RELAZIONE Progetto Architetture degli Elaboratori A.A 2021/2022

Autore:

Giulio Morandini, 7030209, giulio.morandini1@stud.unifi.it

Premessa:

Dati i problemi riscontrati durante l'utilizzo del programma con alcuni registri il progetto è stato steso usando la versione 2.0.1, pertanto per il corretto funzionamento del progetto è necessario scaricare ed usare la versione Ripes 2.0.1
(*https://github.com/mortbopet/Ripes/releases/tag/v2.0.1)*

Richiesta:

L'algoritmo legge il messaggio da cifrare e la chiave. Una serie di condizioni *beqz* rimandano all'etichetta esatta in base alla chiave fornita in input, e a loro volta queste etichette divise in CaseA, CaseB, CaseC, CaseD, CaseE, rimandano alle procedure che elaboreranno i vari algoritmi in modo differente. Ogni algoritmo di cifratura e decifratura utilizza lo stack pointer per la gestione delle variabili.

Cifrario di Cesare (Algoritmo A): **(scalando una posizione a destra)**

subc_while: Effettua un ciclo while sui caratteri della stringa. Carica il primo byte su *a7* e con un ***beqz*** verifica che il primo byte sia zero o meno; se corrisponde al valore zero, salta a ***subc_exit*** altrimenti effettua un secondo controllo sul valore presente in *a7* verificando se la lettera è maiuscola (valore ASCII compreso tra 65 e 90) o minuscola (valore ASCII compreso tra 97 e 122). Successivamente aggiunge una costante K=1 (mentre nell'algoritmo inverso la costante verrà sottratta) al corrispondente ASCII del carattere caricato in *a7* e sottrae 65 (corrisponde al valore ASCII della prima lettera maiuscola dell'alfabeto); se il risultato è positivo effettuo modulo 26 (lettere dell'alfabeto), altrimenti aggiungo 26. Dopodiché al valore calcolato viene sommato 65, e sostituiamo il nuovo carattere con quello che vi era in precedenza. Se, invece, la lettera è minuscola salta all'etichetta.
subc_L1: Esegue lo stesso procedimento dell'etichetta ***subc_while*** ma con i valori ASCII corrispondenti alle lettere minuscole dell'alfabeto (97 - 122).
subc_L2: Scrive il carattere codificato nel registro *a7*, incrementa di un'unità la variabile i e ritorna al ciclo iniziale.
subc_exit: Richiamata da ***subc_while*** quando la stringa è finita. Ritorna con un ***lw ra*** alla procedura chiamante e ripristina lo **sp**.

Cifrario a Blocchi (Algoritmo B):

blkc_while: Effettua un ciclo while sui caratteri della stringa. Carica il primo byte su *a5* e con un ***beqz*** verifica che il primo byte sia zero o meno; se corrisponde al valore zero salta a ***blkc_exit*** altrimenti viene caricato il primo byte della chiave sul registro *a4* e verifica che sia diversa da zero. Se è diversa da zero salta a ***blkc_L***

blkc_L: Effettua l'operazione {[(cod(bij)–32)+(cod(keyj)–32)]%96}+32. Sostituisce il carattere codificato con quello che vi era in precedenza ed infine scorre la stringa da criptare e la chiave.

blkc_exit: Richiamata da ***blkc_while*** quando la stringa è finita. Ritorna con un ***lw ra*** alla procedura chiamante e ripristina lo **sp**.

Cifratura Occorrenze (Algoritmo C):

Viene contata la lunghezza della stringa. Quando l'indice del vettore risulta uguale a zero aggiungiamo il null byte alla fine della stringa da criptare. A questo punto alloco lo spazio in memoria in modo tale da avere spazio sufficiente per criptare la stringa. Se l'indice del vettore supera la lunghezza del vettore che abbiamo allocato memoria (max 100 caratteri) il ciclo si interrompe. Aggiungo un contatore j che punta alla stringa criptata ed un flag da passare all'istruzione successiva dopo il confronto tra la variabile **occtext[i]** e **filter_text[j]**.

occc_wl_ftc: Legge il carattere con cui effettuare il confronto (onde evitare di inserire un carattere già presente nella stringa criptata), setta il contatore **j** ed il **flag**.

occc_wl_ftc_l1: Legge e salva in una variabile temporanea il carattere che dovrà essere eventualmente inserito nella stringa criptata a seconda del risultato del confronto. Salta a **occc_wl_ftc_l2** se la stringa non è terminata, altrimenti setta il flag da passare all'istruzione successiva e salta ad **occc_wl_ftc_l4**.

occc_wl_ftc_l2: Effettua il confronto tra i 2 caratteri, se sono diversi salta ad **occc_wl_ftc_l3**, altrimenti se gli operandi sono uguali la sottrazione da come risultato zero per cui il flag viene impostato uguale a zero e salta ad **occc_wl_ftc_l4**.

occc_wl_ftc_l3: L'etichetta viene chiamata da **occc_wl_ftc_l2** quando i 2 caratteri sono diversi (flag=1). Per cui aggiungo il nuovo carattere incontrato per la prima volta e ritorno ad **occc_wl_ftc_l1**.

occc_wl_ftc_l4: L'etichetta viene chiamata da **occc_wl_ftc_l1** e **occc_wl_ftc_l3**, per cui quando si verificano le 2 condizioni: 1. stringa non terminata 2. flag=0 (caratteri uguali). Se si verifica la seconda condizione salta a **occc_wl_ftc_l5**. Il carattere già trovato non dovrà essere inserito nuovamente nella stringa finale, pertanto si avanza di 1 byte nella stringa da criptare. Se invece, si verifica la prima condizione per cui il flag=1, scriviamo il nuovo carattere incontrato per la prima volta nella stringa finale.

occc_wl_cipher: Adesso inizia la vera e propria fase del processo di crittografia della stinga convertendo il numero integer nel corrispondente ASCII, scrive tale posizione nella stringa criptata e scorre nuovamente la stringa da criptare.

occc_wl_c_L1: Se la stringa criptata è terminata salta a **occc_wl_c_L6**, altrimenti esegue il confronto. Se il carattere non è presente nella stringa salta a **occc_wl_c_L5**, altrimenti la posizione sarà seguita da '-'.

occc_wl_c_L2: La funzione ha il compito di scrivere la posizione corrispondente del carattere. Se essa è composta da più di un byte, e quindi è maggiore di 9, salta a **occc_wl_c_L3**, altrimenti converto la posizione nel corrispondente ASCII sommandogli 48, ovvero il corrispondente del primo carattere numerico in ASCII e salto a **occc_wl_c_L4**.

occc_wl_c_L3; Viene chiamato da **occc_wl_c_L2**, quando la posizione del carattere è

formata da più di 1 byte, dunque da più cifre. Effettua quindi una divisione per 10 per crittografare la posizione una cifra alla volta sulla quale poi calcolare il modulo 10 e sommargli 48 (esempio 12/10= 1,2). Richiama **occc_wl_c_L2**.
occc_wl_c_L4: Ottiene, effettivamente, la stringa finale criptata. Se non ha più lo stesso carattere all'interno della stringa salta a **occc_wl_c_L5**, altrimenti si procede.
occc_wl_c_L5: Viene chiamata da **occc_wl_c_L1** e **occc_wl_c_L4** quando si verifica la stessa condizione, ovvero la mancata presenza dello stesso carattere. Dunque scorre la stringa e torna a **occc_wl_c_L1**.
occc_wl_c_L6: Viene chiamata da **occc_wl_c_L1**. Aggiunge lo spazio al byte successivo quando finisco le occorrenze del carattere. Scorro la stringa e ritorno al ciclo while iniziale per continuare il processo di crittografia.
occc_ret: Ritorna con un **lw ra** alla procedura chiamante e ripristina lo **sp**.

Dizionario (Algoritmo D):

dicc_while: Effettua un ciclo while sui caratteri della stringa. Carica il primo byte su *a5* e con un ***beqz*** verifica che il primo byte sia zero o meno; se corrisponde al valore zero, salta a ***dicc_exit*** altrimenti inizia la mappatura suddivisa in vari casi:
dicc_L1: Verifica che il carattere sia una lettera maiuscola. Successivamente copia *a5* in *a6* in cui effettuiamo varie operazioni per ottenere l'equivalente minuscolo dell'alfabeto inverso: sottraggo 65 (corrisponde al valore ASCII della prima lettera maiuscola dell'alfabeto). Il risultato viene a sua volta sottratto di 25 per ottenere il suo inverso ed infine viene sommato 97 (corrisponde al valore ASCII della prima lettera minuscola dell'alfabeto) per ottenere l'equivalente minuscolo.
dicc_L2: Stesso procedimento di *dicc_L1* ma con i valori ASCII delle lettere minuscole ai quali viene sommato 65 per ottenere l'equivalente maiuscolo.
dicc_L3: Procedimento svolto con la stessa idea con la quale sono state svolte le altre 2 etichette, ma con i valori ASCII corrispondenti ai numeri (48 - 57) ed eseguendo la funzione ASCII(cod(9)-num).
dicc_L4: I simboli rimangono invariati. Dunque ci limitiamo a copiare *a5* in *a6.*
dicc_L5: Scrive il carattere codificato in *a6*. Incrementa di un'unità la variabile i e torna al ciclo while.
dicc_exit: Richiamata da **dicc_*while*** quando la stringa è finita. Ritorna con un ***lw ra*** alla procedura chiamante e ripristina lo *sp.*

Inversione (Algoritmo E):

invc_while: Effettua un ciclo while sui caratteri della stringa. Carica il primo byte su *a5* e con un ***beqz*** si procura la lunghezza della stringa. Quando raggiunge la fine salta a *invc_L0*
invc_L0: Due caratteri vengono scambiati nella loro posizione nella stringa di testo indicata da *a7* e *a6.* Per questo algoritmo è necessario avere la posizione del carattere mediano in modo da utilizzare 2 puntatori che puntano rispettivamente all'inizio e alla fine della stringa. Dunque controlla se la stringa è formata da un numero negativo di caratteri; se negativo aggiungo 1 in *a4.* ***srai a5, a5, 1:*** Sposta 1 bit a destra ed effettua la divisione della stringa in 2 parti.
invc_L1while: Incrementa il puntatore che punta al primo carattere della stringa e decrementa il puntatore che punta all'ultimo carattere della stringa fino a quando entrambi i puntatori raggiungono il centro della stringa.
invc_exit: Richiamata da *invc_L1while* quando la stringa è finita. Ritorna con un **lw ra** alla procedura chiamante e ripristina lo **sp**.

Algoritmi di decifratura:

Dopo aver scritto la stringa criptata, il programma legge la chiave di decriptazione da destra verso sinistra e applica a tale stringa gli algoritmi inversi corrispondenti tramite una condizione **beq.** Infine applicando tutti gli algoritmi inversi otteniamo la stringa di partenza e il programma termina.

Algoritmo A:

L'algoritmo A inverso opera esattamente come l'AlgoritmoA.

Algoritmo B:

La procedura dell'algoritmo B inverso è uguale a quella dell'algoritmo B con la differenza che la chiave del blocco cifrario viene negata; se la seguente operazione è negativa somma 96, altrimenti scrive il carattere, scorre la stringa da criptare e la chiave.

Algoritmo C:

occdec_while: Legge la stringa criptata con l'algoritmo C e carica il primo carattere controllando se esso sia il null byte, in tal caso termina e salta a **occdec_L7**. In caso contrario si scorre di 1.
occdec_L1: Controlla la posizione del carattere, se non corrisponde allo spazio salta a **occdec_L2** altrimenti salta a **occdec_L6** che ritorna al ciclo while iniziale per continuare la decrittografia.
occdec_L2: Controlla il byte successivo, se esso è un trattino salta a ***occdec_L5*** che continua a scorrere la stringa da decriptare. Converte la posizione dal valore ASCII al corrispettivo integer. Altrimenti se il byte letto non è né uno spazio, né un trattino ma una cifra, salta all'etichetta **occdec_L3**. Se ha più di una cifra si moltiplica per 10 e si aggiunge la cifra appena letta ad essa (dopo aver convertito da ASCII ad integer) e si effettua nuovamente il ciclo incrementando la posizione di un byte. Se la nuova posizione letta è minore o uguale di 47 o maggiore di 57 significa che il carattere non è presente una seconda volta nella stringa e salta a **occdec_L4** nel quale si ottiene la posizione corretta sottraendo il byte aggiunto prima.
occdec_L8: Effettuo l'ultimo ciclo while in cui viene restituita la stringa inversa ed effettuo nuovamente il ciclo fino a quando non termina la stringa.
occdec_exit: Ritorna con un **lw ra** alla procedura chiamante e ripristina lo **sp**.

Algoritmo D:

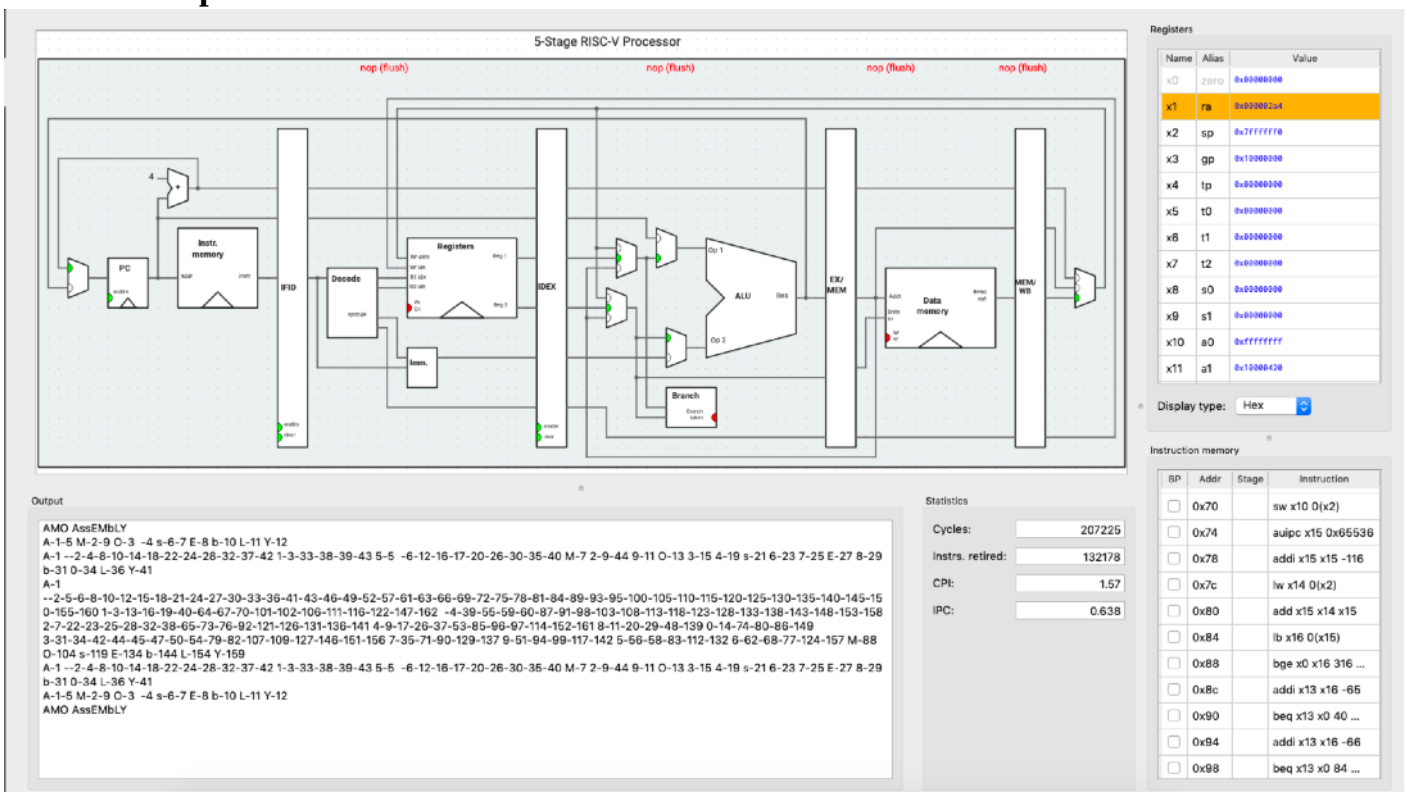
L'algoritmo D inverso opera esattamente come l'AlgoritmoD.

Algoritmo E:

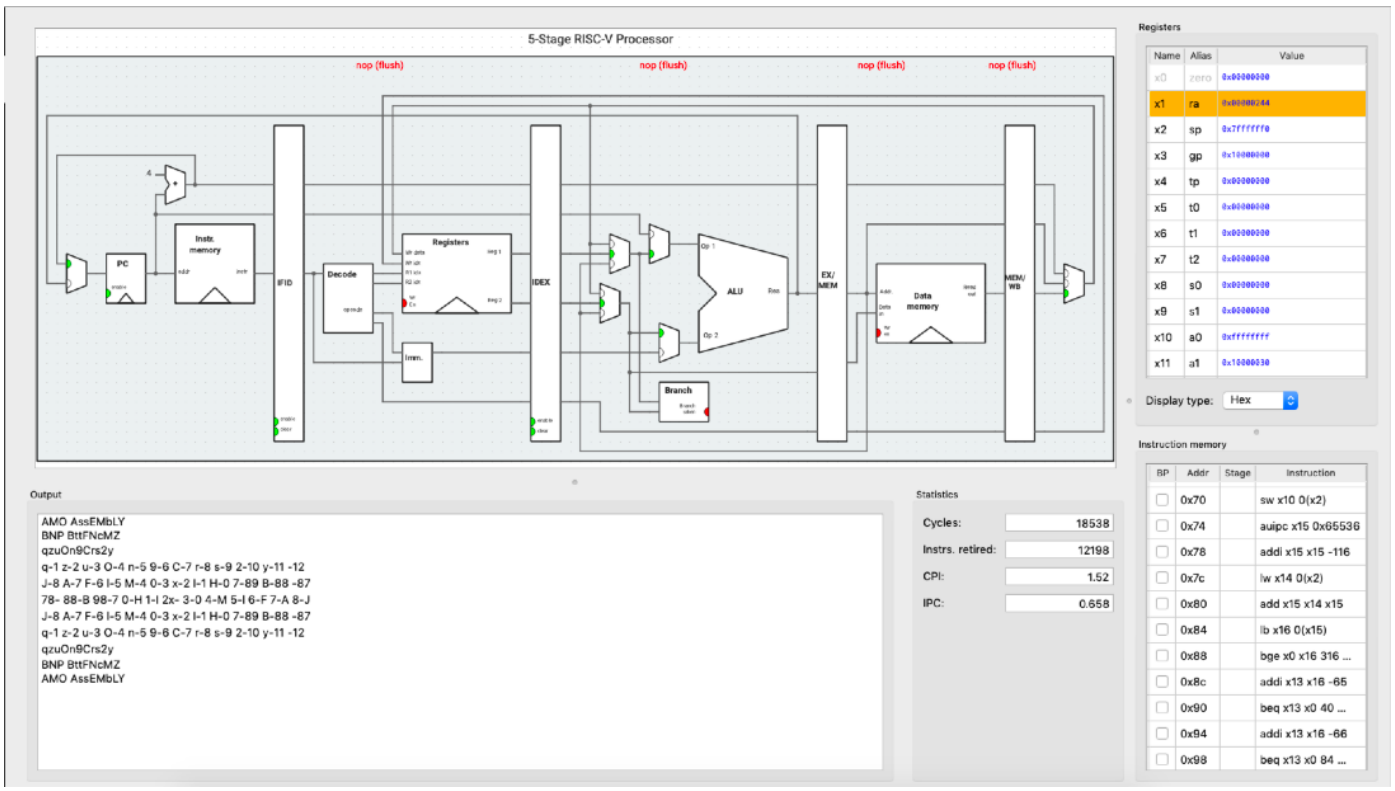
L'algoritmo E inverso opera esattamente come l'AlgoritmoE.

Test di corretto funzionamento:

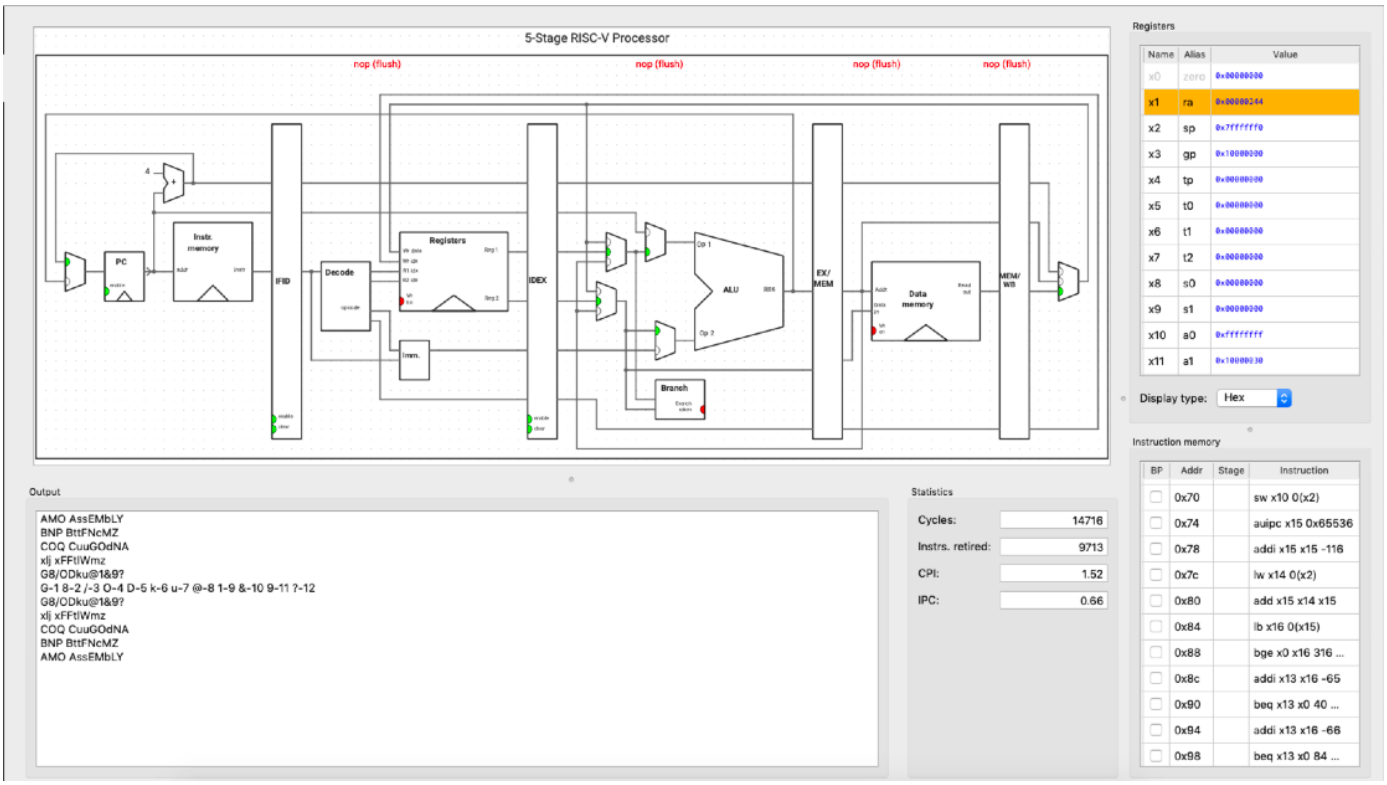
Esempio di funzionamento con chiave CCC:



Esempio funzionamento con chiave ABCDE:



Esempio funzionamento con chiave AADBC:



Codice RISC-V:

Per i dettagli della stesura del codice RISC-V, si rimanda direttamente al codice.