Progetto Assembly RISC-V

Corso di Architetture degli Elaboratori

a.a 2021/2022

Messaggi in Codice

Autore:

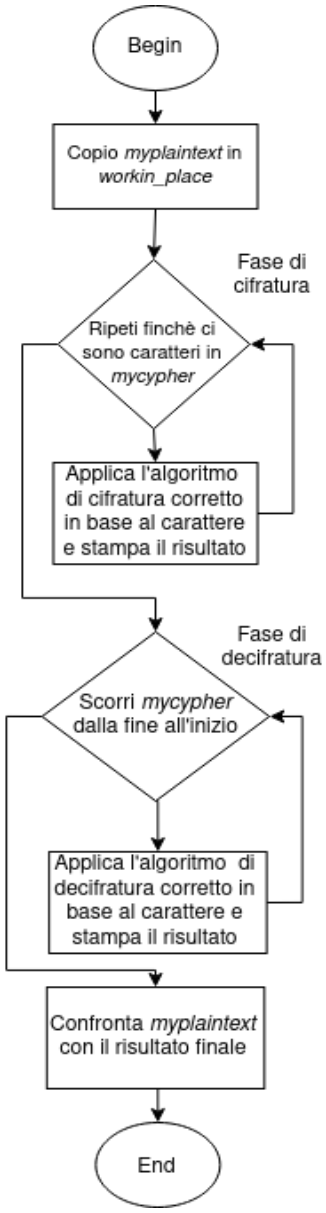
Lorenzo Bartolini
Matricola: **7073016**
Mail: *lorenzo.bartolini8@stud.unifi.it*

Consegnato in data

Abstract

Il progetto richiede di scrivere un programma in Assembly RISC-V che applichi in cascata una sequenza di algoritmi di cifratura ad una stringa fornita come parametro.
Vengono forniti due input: la stringa da cifrare (myplaintext) e la sequenza di algoritmi da usare (mycypher).
Una volta cifrato il messaggio è richiesto di decifrarlo per tornare alla stringa in chiaro da cui è stato inizializzato il programma.
Ho organizzato il programma con una procedura separata per ogni algoritmo e, dove necessario, ho separato la fase di cifratura da quella di decifratura. Ho unito il tutto all'interno della procedura Main che funge anche da entrypoint del programma.

Main



Inizialmente copio il contenuto di *myplaintext* in una nuova posizione di memoria, puntata dalla variabile *working_place*. Questo per dei problemi legati alla dimensione della stringa durante l'esecuzione dell'algoritmo ad Occorrenze, infatti in tutti gli altri casi la dimensione della stringa rimane invariata e non crea problemi ma in caso la dimensione varia e rischia di sovrascrivere porzioni di memoria contenenti altre informazioni. Non sovrascrivere *myplaintext s*erve anche come reference alla fine degli algoritmi per verificare il corretto funzionamento del programma.
Procedo scorrendo la stringa *mycypher* e applico, su *working_place*, per ogni carattere della stringa l'algoritmo che gli corrisponde. Dopo ogni algoritmo stampo a video il risultato parziale che ho appena calcolato.
Una volta terminato di scorrere la stringa significa che ho cifrato la stringa usando tutti gli algoritmi richiesti in cascata.
A questo punto scorro *mycypher* al contrario, tramite l'indice che ho usato precedentemente, e per ogni carattere applico la versione per la decifratura dell'algoritmo corrispondente. Come per la fase di cifratura stampo a video il risultato parziale.
Una volta terminata l'esecuzione stampo la stringa originale, ovvero il *myplaintext* intatto, e quella che ha subito gli algoritmi di cifratura e decifratura.

esempio di esecuzione usando due diversi algoritmi

```
Cifrato usando: Algoritmo di Cesare (A)
Uftu_Sfmbajpof_BEF_2021_2022

Cifrato usando: Algoritmo a Blocchi (B)
dry$kXuygpvu~rdQQKn>5A=dA<7A

Decifrato usando: Algoritmo a Blocchi (B)
Uftu_Sfmbajpof_BEF_2021_2022

Decifrato usando: Algoritmo di Cesare (A)
Test_Relazione_ADE_2021_2022



Decifrato: Test_Relazione_ADE_2021_2022
Originale: Test_Relazione_ADE_2021_2022
```

Algoritmo a Sostituzione

cifrario di Cesare

Questo cifrario sostituisce ogni lettera del testo in chiaro con una che si trova dopo un certo numero di posizioni all'interno dell'alfabeto. E' stato diviso in due sezioni, una per la cifratura e una per la decifratura.
In entrambi i casi scorro la stringa e per ogni carattere controllo che sia una lettera minuscola o maiuscola, negli altri casi proseguo il ciclo. Quando trovo una lettera per prima cosa sottraggo la base relativa al suo gruppo, ovvero sottraggo 65 per le lettere maiuscole e 97 per quelle minuscole. Successivamente sommo o sottraggo la chiave di sostituzione in base all'operazione che sto svolgendo, sommo per la cifratura e sottraggo per la decifratura. Infine sommo nuovamente la base che ho precedentemente tolto per tornare nel range di partenza.
Per entrambe le sezioni utilizzo i registri *a0* e *a1* per passare i parametri alla procedura: *a0* corrisponde alla stringa mentre *a1* è la chiave di sostituzione. Per questa procedura non è stato necessario l'utilizzo della memoria stack.

esempio usando la chiave di sostituzione = **1**

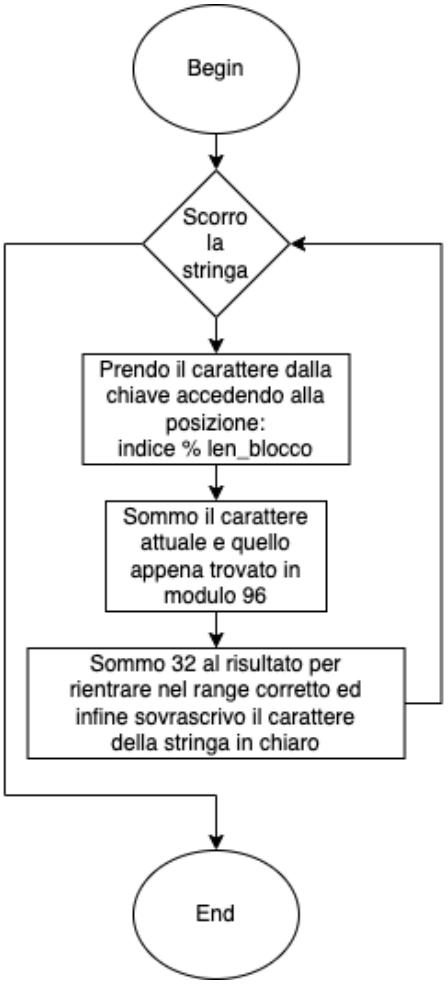
```
Cifrato usando: Algoritmo di Cesare (A)
uftu_Djgsbsjp_Dftbsf

Decifrato usando: Algoritmo di Cesare (A)
test_Cifrario_Cesare



Decifrato: test_Cifrario_Cesare
Originale: test_Cifrario_Cesare
```

Cifrario a blocchi



L'algoritmo consiste nello scorrere la stringa in chiaro utilizzando un indice *i* e sommando, in modulo 96, ad ogni carattere attraversato, il carattere della chiave corrispondente. Per calcolare il carattere del blocco si effettua l'operazione di modulo *i* % len_blocco. La lunghezza del blocco era stata precedentemente calcolata richiamando la procedura *str_len*.
Il cifrario a blocchi funziona simmetricamente per la fase di cifratura e decifratura, ovvero durante la prima fase vado a sommare la chiave con il carattere mentre durante la seconda la sottraggo.
Un'altra differenza tra cifratura e decifratura è che, nella seconda delle due, prima di effettuare il modulo 96 vado anche a sottrarre 32 insieme alla sottrazione del carattere della chiave.
Questi passaggi non sono sufficienti durante la decifratura in quanto sorgono alcuni problemi se il carattere di partenza ha codice maggiore uguale a 111, ovvero il carattere '*o*'. Ho osservato come, in questi casi, dopo aver sottratto la chiave e il valore immediato 32, il problema sorge quando il risultato di queste operazioni è minore di 32. Mi è bastato inserire un controllo aggiuntivo che, in caso di valore inferiore a 32, sommi 96.

```
Cifrato usando: Algoritmo a Blocchi (B)
#qx#kHxrwp~n~kG{{hrtn

Decifrato usando: Algoritmo a Blocchi (B)
test_Cifrario_Blocchi



Decifrato: test_Cifrario_Blocchi
Originale: test_Cifrario_Blocchi
```

*esempio con chiave "*OLE*"*

Inversione

Questo algoritmo inverte la stringa di partenza. Per farlo utilizza un indice che termina quando raggiunge la metà della stringa. Ogni carattere lo scambia con quello in posizione *len_stringa-i*. Nel caso in cui la lunghezza sia dispari terminerà alla posizione prima della metà infatti il carattere centrale non cambierà posizione.

```
Cifrato usando: Algoritmo a Inversione (E)
enoisrevnI_tset

Decifrato usando: Algoritmo a Inversione (E)
test_Inversione



Decifrato: test_Inversione
Originale: test_Inversione
```