  a0, errStr
li a7, 4
ecall            # Print error message
j endMain        # Jump to end


# ============================================================================================
#                               cipher
#
#  Note: Procedure that iterates mycypher string to extrapolate the letter of the
#        algorithm to apply on the myplaintext string (afterwards on 'cyphertext')
#
#  Paramethers:
#    a1 <- length of mycypher string      t0 <- loop Counter
#    a3 <- mycypher string                t1 <- current counter
#    ra <- return address (for main)      t2 <- Base address (mycypher + index)
#
#  Return:
#    a1 <- length of mycypher string
#    t1 <- current counter used from algChoice
# ============================================================================================
cipher:
addi sp, sp, -4              # Adjust stack for 1 items
sw ra, 0(sp)               # Save return address in the stack, will be used for return to the main

li t0, 0                     # Loop Counter

loopCipher:
add t2, t0, a3               # Pass through mycypher string
lb  t1, 0(t2)                # Current character
beq t1, zero, endLoopCipher  # Check the end of mycypher string

# ============================================================================================
# Jump to the procedure that decide which algorithm of cypher must be applied to cyphertext
```

```
# ================================================================================================
jal algChoice

# ================================================================
# Jump to the procedure that print the result of each encryption
# ================================================================
jal printCipherDecipher

updateLoopCipher:
addi t0, t0, 1         # Increment the counter
j loopCipher

endLoopCipher:
lw  ra, 0(sp)          # Load return address
addi sp, sp, 4         # Restore stack

addi a1, t0, -1        # Number of character in mycypher
jr ra


# ================================================================================
#                              decipher
#
#  Note: Procedure that iterates mycypher string in rever order to extrapolate the letter
#        of the algorithm to apply on the cyphertext to get myplaintext again.
#
#  Paramethers:
#    a1 <- length of mycypher string               t0 <- a1
#    a3 <- mycypher string                         t1 <- Current character
#    a5 <- flag that trigger the decry. alg.       t2 <- Base address
#    ra <- return address (for main)
#
#  Return:
#     t1 <- current counter used from algChoice
# ================================================================================
decipher:
addi sp, sp, -4       # Adjust stack for 1 items
sw ra, 0(sp)          # Save return address in the stack, will be used for return to the main


add t0, a1, zero      # Number of character in mycypher (Loop Counter)
addi a5, zero, -1     # Flag that trigger the use of decryption algorithms

loopDecipher:
add t2, t0, a3        # Pass through mycypher string
lb  t1, 0(t2)         # Current character

beq t1, zero, endLoopDecipher     # End of mycypher string

# ================================================================================================
# Jump to the procedure that decide which algorithm of cypher must be applied to cyphertext
# ================================================================================================
jal algChoice

# ================================================================
# Jump to the procedure that print the result of each decryption
# ================================================================
jal printCipherDecipher

updateLoopDecipher:
addi t0, t0, -1      # Decrease the counter
j loopDecipher

endLoopDecipher:
lw  ra, 0(sp)        # Load return address
addi sp, sp, 4       # Restore stack

jr ra                # Jump to the main


# ================================================================================
#                          algChoice
#  Note: Compare the character extracted from the cipher or decipher
#        procedures and select the specified encryption/decryption algorithm
#
#  Paramethers:
#      t1 <- curret mycypher character        t4 <- character B in ASCII
#      t3 <- character A in ASCII             t5 <- character C in ASCII
```

```
#
# ============================================================================
algChoice:
li t3, 65  # A in ASCII
li t4, 66  # B in ACSII
li t5, 67  # C in ASCII

beq t1, t3, caesarCipherDecipher  # Caesar cipher is invoke if  one character of mycypher is equal A
beq t1, t4, blockCipherDecipher   # Block cipher is invoke if  one character of mycypher is equal B
beq t1, t5, encrDecrOccurrence    # Encryption Occur. is invoke if  one char. of mycypher is equal C
j dictionary             # Dictionary is invoked otherwise, that is when the mycypher character is D




# ============================================================================
#                               caesarCipherDecipher
#
#  Note: Inizializes the Caesar Algorithm...
#
#  Paramethers:
#      t1 <- Current index             a4 <- Alphabetical shift (sostK)
#      t3 <- flag checks               a5 <- flag triggers decryption
#      t6 <- value of 26 (alphabet characters)
#            use for modulus of sostk
#
# ============================================================================
caesarCipherDecipher:
addi sp, sp, -8               # Adjust stack for 2 items
sw ra, 4(sp)                  # Return address
sw t0, 0(sp)                  # Loop counter of cipher/decipher procedure

li t1, 0                      # Current index
li t3, -1                     # flag checks <- if is a letter/if we are in the decryption
li t6, 26                     # t6 <- alphabet characters
lw a4, sostK                  # Alphabetical shift for Caesar algorithm


#============================================================================================================
#                          moduleCipher/moduleNegCipher
#  Note: Used to apply  modulus to the alphabetic shift(sostK) limiting its value to the characters
#        of the alphabet.
#        Since the modulus instruction is not present in Risc-V; the remainder(REM) is used here, in
#        the case of positive Sostk.
#        If Sostk is negative, then two possible choices for the remainder occur. 'In mathematics,
#        the remainder is positive, but implementations in programming languages differ as in
#        Risc-V'. To overcome this problem, the result of the remainder is taken and then it is
#        subtracted from the number of characters in the alphabet(26).
#        Which therefore is equivalent to the result you would have with the modulus.
#
#============================================================================================================
moduleSostK:
rem a4, a4, t6                  # a4 <- sostK rem 26
blt a4, zero, negModuleSostK    # In case of sostK is negative
j coreCaesarAlgorithm

negModuleSostK:
add a4, t6, a4                  # a4 <- 26 - (sostK rem 26)
j coreCaesarAlgorithm


# ============================================================================
#                               coreCaesarAlgorithm
#
#  Note: Iterates myplaintext/cyphertext string to extrapolate the character
#        where to apply the Caesar cipher/decipher. A jump and link of
#        'getCharOffset' is perform to get the offset, useful later.
#
#
#  Paramethers:
#      t1 <- loop counter              a6 <- current character
#      t2 <- base address              a0 <- myplaintext/cyphertext string
#
# ============================================================================
coreCaesarAlgorithm:
add t2, t1, a0                          # Pass through myplaintext/cyphertext string
lb a6, 0(t2)                            # Current character
```

```
beq a6, zero caesarCipDecipEnd   # End of myplaintext/cyphertext string (End the Caesar's algorithm)

jal getCharOffset                           # Returns the right offset to execute caesar operation
add t5, a6, zero                           # t5 take the value of the current character
# ============================================================================
#  Chack if the current character is a letter (a1 <- return -1 from
#  'getCharOffset' procedure if the current character isn't a letter)
# ============================================================================
beq a1, t3, caesarNextChar                  # Check if it isn't a letter
addi t4, a1, 0                              # t4 <- offset

# Check if the alogorithm to apply is the encryption or decryption,through a flag(a5)
beq a5, t3, decipherCaesarAlgorithm


# ============================================================================================
#                               cipherCaesarAlgorithm
#
#  Note: The cipher algorithm is the following
#           {[(l - o) + k] % 26} + o
#
#        Where:
#           l <- letter ASCII     o <- ASCII offset character
#           k <- sostk % 26
#  Result:
#     t0 <- store the result in t2(base address)
# ============================================================================================
cipherCaesarAlgorithm:
sub t0, t5, t4                               # t0 = letter - offset
add t0, t0, a4                               # t0 += sostK
rem t0, t0, t6                               # t0 rem (26)
add t0, t0, t4                               # t0 += offset

sb t0, 0(t2)                                # Store the result in the specified location
j caesarNextChar


# ============================================================================================================
#                 decipherCaesarAlgorithm/negSostKDecipCaesarAlgorithm
#
#  Note: The decipher algorithm is the following
#               [(l - o) - k] + o
#
#         if (l - o - k) < 0 is the following
#               {[(l - o) - k] + 26*} + o
#         Where:
#           l <- letter ASCII      o <- ASCII offset character
#           k <- sostk % 26 / key
#  Result:
#      t0 <- store the result in t2(base address
#
#   *equivalent to use modulus of 26, since ((l - o) - k) is never less then -26 in this case
# ============================================================================================================
decipherCaesarAlgorithm:
sub t0, t5, t4                                 # t0 = letter - offset
sub t0, t0, a4                                 # t0 = t0 - a4(sostK)

blt t0, zero, negSostKDecipCaesarAlgorithm
add t0, t0, t4                                 # t0 += offset

sb t0, 0(t2)                                   # Store the result in the specified location
j caesarNextChar

# ======================================================
# formula used when -> (letter ASCII - offset - key) < 0
# ======================================================
negSostKDecipCaesarAlgorithm:
add t0, t6, t0                                 # t0 <- 26 - t0 /
add t0, t0, t4                                 # t0 += offset

sb t0, 0(t2)                                   # Store the result in the specified location
j caesarNextChar


caesarNextChar:
addi t1, t1, 1                                # Increment the counter for 'coreCaesarAlgorithm'
j coreCaesarAlgorithm
```

```
caesarCipDecipEnd:
lw t0, 0(sp)                          # Load loop counter of cipher/decipher procedure
lw ra, 4(sp)                          # Load return address, to return to cipher/decipher procedure
addi sp, sp, 8                        # Restore stack
jr ra




#=============================================================================================
#                               blockCipherDecipher
#
#  Note: Inizializes the Block Algorithm...
#
#  Paramethers:
#      t0 <- loop counter of blocKey string             a1 <- Key of the block algorithm (blocKey)
#      t1 <- loop counter of myplaintext/cyphertext string   a6 <- flag for decipher algorithm
#      t6 <- value for the calculation of modulus (96)
#
#
#=============================================================================================
blockCipherDecipher:
addi sp, sp, -4                       # Adjust stack for 1 items
sw t0, 0(sp)                          # Loop counter of cipher/decipher procedure

la a1,  blocKey                       # Load head of Key for Block cipher/decipher
li a6, -1                             # Flag for decipher algorithm
li t0, 0                              # Loop counter of blocKey
li t1, 0                              # Loop counter of myplaintext/cyphertext
li t6, 96                             # Value for the calculation of modulus


#=============================================================================================
#                               coreBlockAlgorithm
#
#  Note: Iterates myplaintext/cyphertext string to extrapolate the character where to apply the
#        Block cipher/decipher.
#        It is also calcultated [cod(char) - 32] and [cod(blocKey) - 32], parts of the formula in
#       common to encryption and decryption.
#
#  Paramethers:
#      a0 <- myplaintext/cyphertext string               a1 <- blocKey string
#      t1 <- loop counter myplaintext/cyphertext         t0 <- loop counter blocKey string
#      t2 <- base address           ''                   t3 <- base address        ''
#      t4 <- current character      ''                   t5 <- current character   ''
#
#=============================================================================================
coreBlockAlgorithm:
add t2, t1, a0                        # Pass through myplaintext/cyphertext string
lb t4, 0(t2)                          # Current character of plain/cypher string
beq t4, zero, blockCipDecEnd  # End of myplaintext/cyphertext string (End the Block chiper/decipher)
addi t4, t4, -32                      # t4 <- cod(char) - 32

add t3, t0, a1                        # Pass through blocKey string
lb t5, 0(t3)                          # Current character of blocKey string
beq t5, zero, blocKeyLoop          # Restart the loop of blocKey string when the string is finished


coreBlockKeyAlgorithm:
addi t5, t5, -32                       # t5 <- cod(blocKey) – 32
# Check if the alogorithm to apply is the encryption or decryption,through a flag(a5)
beq a5, a6, decipherBlockAlgorithm


# ============================================================================
#                               cipherBlockAlgorithm
#
#  Note: The cipher algorithm is the following
#            {[(cod(b ij) – 32) + (cod(key j) – 32)] % 96} + 32
#
#  Result:
#     t4 <- store the result in t2(base address)
# ============================================================================
cipherBlockAlgorithm:
add t4, t4, t5                        # t4 <- (cod(char) - 32) + (cod(key) - 32)
```

```
rem t4, t4, t6                        # t0 % modulus (96)
addi t4, t4, 32

sb t4, 0(t2)                          # Store the result in the specified location
j blockNextChar

# ============================================================================================
#                   decipherBlockAlgorithm/negModuloDecipBlockAlgorithm
#
#  Note: The cipher algorithm is the following
#            [(cod(b ij) – 32) - (cod(blocKey j) – 32)] + 32
#
#          if (cod(b ij) – 32) - (cod(blocKey j) – 32) < 0 is the following
#               {[(cod(b ij) – 32) - (cod(blocKey j) – 32)] + 96*} + 32
#       Where:
#          t4 <- (cod(char) - 32)  calculated in 'coreBlockAlgorithm'
#          t5 <- (cod(blocKey) - 32)   calculated in 'coreBlockAlgorithm'
#          t6 <- 96 for the modulus
#  Result:
#     t4 <- store the result in t2(base address)
#  *equivalent to use modulus of 96, since ((l - o) - k) is never less then -96 in this case
# ============================================================================================
decipherBlockAlgorithm:
sub t4, t4, t5                                # t4 <- (cod(char) - 32) - (cod(blocKey) - 32)
blt t4, zero, negModuloDecipBlockAlgorithm
addi t4, t4, 32                               # t4 += 32

sb t4, 0(t2)                                  # Store the result in the specified location
j blockNextChar

# ============================================================================
# formula used when -> ((cod(b ij) – 32) - (cod(blocKey j) – 32)) < 0
# ============================================================================
negModuloDecipBlockAlgorithm:
add t4, t6, t4                         # t4 <- 96 - t4 / equivalent to the modulus of 96 in this case
addi t4, t4, 32                          # t4 += 32

sb t4, 0(t2)                             # Store the result in the specified location
j blockNextChar

blockNextChar:
addi t0, t0, 1                           # Increment the counter for blocKey string (coreBlockAlg.)
addi t1, t1, 1                           # Increment the counter for myplaintext/cyphertext string
(coreBloAlg)
j coreBlockAlgorithm

blocKeyLoop:
li t0, 0                                 # Refresh the loop counter for blocKey string
add t3, t0, a1                           # Pass through blocKey string for the new cicle of the string
lb t5, 0(t3)   # Only the first time, afterwards are used the 'coreBlockAlgorithm' for the iteration
j coreBlockKeyAlgorithm

blockCipDecEnd:
lw t0, 0(sp)                             # Load loop counter of cipher/decipher procedure
addi sp, sp, 4                           # Restore stack

jr ra




# ============================================================================
#                              encrDecrOccurrence
#
#  Note: Inizializes the Occurrences Algorithm...
#
# Paramethers:
#  t1  <- Base address use in 'positionEncodInASCII' and 'charPositionEncode'
#  t2  <- Base address use in 'charPositionOccur'
#  t3  <- Base address use in 'charEncodeOccur'
#  t4  <- Base address use in 'couterStringOccur'
# ============================================================================
encrDecrOccurrence:
addi sp, sp, -12                # Adjust stack for 3 items
sw ra, 8(sp)                    # Return address for cipher/decipher procedure
sw a3, 4(sp)                    # a3 <- is use for mycypher string
```

```
sw t0, 0(sp)                    # Store loop counter of cipher/decipher procedure

li t0, -1                       # Flag for decipher algorithm
li t5, 0                        # Loop counter use in 'charPositionOccur'
li t6, 0                        # Loop counter use in 'charEncodeOccur'
li a4, 45                       # Dash separator
li a6, 32                       # Space separator
lw a1, buffOccurString   # a1 <- Address where store the string of the Encryption/Decryption Occur.
la a3, buffPosition             # Load head of buffPosition

# Check if the alogorithm to apply is the encryption or decryption,through a flag(a5)
beq a5, t0, decrOccur
li t0, 0                        # Used for counts the length of myplaintext string

# ============================================================
#                    couterStringOccur
#  Note: Counts the number of char of the myplaintext string
#
#  Return: t0 <- will be used in 'charEncodeOccur'
# ============================================================
couterStringOccur:
add t2, t0, a0                  # Pass through myplaintext string
lb t4, 0(t2)                    # Current character of myplaintext string
beq t4, zero, encrOccur         # Begins the encryption just obtained the string length

addi t0, t0, 1                  # Counts length of myplaintext string
j couterStringOccur


#=============================================================================================================
#                                    encrOccur
#
#  Note: Inizializes the Encryption Occurrences...
#
#=============================================================================================================
encrOccur:
li t4, 0                         # Loop counter use in 'occurSeparDash'
# Flag to identify the first occurrence of each character except the first of 'myplaintext'
li a7, 1

#=============================================================================================================
#                                    charEncodeOccur
#
#  Note: It occupies of pass through 'myplaintext' string to carry out of two checks.
#          1. Check that it have pass through the entire string, if so the Encryption
#              Occurrences end.
#          2. Check that the current character has not already been previously encrypted, in case
#             it is present several times in the string.
#              - Check that the current character is actually present, since once encrypted a
#               procedure takes care of deleting the character from all its positions in the string.
#               To avoid the same character from being encrypted multiple times when it repeats in
#               the string.
#
#=============================================================================================================
charEncodeOccur:
add t3, t6, a0                      # Pass through myplaintext string
lb a2, 0(t3)                        # a2 <- Current character
beq t6, t0, encrDecrOccurEnd       # Check if myplaintext string is ends, jump to 'encrDecrOccurEnd'
beq a2, zero, nextcharEncodeOccur   # Check that the current character has not already been encry.


#=============================================================================================================
#                                    charPositionOccur
#
#  Note: Determines the positions where of character 'x', extracted with 'charEncodeOccur',
#        appears in 'myplaintext' string.
#        When it finds a match, it executes 'coreOccurAlgor' which encrypts the position of 'x'.
#        When 'coreOccurAlgor' is finished, 'charPositionOccur' resumes its search where it
#        left off.
#
#=============================================================================================================
charPositionOccur:
add t2, t5, a0                      # Pass through myplaintext string
lb a5, 0(t2)                        # a5 <- Current character
beq t5, t0, nextcharEncodeOccur       # Check if myplaintext string is ends
# Search for all places where character 'x' appears in 'myplaintext' string
beq a5, a2, coreOccurAlgor
```

```
charPositionOccurLoop:
addi t5, t5, 1               # Increment the counter for myplaintext string in 'charPositionOccur'
j charPositionOccur


nextcharEncodeOccur:
# Reset the flag to identify the first occurrence of each character, now to 0 because the fist char
# of 'myplaintext' has been encrypted
li a7, 0
addi t6, t6, 1                        # Increment the loop counter used in 'charEncodeOccur'
# Set up loop counter used in 'charPositionOccur', scroll by t6 positions, already encrypted
addi t5, t6, 0
j charEncodeOccur

#=============================================================================================================
#                                    coreOccurAlgor
#
#  Note: Is responsible of the encryption operations. Store the result in a3 which represents the
#        address of the head of 'buffOccurString' through 4 steps.
#       1. The first character of the string is loaded via 'firstCharEncode', this step is performed
#          only the first time for each 'cipher'.
#       2. The following characters are handled by 'followSpaceCharEncode' as they must be preceded
#          by a space.
#       3. 'occurSeparDash' follows the procedure that responsible for the separator character '-'.
#       4. The following steps are aimed at converting t5 into ASCII code (index used in
#          'charPositionOccur')
#          which represents the actual position of the occurrences. Coding required to be printed on
#          screen.
#        - For occurrences with t5 >= 10 'positionEncodInASCII' and 'charPositionEncode' are used as
#          t5 must be decomposed into units in order to perform the ASCII conversion.
#        - For occurrences with t5 < 10 'singleDigitPositEncInASCII' is used as only one digit needs
#          to be converted to ASCII.
#
#=============================================================================================================
coreOccurAlgor:
addi sp, sp, -12                      # Adjust stack for 3 items
sw t6, 8(sp)                          # Store loop counter use in 'charEncodeOccur'
sw t5, 4(sp)                          # Store loop counter use in 'charPositionOccur'
sw t0, 0(sp)                          # Store myplaintext length

# Use for purpose,flag of single digit position and used with REM instruction for multi-digits
# position
li t2, 10
li t3, 3                              # Index counter used for 'buffPosition'
# Check if it is the first iteration of the encryption, t5 use in 'charPositionOccur'
beq t5, zero, firstCharEncode
beq a7, zero, followSpaceCharEncode    # Check if it is first occurrence of each character

#=============================================================================================================
# Each position is preceded by the separator character '-' (to distinguish the elements of the
# sequence from positions)
#=============================================================================================================
occurSeparDash:
add t6, t4, a1                        # Pass through buffOccurString
sb a4, 0(t6)                          # Add dash (45 ASCII)
addi t4, t4, 1         # Increment couter position where to store the next char sequence to encrypt

addi t5, t5, 1          # t5 <- occurrences position to convert in ASCII (t5+1 because start from 0)
blt t5, t2 singleDigitPositEncInASCII  # Check if Occurrences position is less then 10

#=============================================================================================================
#  Occurrences position (t5) >= 10 is converted in ASCII. t5 which represents the position of the
#  occurrences is decomposed into single digit and store in 'buffPosition'
#   (es. t5 = 10 -> buffPosition = 1-0).
#=============================================================================================================
positionEncodInASCII:
rem t0, t5, t2                     # t5 'modulus' 10 to store in t0 the units (es. 12 REM 10 -> 2)
# Pass through 'buffPosition' to store the occurrence position broken down into single digit
add t1, t3, a3
sb t0, 0(t1)               # Store the single digit of the occurrence position in 'buffPosition'

# Decrement index to sroll the position where to store the next digit in 'buffPosition'
addi t3, t3, -1
div t5, t5, t2          # Division on t5 remove the first digit already store in 'buffPosition'.
# If t5 becomes 0 it means that the position value has already been fully decomposed in single
# digits
beq t5, zero, charPositionEncode
j positionEncodInASCII
```

```
# ================================================================================================
#  Load and convert the single digits store in 'buffPosition' to ASCII code and then store them into
#  'buffOccurString' so that they can be printed on the screen as a number.
#  (es. cod(1), cod(2) -> 49, 50)
# ================================================================================================
charPositionEncode:
addi t3, t3, 1     # Inecrement index to load the digit right of 'buffPosition' and convert in ASCII
add t1, t3, a3                          # Pass through 'buffPosition'
lb t0, 0(t1)                            # Load the single digit of the occurrence position
addi t0, t0, 48                         # Converted in ASCII encode
beq t0, zero, nextCharPositionOccur    # Check If 'buffPosition' is emty, search for next occur.

add t6, t4, a1                          # Pass through buffOccurString
sb t0, 0(t6)                # Store the each encoded digit (position) in 'buffOccurString'

addi t4, t4, 1                          # Increment counter to iterate over the 'buffOccurString'
j charPositionEncode

# ============================================================
#  Occurrences position (t5) < 10 is converted in ASCII.
# ============================================================
singleDigitPositEncInASCII:
addi t0, t5, 48                         # Converted t5 in ASCII encode
add t6, t4, a1                          # Pass through buffOccurString
sb t0, 0(t6)                            # Store the position of each char

addi t4, t4, 1                          # Increment counter to iterate over the 'buffOccurString'
j nextCharPositionOccur

# ================================================================================================
#  Executed for the first character of string to be encrypted. Store the character into
#  'buffOccurString' and increment t4 which is the counter to scroll a1 ('buffOccurString').
# ================================================================================================
firstCharEncode:
add t6, t4, a1                          # Pass through buffOccurString
sb a5, 0(t6)                            # Store the first character in 'buffOccurString'
addi t4, t4, 1         # Increment couter position where to store the next char sequence to encrypt
j occurSeparDash

# ====================================================================================================
#  Stores space followed by the occurrences of the character to be encrypted into 'buffOccurString'
#  and increment t4 which is the counter to scroll a1 ('buffOccurString'). Also increment a7, which
#  is used to identify the first occur. of each character, because the following operations require
#  only to store the position of the occurrences.
# ====================================================================================================
followSpaceCharEncode:
add t6, t4, a1                          # Pass through 'buffOccurString'
sb a6, 0(t6)                            # Add space (ASCII 32)
addi t4, t4, 1                          # Increment couter position

add t6, t4, a1                          # Pass through 'buffOccurString'
sb a5, 0(t6)                            # Store the Occurrences of the character
addi t4, t4, 1                          # Increment couter position
addi a7, a7, 1                        # Increment the flag to identify the first occur. of each char
j occurSeparDash

# ============================================================================================
#  Restore the index used in 'charEncodeOccur' and 'charPositionOccur'. Remove every occurrence
#  of each character store in 'myplaintext' string, to avoid duplication.
# ============================================================================================
nextCharPositionOccur:
lw t0, 0(sp)                            # Load length of 'myplaintext' string
lw t5, 4(sp)                            # Load loop counter use in 'charPositionOccur'
lw t6, 8(sp)                            # Load loop counter use in 'charEncodeOccur'
addi sp, sp, 12                         # Restore stack

add t2, t5, a0                          # Pass through 'myplaintext' string
sb zero, 0(t2)                          # Remove every occurrence of each character
addi t5, t5, 1    # Increment the counter used in 'charPositionOccur' (research of character occur.)
j charPositionOccur



# ============================================================================
#                               decrOccur
#
#  Note: Inizializes the Decryption Occurrences...
```

```
#
# ============================================================================
decrOccur:
addi sp, sp, -4                         # Adjust stack for 1 items
sw a5, 0(sp)                      # a5 (= -1) -> Flag that trigger the use of decryption algorithms

li t0, 0                                # Loop counter used in 'charDencodeOccur'
li a7, 10                             # Used in 'storePositionOccurCharOccurChar' for multiplication

#=============================================================================================
#                                    charDencodeOccur
#
#  Note:  The character to be decrypted is extracted from a0,'cyphertext' string, and then store
#         into 'buffOccurString', in all the positions occupied by the character before encoding.
#          (es. a-1-3...c-5 -> a_a_c__)
#         The  'cyphertext' string is cleaned up, from the character and from the next dash.
#         (es. //1-3.. -> a_a____)
#         In the following steps, all characters of the encryption will be removed to be stored into
#          decrypted 'buffOccurString'.
#
#=============================================================================================
charDencodeOccur:
add t3, t0, a0                          # Pass through 'cyphertext' string
lb t1, 0(t3)                            # Current character of cyphertext string

li t5, 0                                # Use to identify first digit of occurrence position
# t4 <- loop couter used in 'decryptOccurAlgorithm' increased by 2 to avoid char and dash
addi t4, t0, 2
sb zero, 0(t3)                          # Remove letter form a0
sb zero, 1(t3)                          # Remove dash form a0


#=============================================================================================
#                                   decryptOccurAlgorithm
#
#  Note: Load the single digit that shape the position of the occurrence of the character.
#        To do the reverse conversion from ASCII to a usable index. Make 3 checks:
#         1. Check if the 'cyphertext' string has been fully crossing, ending the decryption
#             algorithm.
#         2. Check if the loaded character is a Dash(-), if it is increment t4 (loop counter) via
#            'nextPositOccurDash' (es. ...8->-10, (-)->10).
#         3. Check if the loaded character is a Space( ), if so it means that all occurrences of the
#            character have been decrypted (es. a-1-2->' 'b-3, ' '->b...)
#            It executes 'nextCharDecOccurSpace' which takes care of moving to the next character to
#             be deciphered.
#        Convert from ASCII to position, Check if it is the first iteration of
#        'decryptOccurAlgorithm', if so, load the value of t2 in 'buffPosition' (a3)
#        and then remove the value from the 'cyphertext' character that identifies the position of
#        the occurrence.
#        Otherwise the position is multi-digit >= 10 and 'storePositionOccurChar' is executed.
#=============================================================================================
decryptOccurAlgorithm:
add t3, t4, a0                          # Pass through 'cyphertext' string
lb t2, 0(t3)                            # Load a single digit (es. ..->40, t2->4)
beq t2, zero, endDecrOccur              # Check if the 'cyphertext' string has been fully crossing
beq t2, a4, nextPositOccurDash          # Check if the loaded character is a Dash('-', ASCII 45)
beq t2, a6 nextCharDecOccurSpace        # Check if the loaded character is a Space(' ', ASCII 32)

addi t2, t2, -48                        # Converted ASCII in position (es. ASCII-cod(54)→(54-48)→6)
# Check if it is the first itera. of 'decryptOccurAlgorithm', used for multi-digits occur. position
bne t5, zero, storePositionOccurChar

add t6, a3, zero                        # Pass through 'buffPosition'
# Store the occurrences position of each character in 'buffPosition' (es. ...->1|00 -> a3=1)
sw t2, 0(t6)
sb zero, 0(t3)     # Remove the occurrence position form 'cyphertext' string (a0 -> ..8-12, ../-12)
addi t5, t5, 1                   # Increment the index in case of multi-digits occurrences position
j nextPositDecrOccurAlg

#=============================================================================================
#   It deals with recomposing the occurrence position. Since it is saved in ASCII any code must be
#   read converted and reassembled.
#   Calcultate the occurrence position >= 10 (multi-digits). The last digit store in 'buffPosition'
#   is loaded in a5, multiplied by 10 to move from units to decenes etc. Next digit save in t2 is
#   added to a5, then store back to 'buffPosition' (es. 123 ->cod(1),cod(2),cod(3)->
#   ASCII:49,50,51-> t2=49-48= 1 store in a3-> t2=50-48= 2-> load a5=1*10 = 10+t2= 12 store in a3
#   t2=51-48= 3-> load a5=12*10 = 120+t2= 123-> occurrence position converted)
#=============================================================================================
```

```
storePositionOccurChar:
# Load in a5 an intermediate calculation of occurrence position (es. 1. a5<-a3=1 -> 2. a5<-a3=12)
lw a5, 0(a3)
mul a5, a5, a7                      # a5 *= 10(a7) (es. 1. a5=1*10=10 -> 2. a5=12*10=120)
add a5, a5, t2                     # a5 += t2 (es. 1. a5=10+t2(2)=12 -> 2. a5← a5← a5=120+t2(3)=123)

add t6, a3, zero
sw a5, 0(t6)                   # Store the occurrence position in 'buffPosition'  (1.a3← a3<-a5=12)
sb zero, 0(t3)                     # Remove the occurrence position form 'cyphertext' string

nextPositDecrOccurAlg:
addi t4, t4, 1                         # Increment the counter used in 'decryptOccurAlgorithm'
j decryptOccurAlgorithm

#=============================================================================================================
#  It means that all occurrences of the character have been decrypted (es. a-1-2->' 'b-3, ' '->b-3)
#=============================================================================================================
nextCharDecOccurSpace:
jal storeDecrChar
addi t0, t4, 1        # Set the counter used in 'charDencodeOccur' (es. ->a-1-2 b-3, ...->b-3 t0=5)
j charDencodeOccur

# ============================================================================
# When there is a dash, the letter is saved in the position specified by a5 less 1
# ============================================================================
nextPositOccurDash:
jal storeDecrChar
li t5, 0                           # Reset t5, use to identify first digit of occurrence position
addi t4, t4, 1                         # Increment the counter used in 'decryptOccurAlgorithm'
j decryptOccurAlgorithm

# ============================================================================================
#  Load the occurrence position, decrease by because array starts from 0.
#  Use a5 to calculate the position of the character to store on the 'cyphertext' string.
# ============================================================================================
storeDecrChar:
lw a5, 0(a3)                   # Load the occurrence position of the decrypted character (es. a5=3)
addi a5, a5 , -1                       # a5 -= 1 (array start from 0) (es. a5-1=2)
add a2, a5, a1  # String to store the decryption with the exact location of the character (es. __a_)
sb t1, 0(a2)                           # Current character to store (es. 'a') in buffOccurString
sb zero, 0(t3)                         # Remove the position number form a0, 'cyphertext' string
jr ra


endDecrOccur:
jal storeDecrChar

lw a5, 0(sp)                           # Restore a5 to -1
addi sp, sp, 4                         # Restore stack
j encrDecrOccurEnd




#=============================================================================================================
#                                encrDecrOccurEnd
#   In the Occurrences algorithm 'buffOccurString' is used to load all cryptography and decryption
#   results starting from address 2000 (section .text) .
#   It is concluded by transferring all the characters from a1 'buffOccurString', to the starting
#   position a0, 'myplaintext'/'cyphertext'(section .data). This process is carried out to avoid
#   overlapping with the application of more than one occurrence algorithm ('CC') on the same
#   string, since it is parsed in random order according to the reference character, unlike other
#   algorithms that have a sequential trend (character that is read is the same one that is
#   encrypted, in the same position).
#
#=============================================================================================================
encrDecrOccurEnd:                      # Each char is pass by value to a0 from a1
li t5, 0                               # Loop counter

encrDecrOccurEndByValue:
add t2, t5, a1                         # Pass through 'buffOccurString'
lb t1, 0(t2)                           # Load character from 'buffOccurString'
beq t1, zero restoreStakOccurEnd       # Check when it has been fully crossing

add t3, t5, a0                         # Pass through 'myplaintext'/'cyphertext' string
sb t1, 0(t3)                       # Store the character load from 'buffOccurString' in to the string

loopEndOccur:
```

```
addi t5, t5, 1                          # Increment loop counter
sb zero, 0(t2)                          # Remove all characters form 'buffOccurString' (a1)
j encrDecrOccurEndByValue


restoreStakOccurEnd:
lw t0, 0(sp)                            # Load loop counter of cipher/decipher procedure
lw a3, 4(sp)                            # Load mycypher string used in cipher/decipher
lw ra, 8(sp)                            # Load return address, to return to cipher/decipher procedure
addi sp, sp, 12                         # Return stack
jr  ra




# ============================================================================================
#                                    dictionary
#  Note : If it is a letter apply formula letters:
#          'Z' - letter + 'a' / 'z' - letter + 'A'
#         If it is a number apply formula number: cod(57) - number
#         Otherwise it loads the same character.
#         Dictionary algorithm is used for both encryption and decryption with the same process.
# ============================================================================================
dictionary:
addi sp, sp, -8                         # Adjust stack for 2 items
sw ra, 4(sp)                            # Store return address for cipher/decipher procedure
sw t0, 0(sp)                            # Store loop counter of cipher/decipher procedure

li t1, 0                                # Loop couter used in 'coreDictionaryAlgorithm'
li t3, -1                               # Flag for character that are NOT letters
li t4, 97                               # Flag for lowercase letter
li t6, 57                               # ASCII code for character '9' - 57
li a7, 48                               # ASCII code for character '0' - 48


coreDictionaryAlgorithm:
add t2, t1, a0                          # Pass through 'myplaintext'/'cyphertext' string
lb a6, 0(t2)                            # Current character
beq a6, zero, endDictionary             # Check if string has been fully crossed

# Executes a procedure that identifies the nature of the char. and returns the appropriate offset
jal getCharOffset
add t5, a6, zero                        # t5 take the value of the current character
beq a1, t3, dictionaryIsNotALetter      # Check if a1= -1 it isn't a letter

# Check that it is a lowercase letter, if it is > 122 in 'getCharOffset' would be set a1 to -1.
bge a1, t4 dictLowercaseAlgorithm

# ==========================================================
# 'z' - letter + 'A'
# 122 - letter + 65   (es. 122 - 'A/65' + 65 =  'z/122')
# ==========================================================
dictUpperCaseAlgorithm:
neg t5, t5
addi t5, t5, 122
add t5, t5, a1
j nextCharDictionary

# ================================================
# 'Z' - letter + 'a'
# 90  - letter + 97
# ================================================
dictLowercaseAlgorithm:
neg t5, t5
addi t5, t5, 90
add t5, t5, a1
j nextCharDictionary

# ============================================================
#  Check if it's a number, otherwise it loads the same character
# ============================================================
dictionaryIsNotALetter:
blt t5, a7 nextCharDictionary           # If t5 is less of 0(48), it isn't a number
bgt t5, t6 nextCharDictionary           # If t5 is greater of 9(57), it isn't a number

# ====================================================================
```

```
#  cod(57) - number
# (es. cod(4)->ASCII(52)->ASCII(52-48)=4->ASCII(57-4)->ASCII(53)=5)
# ========================================================================
addi t5, t5, -48                        # t5 -= 48
sub t5, t6, t5                          # t5 <- ASCII(57 - t5)
j nextCharDictionary

# ======================================================
# Store the result in 'myplaintext'/'cyphertext' string
# ======================================================
nextCharDictionary:
sb t5, 0(t2)
addi t1, t1, 1                          # Increment loop couter used in 'coreDictionaryAlgorithm'
j coreDictionaryAlgorithm

endDictionary:
lw t0, 0(sp)
lw ra, 4(sp)
addi sp, sp, 8

jr  ra



#=============================================================================================================
#                               getCharOffset
#
#  Note: Returns the right offset to execute operations in 'coreCaesarAlgorithm' and
#        'coreDictionaryAlgorithm'.
#        It deals with classifying the characters in three ways:
#        1. if it is a lowercase letter it is assigned offset 97, ASCII 'a'.
#        2. if it is a UPPERCASE letter it is assigned offset 65, ASCII 'A'.
#        3. if it is not a letter, a1 is set to -1 as flag.
#
#  Paramether:
#
#
#  Return: Possible results...
#            1. a1 = 97-(lowercase)     2. a1 = 65-(UPPERCASE)    3. a1 = -1-(NOT a letter)
#
#=============================================================================================================
getCharOffset:
addi sp, sp, -8                         # Adjust stack for 2 items
sw ra, 4(sp)              # Return address for 'coreCaesarAlgorithm' or 'coreDictionaryAlgorithm'
sw t1, 0(sp)                            # Store current index use in 'coreCaesarAlgorithm'

jal isLowerCase

lw t1, 0(sp)
lw ra, 4(sp)
addi sp, sp, 8                          # Return stack

beq a1, zero, offsetCipherUpperCase  # Check if it is UPPERCASE (a1 = 0) of lowercase (a1 = 1)

offsetCipherLowerCase:
li a1, 97                               # Offset 97, ASCII 'a'
jr ra

offsetCipherUpperCase:
li a1, 65                               # Offset 65, ASCII 'A'
jr ra

# ============================================================================
# Check If a6 is a lowercase letter a1 is set to 1
# ============================================================================
isLowerCase:
li t0, 97                               # ASCII code for character 'a' - 97
li t1, 122                              # ASCII code for character 'z' - 122

blt a6, t0, isNotLowerCase              # Check that a6 < 97, it isn't lowercase letter
bgt a6, t1, isNotALetter                # Check that a6 > 122, it isn't a letter

li a1, 1                                # Used to execute 'offsetCipherLowerCase'
jr ra

# ============================================================================
#  Check if ASCII character is not a letter between 91 and 96 or less of 65.
#  Otherwise a6 is a UPPERCASE letter and is set a1 to 0.
```

```
# ================================================================================
isNotLowerCase:
li t0, 91                           # ASCII code for character '[' - 91
li t1, 65                           # ASCII code for character 'A' - 65

bge a6, t0, isNotALetter            # greater than 90(Z)
blt a6, t1, isNotALetter            # less than 65(A)

li a1, 0                            # Used to execute 'offsetCipherUpperCase'
jr ra

isNotALetter:
lw t1, 0(sp)
lw ra 4(sp)
addi sp, sp, 8

li a1, -1                           # It isn't a letter, set a1 = -1 as flag
jr ra




# ============================================================================
# Print 'myplaintext' string the first time in absolut and return to the main
# ============================================================================
print_plaintext:
add a0, a2, zero
li a7, 4
ecall

jr ra                       # Return to the main


# ============================================================================
# Print the outputs of each encryption and decryption separataed by 'newline'.
# Return to the chiper/decipher procedure
# ============================================================================
printCipherDecipher:
add a1, a0, zero            # use a1 to print the output because a0 is used by 'newline'

li a0, 10                   # 'newline' ASCII 10 code, used to differentiate the outputs
li a7, 11
ecall

add a0, a1, zero            # Print the output, 'myplaintext'/'cyphertext' string
li a7, 4
ecall

jr ra                       # Return to the chiper/decipher procedure

# ====================================
# ...end program
# ====================================
endMain:
```