Progetto Assembly MIPS per il Corso di Architetture degli Elaboratori – A.A. 2018/2019 –

Messaggi Cifrati

Nuova versione del documento, aggiornata il 14/3/2019. Modifiche rispetto alla precedente versione, rese necessarie per evitare problemi di saturazione della memoria:

- la lunghezza massima *k* della parola chiave è pari a **5 caratteri**, quindi $S="S_1 \cdots S_k"$ con $1 \leq k \leq 5$ (nella versione precedente del documento, *k* era uguale a 16);
- il file di testo "messaggio.txt" contenente il messaggio di testo da cifrare è composto da un massimo di **256 caratteri** (nella versione precedente del documento, il file poteva contenere fino a 1024 caratteri).

I codici sono un modo per alterare un messaggio per nasconderne il significato originale, e di solito richiedono una **parola chiave** per essere interpretati. I cifrari sono algoritmi applicati a un messaggio che nascondono o criptano le informazioni trasmesse. Questi cifrari vengono quindi **invertiti** per tradurre o decifrare il messaggio.

Lorenz cipher machine

Utilizzando QtSpim, scrivere e provare un programma in assembly MIPS che simuli la funzionalità di cifratura e decifratura di un messaggio di testo. In particolare il programma dovrà consentire di:

- **Cifrare un messaggio di testo** con una combinazione qualsiasi dei seguenti cifrari (algoritmi):
    - Algoritmo **A**: il codice ASCII standard su 8 bit di ciascun carattere del messaggio di testo viene modificato sommandoci una costante decimale *K=4*, modulo 256. Ovvero, se *cod(X)* è la codifica ascii standard decimale di un carattere *X* del messaggio, la cifratura di *X* corrisponderà a: *(cod(X)+K) mod 256*.
    - Algoritmo **B**: si applica l'Algoritmo A a tutti i caratteri del messaggio di testo che sono in posizione di indice pari (il primo carattere del messaggio avrà indice 0, quindi pari).
    - Algoritmo **C**: si applica l'Algoritmo A a tutti i caratteri del messaggio di testo che sono in posizione di indice dispari (il primo carattere del messaggio avrà indice 0, quindi pari).
    - Algoritmo **D**: il messaggio viene cifrato invertendo l'ordine dei caratteri del messaggio di testo. Ad esempio, se "ciao, prova a cifrarmi" è il contenuto del messaggio di testo da cifrare, allora la cifratura con l'Algoritmo D produrrà come risultato: "imrarfic a avorp ,oaic".

- Algoritmo **E**: a partire dal primo carattere del messaggio (quello alla posizione 0), il messaggio viene cifrato come una sequenza di stringhe separate da esattamente 1 spazio (codice ascii decimale 32) in cui ciascuna stringa ha la forma "x-p_1-...-p_k", dove x è la prima occorrenza di ciascun carattere presente nel messaggio, $p_1...p_k$ sono le posizioni in cui il carattere x appare nel messaggio (con $p_1<...<p_k$), ed in cui ciascuna posizione è preceduta dal carattere separatore '-' (per distinguere gli elementi della sequenza delle posizioni).
  Ad esempio, se "esempio di messaggio criptato -1" è il contenuto del messaggio di testo da cifrare, allora la cifratura con l'Algoritmo E produrrà come risultato: "e-0-2-12 s-1-13-14 m-3-11 p-4-24 i-5-9-18-23 o-6-19-28 -7-10-20-29 d-8 a-15-26 g-16-17 c-21 r-22 t-25-27 --30 1-31".
  Note sull'esempio:
  - Nella stringa " -7-10-20-29" il carattere in codifica è lo spazio (' ', codice ascii decimale 32), che appare nelle posizioni 7, 10, 20 e 29 del messaggio.
  - Nella stringa "--30" il carattere in codifica è '-' (il secondo carattere '-' è il carattere separatore fra gli elementi della sequenza), che appare nella posizione 30 del messaggio.
  - Nella stringa "1-31" il carattere in codifica è '1', che appare nella posizione 31 del messaggio.

.

Il messaggio di testo verrà cifrato utilizzando una **parola chiave** che definisce la **sequenza delle cifrature** da applicare al messaggio. La parola chiave è una stringa S="$S_1...S_k$" formata da al massimo 5 caratteri (quindi con $1 \leq k \leq 5$), in cui ciascun carattere S_i (con $1 \leq i \leq k$) corrisponde ad uno fra i caratteri 'A', 'B', 'C', 'D' oppure 'E', ed identifica il corrispondente cifrario da applicare al messaggio al passo i-mo. L'ordine delle cifrature è quindi stabilito dall'ordine in cui appaiono i caratteri nella stringa. A titolo di esempio, si riportano alcuni possibili parole chiave: "C", oppure "AEC", oppure "DEDD", oppure "EEEEE", ... Ad esempio, la cifratura del messaggio di testo con la parola chiave "AEC" determinerà l'applicazione dell'algoritmo A, poi dell'algoritmo E (sul messaggio già cifrato con A) ed infine dell'algoritmo C (sul messaggio già cifrato prima con A e poi con E).

- **Decifrare il messaggio di testo cifrato** utilizzando la parola chiave **invertita** $\hat{S}$= "$S_k... S_1$", ovvero **invertendo gli Algoritmi di Cifratura e richiamandoli in ordine inverso** (dall'ultimo algoritmo applicato al primo, ovvero da S_k a S_1).

Si assuma che il messaggio di testo da cifrare sia disponibile in un file di testo chiamato "messaggio.txt", composto da un massimo di 256 caratteri, e che la parola chiave S= "$S_1 \cdots S_k$" con cui cifrare il messaggio sia disponibile in un file di testo chiamato "chiave.txt". Si supponga inoltre che la chiave nel file "chiave.txt" sia sintatticamente corretta (composta solo dai caratteri 'A', 'B', 'C', 'D' oppure 'E') e che sia di lunghezza $1 \leq k \leq 5$.

Il programma dovrà leggere in input il messaggio da cifrare e la parola chiave, e produrre in output:

- un file di testo chiamato "messaggioCifrato.txt", contenente il messaggio cifrato utilizzando la parola chiave S="$S_1 \cdots S_k$";

- Un file di testo chiamato “messaggioDecifrato.txt”, contenente il messaggio decifrato a partire dal messaggio precedentemente cifrato utilizzando la parola chiave invertita $\hat{S}$= “S_k…S_1“ e gli Algoritmi di Cifratura invertiti. Nota: il messaggio decifrato correttamente dovrebbe ovviamente corrispondere al messaggio originale contenuto nel file “messaggio.txt”.

NOTE IMPORTANTI (LEGGERE CON ATTENZIONE):

- Tutti i file di testo (sia di input che di output) devono essere già stati creati (anche senza contenuto per i file di output) e devono risiedere nella stessa cartella in cui è presente l’eseguibile QtSpim.
- Seguire fedelmente tutte le specifiche dell’esercizio (incluse quelle relative ai nomi dei file e al formato del loro contenuto).
- Rendere il codice **modulare** utilizzando ove opportuno **chiamate a procedure** con l’istruzione **jal (jump and link)**, e **rispettando le convenzioni fra procedura chiamante/chiamata**. La modularità del codice ed il rispetto delle convenzioni saranno aspetti fondamentali per ottenere un’ottima valutazione del progetto. Si consiglia in particolare di **implementare ogni cifrario (ciascun algoritmo A-B-C-D-E, e le loro inversioni per la decifratura) come una procedura**.

Modalità di consegna del progetto:

- Per sostenere l'esame è necessario consegnare preventivamente il codice e una relazione (in pdf) sul progetto assegnato
  - Il progetto può essere svolto in gruppo, **max 3 persone per gruppo**
  - Ogni gruppo di lavoro dovrà consegnare un’unica relazione
  - Il codice consegnato deve essere funzionante sul simulatore QtSpim, anche se sviluppato e testato con altri simulatori (es, MARS)
- Un archivio contenente il **codice**, i **file di input** utilizzati per verificare il corretto funzionamento del programma, i relativi **file di output**, e la **relazione** dovrà essere caricato sul sito moodle del corso seguendo l'apposito link che verrà reso disponibile alla pagina del corso
  - Non vengono prese in considerazione altre modalità di consegna
- La scadenza esatta della consegna verrà resa nota di volta in volta
- Discussione e valutazione: la discussione degli elaborati avverrà contestualmente all’esame orale e prevede anche domande sull’assembly e su tutti gli argomenti di laboratorio trattati a lezione.

Struttura della relazione (in formato pdf):

- Info su autori e data di consegna
  - gli autori (e loro indirizzo e-mail – preferibilmente quello universitario@stud.unifi.it)
  - la data di consegna
- Descrizione della soluzione adottata, trattando principalmente i seguenti punti:
  - Descrizione ad alto livello dell’algoritmo (in linguaggio naturale, con flow-chart, in pseudo-linguaggio, etc.), delle strutture dati utilizzate (liste, vettori, etc.) e delle procedure utilizzate (argomenti, funzionalità svolte, risultati prodotti)
  - Uso dei registri e memoria (stack, piuttosto che memoria statica o dinamica)
  - Motivazione delle scelte implementative
- Test di corretto funzionamento
  - In questa sezione dovranno essere fornite le evidenze del corretto funzionamento del programma, ad esempio facendo vedere i valori di correttezza dei singoli sensori e dell’intero sistema (in base alle diverse politiche di aggregazione) in corrispondenza di una o più sottosequenze di input
- Codice MIPS assembly implementato e commentato in modo chiaro ed esauriente.
  - Nota: alla fine della relazione va inserito **TUTTO** il codice così come appare nel sorgente

Codice

- Ricordarsi di inserire anche nel codice gli autori (e loro indirizzo e-mail) e la data di consegna.
- Seguire fedelmente le convenzioni sull’uso della memoria e sulle procedure.
- Commentare il codice in modo significativo (move $s4,$s3 non significa solo che “copio il contenuto del registro $s3 in $s4”....).