RELAZIONE PROGETTO ARCHITETTURE DEGLI ELABORATORI ASSEMBLY RISC-V

MATRICOLA: 7011579
EMAIL: jonathan.colombo@stud.unifi.it
DATA DI CONSEGNA: 10/09/2022

DESCRIZIONE AD ALTO LIVELLO/LINGUAGGIO NATURALE:

Nota: Ogni algoritmo è stato scritto in Java per comprendere al meglio la logica ad alto livello e per fornire una documentazione più accurata.
Nella classe main vengono inizializzate le variabili utilizzate nelle varie funzioni e viene gestita la chiave con i vari controlli per gestire le corrispettive chiamate alle funzioni. (Esempio: se viene trovato il carattere A, allora viene richiamata la funzione di cifratura A ecc...).

Analogamente viene effettuata la stessa procedura per la decifratura scorrendo il vettore di caratteri partendo dall'ultimo elemento.

Algoritmo A:
Per implementare l'algoritmo è necessario dichiarare una variabile per lo shiftamento delle posizioni (un intero positivo o negativo) e avere il vettore dei caratteri da cifrare.
Il vettore viene fatto scorrere dalla prima all'ultima posizione con un ciclo for, e ogni elemento viene controllato sulla base dei range impostati all'interno degli if e viene applicata la funzione di cifratura di Cesare.

Se lo shiftamento inserito è negativo basta renderlo positivo, fare il modulo con 26 (numero delle lettere) e sottrarre il risultato sempre con 26.
Per la decodifica basta moltiplicare per -1 lo shiftamento e riapplicare la solita procedura.

Codice Java per l'algoritmo A:

```
public static char[] AlgorithmA(char[] msg, int shift)

{

  for (int i = 0; i < msg.length; i++)
  {
    if (msg[i] >= 65 && msg[i] <= 90)
     {
      cod = 65 + ((msg[i] + shift - 65) % 26);
      msg[i] = (char) cod;
     }

    if (msg[i] >= 97 && msg[i] <= 122)
     {
      cod = 97 + ((msg[i] + shift - 97) % 26);
      msg[i] = (char) cod;
     }

  }

  System.out.println("Cod A : " + String.valueOf(msg));
  return msg;
```

```
 }
```

Algoritmo B:
Per implementare l'algoritmo è necessario calcolare il numero di blocchi (lunghezza del messaggio diviso la lunghezza della chiave).
Ogni carattere del testo (suddiviso in blocchi) viene sommato alla codifica ASCII di un carattere della chiave con questo metodo.

Codice Java per l'algoritmo B:

```
public static char[] AlgorithmB(char[] msg, char[] key)
{

        int nb = msg.length/key.length;
        int mod_nb = msg.length % key.length;
        int length_key = key.length;

        for(int i=0; i< (key.length * nb); i++)
        {
              msg[i] = (char) (32 + ((msg[i] + key[i%key.length]) % 96));
        }


        for(int i=0; i< mod_nb; i++)
        {
          int indice = (length_key * nb) + i;
          msg[indice] = (char) (char) (32 + ((msg[indice] + key[i%key.length]) % 96));
        }


        System.out.println("Cod B : " + String.valueOf(msg));
        return msg;
 }
```

Per quanto riguarda la decodifica ogni carattere viene decoifrato sommando più 64, sottraendo il key[i%key.length]) (che sarebbe il cod(keyij) poi facendo modulo 96 e controllando se il valore ASCII del carattere cifrato è maggiore di 32 lo salvo altrimenti si somma più 96.

Algoritmo C:
Per l'algoritmo C è necessario avere il messaggio da cifrare e il carattere spazio salvato in una variabile.
Si scorre tutta la lista di caratteri e si controlla ogni carattere e la sua posizione all'interno del vettore. Si concatena il carattere da cifrare con il carattere "-" e successivamente carichiamo la posizione in cui si trova.
Nel momento in cui la variabile non si trova più in altre posizioni della stringa, si concatena dopo le posizioni del carattere lo spazio.

Codice Java per l'algoritmo C:

```
public static char[] CodC(char[] msg)
```

```
{
  String tmp = "";
  for(int i = 0; i < msg.length; i++)
  {
    if (tmp.indexOf(msg[i]) == -1)
    {
      tmp = tmp + " " + msg[i];
      for(int z = 0; z < msg.length; z++)
      {
        if (msg[i] == msg[z])
        {
          tmp = tmp + "-" + z;
        }
      }
    }
  }
```

Algoritmo D:

Per l'algoritmo D è necessario scorrere la lista dei caratteri e controllare uno ad uno il valore ASCII.

Se il numero è una lettera minuscola viene applicata questa formula per trasformarla nella corrispettiva maiuscola in ordine inverso -> characterAscii = (26 - (msg[i] - 97)) + 64.

Se maiuscola, la lettera viene trasformata nella corrispettiva lettera minuscola in ordine inverso -> characterAscii = ((26 - (msg[i] - 65)) + 96).

Se un numero si applica -> characterAscii -> ((9 - (msg[i] - 48)) + 48); altrimenti se non è nessuno di questi caratteri si continua a scorrere la lista.

Per quanto riguarda la decifratura è necessario riapplicare il solito algoritmo.

Codice Java per l'algoritmo D:

```
public static char[] CodD(char[] msg)
 {
        for (int i=0;i<msg.length;i++)
        {
```

```
            if (msg[i] >= 65 && msg[i] < 91)
            {
                msg[i] = (char) ((26 - (msg[i] - 65)) + 96);
            }
            else if (msg[i] >= 97 && msg[i] <= 122)
            {
                msg[i] = (char) ((26 - (msg[i] - 97)) + 64) ;
            }
            else if (msg[i] > 48 && msg[i] < 58)
            {
                msg[i] = (char) ((9 - (msg[i] - 48)) + 48);
            }
        }

        System.out.println("Encrypt D : " + String.valueOf(msg));
        return msg;
    }
```

Algoritmo E:

Per l'algoritmo E è necessario salvarsi l'indice dell'elemento in testa e l'indice dell'elemento in coda di cui vogliamo effettuare lo scambio.

Si scorre la lista, si effettua lo scambio tra l'elemento in testa con quello in coda tramite una variabile temporanea di appoggio e si aggiornano gli indici aggiornando l'elemento in testa e quello in coda di cui vogliamo effettuare lo scambio.
L'algoritmo termina quando l'indice dell'elemento di testa risulta maggiore dell'indice dell'elemento in coda perchè significa che abbiamo superato la metà e che abbiamo scambiato tutti gli elementi della lista in testa con tutti quelli in coda.

Per la decifratura, basta richiamare nuovamente il solito algoritmo utilizzato per la cifratura.

Codice Java per l'algoritmo E:

```
public static char[] algorithmE(char[] msg){
        int begin = 0;
        int end = msg.length-1;
        char temp;
        while(end>begin)
        {
           temp = msg[begin];
```

```
            msg[begin]=msg[end];
            msg[end] = temp;
            end--;
            begin++;
        }

        System.out.println("Encrypt E : " + String.valueOf(msg));
        return msg;
    }
```

DESCRIZIONE BASSO LIVELLO E UTILIZZO REGISTRI E MEMORIA:

Main:

Nel main viene caricata la testa della stringa da cifrare (il myplaintext) e viene fatta stampare a video seguito dalla stampa del carattere a capo.

Viene caricata la chiave nel registro a1 e il testo nel registro a0.

Vengono caricate le varie lettere in valore ASCII (A,B,C,D, E) che corrispondono alle etichette dei vari algoritmi nei registri s1...s5.

Successivamente ci sono varie jump and link sulle etichette di cifratura, decifratura e fine programma.

Nella funzione controlCharacter si controlla la lettera da analizzare e se A, B, C, D o E e si salta al corrispettivo algoritmo e nello stack si salva l'indice dell'ultima lettera analizzata, altrimenti non ci sono lettere da analizzare e il programma termina.

cryptA:

Viene caricato il carattere da cifrare in t0, viene allocata la memoria e si salva nello stack l'indice della lettera da analizzare.

Viene fatto il controllo per permettere di saltare all'etichetta dell'algoritmo indicato dalla lettera della chiave.

Viene allocato lo spazio nello stack per salvare l'indirizzo di ritorno perché si dovrà saltare ad una procedura di sostituzione dei caratteri.

Eseguita la procedura di sostituzione, recupero l'indirizzo di ritorno dallo stack e ripristino quest'ultimo. Successivamente stampo la stringa, il carattere di fine stringa e carico nello stack l'indice dell'ultima della lettera della chiave che avevo inizializzato nel main.

Nella procedura di sostituzione vengono caricati in alcuni registri temporanei, i vari "estremi" delle lettere dell'alfabeto, sia minuscole che maiuscole per effettuare i vari controlli sui valori del testo da cifrare.

Nel caso in cui la lettera analizzata del testo da cifrare non fosse compresa nel range delle lettere maiuscole/minuscole, continuo a scorrere la lista.

Altrimenti, cifro la lettera utilizzando la formula del cifrario di Cesare.

Nel caso il valore ASCII della variabile di shiftamento fosse negativa, rendo positiva la variabile, applico la funzione modulo con il numero delle lettere e sottraggo il risultato di nuovo con

quest'ultimo valore.

Infine i registri utilizzati vengono resettati e si ritorna a controllare il prossimo carattere della chiave.

cryptB:

Vengono inizializzati dei registri temporanei a 0 e caricato il blockKey in t4 e t3 men-

tre il testo da cifrare in t5.

Viene calcolata la lunghezza totale della stringa incrementando i vari indici di block-

Key e myplaintext.

Successivamente viene caricato di volta in volta un carattere del myplaintext da cifra-

re e viene applicato l'algoritmo di cifratura B.

Viene salvato il carattere di cifratura ottenuto, viene controllato l'indice della prossima lettera e si riesegue la procedura fino a che non siamo arrivati a fine stringa.

A fine procedura vengono resettati i registri utilizzati per l'algoritmo B.

cryptC:

Per l'algoritmo C vengono calcolate le occorrenze e i valori già analizzati vengono indicati con -1.

Ogni carattere viene caricato e viene effettuato il controllo se siamo arrivati a fine stringa.

Si controlla anche se il carattere è già stato analizzato e si effettua una procedura per evitare la ripetizione e poi si scorre la lista incrementando l'indice.

Per evitare la ripetizione, viene caricato il carattere della stringa corrente, si carica il carattere – come carattere successivo e il carattere 10 e si effettua il controllo se è maggiore o uguale a 10 e in tal caso divido e sommo 48 per ottenere la cifra delle decine. Riapplico la solita procedura per ottenere anche le unità.

(L'algoritmo si poteva applicare per le centinaia e anche le migliaia. In questo caso

per cifrature con numero di lettere troppo elevate non funziona).

Nel caso in cui il carattere fosse minore di 10 viene aggiunto il valore della codifica ASCII.

Viene allocata la memoria necessaria per salvare le posizioni e i registri utilizzando lo stack e viene caricato il parametro delle occorrenze e l'indirizzo di ritorno.Successivamente viene riequilibrato lo stack e si effettua nuovamente la procedura

dall’inizio caricando il prossimo carattere da cifrare.
In fine vengono resettati tutti i registri utilizzati per la procedura e viene caricata la testa della stringa cifrata nel registro a0.

cryptD:
Per l’algoritmo D vengono salvati tutti gli estremi dove sono presenti i vari caratteri in registri temporanei. Viene caricato il carattere da cifrare e si effettuano vari controlli e azioni:
1) Se il valore in ASCII del carattere è uguale a zero siamo arrivati a fine stringa e possiamo terminare il loop.
2) Se il valore in ASCII del carattere è minore di 48 e diverso da zero si scorre la lista di 1.
3) Se sono false le condizioni precedenti ma il valore in ASCII del carattere è minore di 58, si aggiunge -48, 9, 48, si salva il carattere cifrato e si scorre la lista di 1.
4) Se sono false le condizioni precedenti ma il valore in ASCII del carattere è minore di 65 si scorre la lista di 1.
5) Se sono false le condizioni precedenti ma il valore in ASCII del carattere è minore di 91 si aggiunge -65, 26, 96, si salva il carattere cifrato e si scorre la lista di 1.
6) Se sono false le condizioni precedenti ma il valore in ASCII del carattere è minore di 97 si scorre la lista di 1.
7) Se sono false le condizioni precedenti ma il valore in ASCII del carattere è minore di 123 si aggiunge -97, 26, 64, si salva il carattere cifrato e si scorre la lista di 1.
Una volta terminato l’algoritmo si stampa semplicemente la cifratura ottenuta a video.

decryptA:
Viene moltiplicato per -1 il testo da cifrare e si riapplica la stessa procedura della EncryptA.

decryptB:
Vengono inizializzati vari valori che mi serviranno per la decifratura e carico il blockKey in a1.

Calcolo sempre la lunghezza della stringa incrementando degli indici finché non trovo il carattere zero di fine stringa.

Carico il carattere corrente e faccio sempre il controllo di fine stringa.

Nel caso non sia uno zero, aggiungo -32 e controllo il risultato se è maggiore di 32 e nel caso non lo sia sommo a quest'ultimo 96.

Viceversa sottraggo a quest'ultimo -32 , poi sommo con il valore del carattere corrente caricato all'inizio della procedura, risommo con 32 e controllo se il risultato è maggiore di zero.

Se quest'ultimo è maggiore di zero aggiungo -160 altrimenti controllo subito se è più grande di 32.

Se non lo è aggiungo 96 altrimenti controllo se è maggiore di zero e se non è vero sommo -96 altrimenti controllo se il numero è minore di 128.

Nel caso sia vero carico il carattere cifrato e scorro la lista, altrimenti sommo – 96, carico il carattere cifrato e scorro la lista.

decryptC:

Vengono caricati il valore 45 e 32 in dei registri temporanei e altri registri vengono inizializzati a zero per evitare valori sporchi provenienti da altri algoritmi di cifratura. Il registro a3 conterrà la stringa decifrata e viene calcolata con un fattore moltiplicato uguale a 3 per evitare sovrapposizioni di dati tra registri. Viene recuperato il carattere da decifrare e si effettua il controllo se è zero (quindi carattere di fine stringa) e in tal caso è concluso l'algoritmo altrimenti si prosegue. Viene incrementata una variabile contatore e viene caricato il carattere successivo. Nel caso il carattere successivo sia un trattino in codice ascii, vuol dire il carattere successivo indicherà la posizione del carattere di cui sto contando le occorrenze e quindi salterò alla procedura countLoop. Altrimenti, incremento di uno la variabile contatore e riapplico la procedura per recuperare il prossimo carattere. Nella procedura countLoop incrementerò il contatore di 1, caricherò il carattere corrente in un registro temporaneo e controllerò se è zero (carattere di fine stringa). Nel caso quest'ultima condizione sia vera vuol dire che ho concluso l'algoritmo altrimenti proseguo. Nel registro t2 sarà l'indirizzo dove andrò a memorizzare il valore della occorrenza e sommando -48. Carico il byte successivo in un altro registro e se questo è uguale a 45 o a 32 o a 0 ho finito di effettuare la procedura perché ho terminato la serie dell'occorrenza, altrimenti ricalcolo l'indirizzo di memoria dove andare a memorizzare l'occorrenza con un numero di posizioni maggiore di 9, vado a convertire i due caratteri dell'occorrenza da un codice

ASCII ad un codice numerico ottenendo la decina e l’unità dell’occorrenza. Il caratte-

re decifrato viene caricato nella posizione ricavata e si continua a ripetere la proce-

dura.

DecryptStringD

Si riapplica la stessa procedura della EncryptD.

cryptE e decryptE:

Richiamano la stessa funzionalità ma con messaggio di stampa a video diverso per visualizzare meglio l’output.

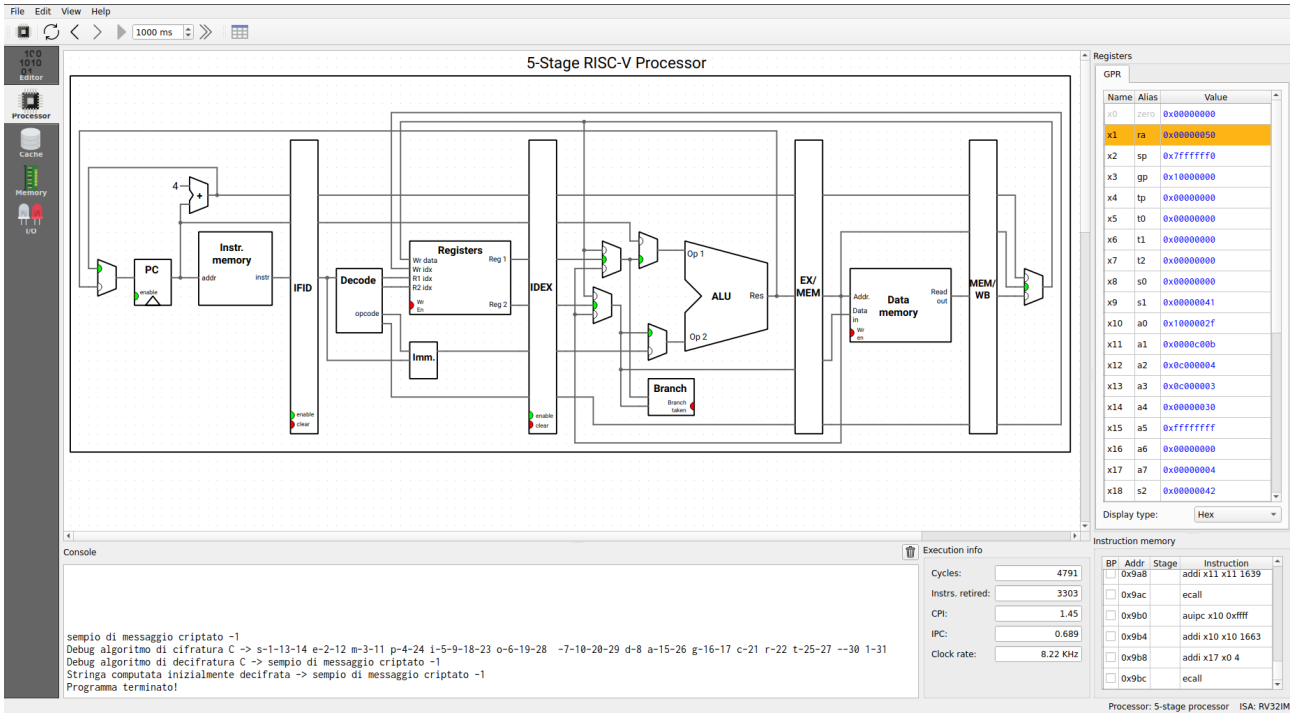
Vengono inizializzati due registri a zero che svolgono il ruolo di contatore. Il primo serve a uscire da un ciclo attraverso una opportuna condizione e l’altro serve per calcolare la lunghezza massima del testo da cifrare.

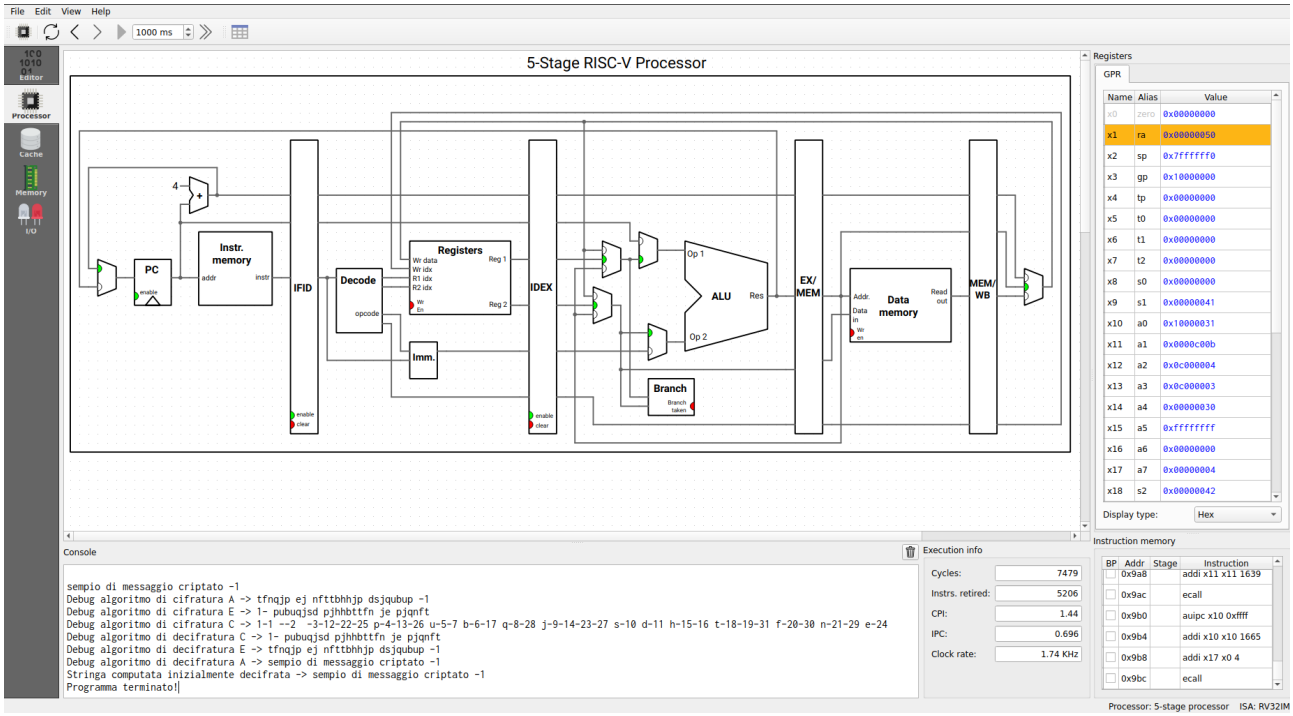
Successivamente si procede all’algoritmo di inversione, salvando e scambiando di volta in volta attraverso degli indici, l’elemento di testa e l’elemento di coda.

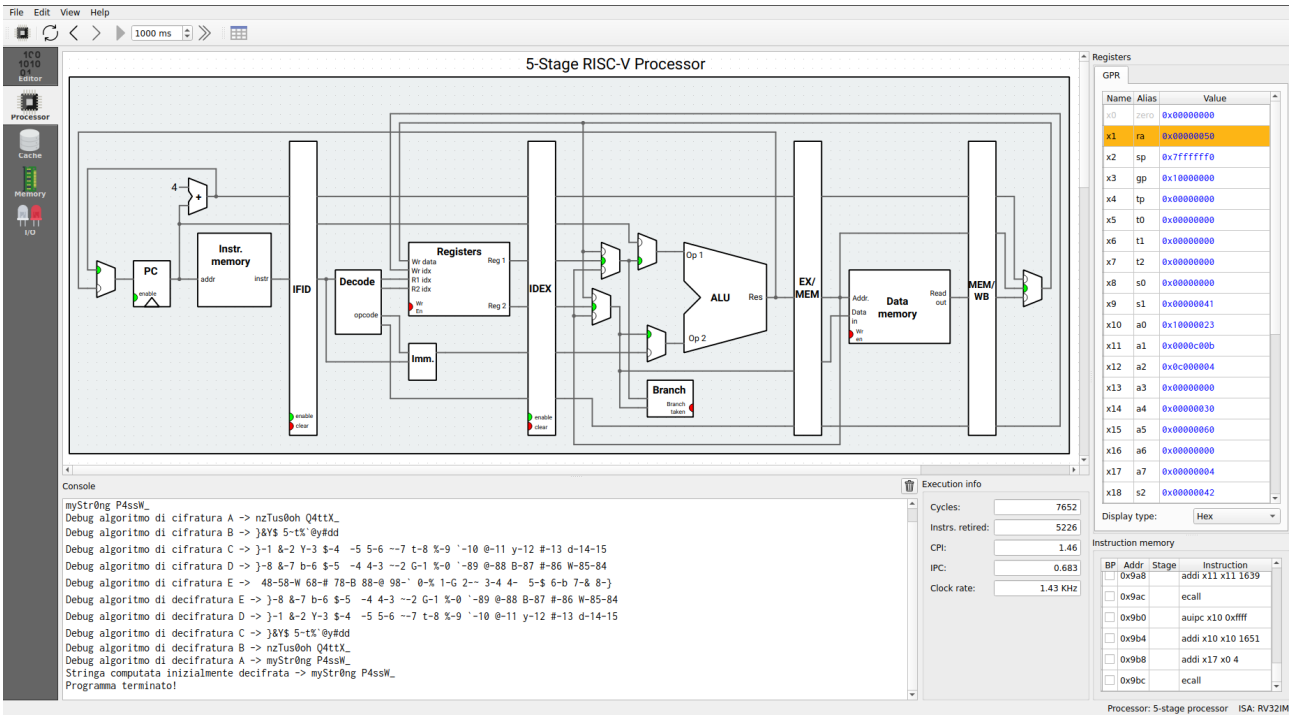
Si effettua un controllo sugli indici di testa e coda e se viene superatà la metà della lunghezza del testo da cifrare, l’algoritmo termina e stampa a video il testo cifrato.

NOTE: Dove ritenuto necessario, è stato utilizzato lo stack per salvare i valori dei registri di ritorno oppure i parametri da utilizzare su determinate funzioni.

TEST DI CORRETTO FUNZIONAMENTO (SCREENSHOTS):




File Edit View Help
1000 ms
Editor
Processor
Cache
Memory
I/O
5-Stage RISC-V Processor
4
PC
enable
Instr. memory
addr
instr
IFID
enable
clear
Decode
opcode
Registers
Wr data
Wr idx
R1 idx
R2 idx
Wr En
Reg 1
Reg 2
Imm.
IDEX
enable
clear
Op 1
Op 2
ALU
Res
Branch
Branch taken
EX/ MEM
Addr.
Data in
Wr en
Data memory
Read out
MEM/ WB
Registers
GPR
Name
Alias
Value
x0 zero 0x00000000
x1 ra 0x00000050
x2 sp 0x7ffffff0
x3 gp 0x10000000
x4 tp 0x00000000
x5 t0 0x00000000
x6 t1 0x00000000
x7 t2 0x00000000
x8 s0 0x00000000
x9 s1 0x00000041
x10 a0 0x10000031
x11 a1 0x0000c00b
x12 a2 0x0c000004
x13 a3 0x0c000003
x14 a4 0x00000030
x15 a5 0xffffffff
x16 a6 0x00000000
x17 a7 0x00000004
x18 s2 0x00000042
Display type:
Hex
Instruction memory
BP
Addr
Stage
Instruction
0x9a8 addi x11 x11 1639
0x9ac ecall
0x9b0 auipc x10 0xffff
0x9b4 addi x10 x10 1665
0x9b8 addi x17 x0 4
0x9bc ecall
Console
sempio di messaggio criptato -1
Debug algoritmo di cifratura A -> tfnqjp ej nfttbhhjp dsjqubup -1
Debug algoritmo di cifratura E -> 1- pubuqjsd pjhhbttfn je pjqnft
Debug algoritmo di cifratura C -> 1-1 --2 -3-12-22-25 p-4-13-26 u-5-7 b-6-17 q-8-28 j-9-14-23-27 s-10 d-11 h-15-16 t-18-19-31 f-20-30 n-21-29 e-24
Debug algoritmo di decifratura C -> 1- pubuqjsd pjhhbttfn je pjqnft
Debug algoritmo di decifratura E -> tfnqjp ej nfttbhhjp dsjqubup -1
Debug algoritmo di decifratura A -> sempio di messaggio criptato -1
Stringa computata inizialmente decifrata -> sempio di messaggio criptato -1
Programma terminato!
Execution info
Cycles: 7479
Instrs. retired: 5206
CPI: 1.44
IPC: 0.696
Clock rate: 1.74 KHz
Processor: 5-stage processor ISA: RV32IM


File Edit View Help
1000 ms
Editor
Processor
Cache
Memory
I/O
5-Stage RISC-V Processor
4
PC
enable
Instr. memory
addr
instr
IFID
enable
clear
Decode
opcode
Registers
Wr data
Wr idx
R1 idx
R2 idx
Wr En
Reg 1
Reg 2
Imm.
IDEX
enable
clear
Op 1
Op 2
ALU
Res
Branch
Branch taken
EX/ MEM
Addr.
Data in
Wr en
Data memory
Read out
MEM/ WB
Registers
GPR
Name
Alias
Value
x0 zero 0x00000000
x1 ra 0x00000050
x2 sp 0x7ffffff0
x3 gp 0x10000000
x4 tp 0x00000000
x5 t0 0x00000000
x6 t1 0x00000000
x7 t2 0x00000000
x8 s0 0x00000000
x9 s1 0x00000041
x10 a0 0x10000023
x11 a1 0x0000c00b
x12 a2 0x0c000004
x13 a3 0x00000000
x14 a4 0x00000030
x15 a5 0x00000060
x16 a6 0x00000000
x17 a7 0x00000004
x18 s2 0x00000042
Display type:
Hex
Instruction memory
BP
Addr
Stage
Instruction
0x9a8 addi x11 x11 1639
0x9ac ecall
0x9b0 auipc x10 0xffff
0x9b4 addi x10 x10 1651
0x9b8 addi x17 x0 4
0x9bc ecall
Console
myStr0ng P4ssW_
Debug algoritmo di cifratura A -> nzTus0oh Q4ttX_
Debug algoritmo di cifratura B -> }&Y$ 5-t%`@y#dd
Debug algoritmo di cifratura C -> }-1 &-2 Y-3 $-4 -5 5-6 ~-7 t-8 %-9 `-10 @-11 y-12 #-13 d-14-15
Debug algoritmo di cifratura D -> }-8 &-7 b-6 $-5 -4 4-3 ~-2 G-1 %-0 `-89 @-88 B-87 #-86 W-85-84
Debug algoritmo di cifratura E -> 48-58-W 68-# 78-B 88-@ 98-` 0-% 1-G 2-~ 3-4 4- 5-$ 6-b 7-& 8-}
Debug algoritmo di decifratura E -> }-8 &-7 b-6 $-5 -4 4-3 ~-2 G-1 %-0 `-89 @-88 B-87 #-86 W-85-84
Debug algoritmo di decifratura D -> }-1 &-2 Y-3 $-4 -5 5-6 ~-7 t-8 %-9 `-10 @-11 y-12 #-13 d-14-15
Debug algoritmo di decifratura C -> }&Y$ 5-t%`@y#dd
Debug algoritmo di decifratura B -> nzTus0oh Q4ttX_
Debug algoritmo di decifratura A -> myStr0ng P4ssW_
Stringa computata inizialmente decifrata -> myStr0ng P4ssW_
Programma terminato!
Execution info
Cycles: 7652
Instrs. retired: 5226
CPI: 1.46
IPC: 0.683
Clock rate: 1.43 KHz
Processor: 5-stage processor ISA: RV32IM