Università degli studi di Firenze

Architetture degli Elaboratori A.A. 2018/2019

Progetto assembly MIPS - Messaggi Cifrati

A cura di Gioele Dimilta e Alberto Brogi

Indice

***Data di consegna**: 29/05/2019*

1. Introduzione

Il progetto consiste nella realizzazione di un programma in linguaggio assembly MIPS che simuli la funzionalità di cifratura e decifratura di un messaggio di testo. In input vengono forniti due file, messaggio.txt e chiave.txt, che contengono rispettivamente la stringa da criptare e la sequenza di caratteri associati agli algoritmi di cifratura da applicare al messaggio. Il programma, dopo aver letto i file in input, cripterà il messaggio e memorizzerà il risultato in un file di output denominato messaggioCifrato.txt, e, successivamente, eseguirà la procedura inversa di decrittazione salvando il nuovo risultato in un altro file di output, ovvero messaggioDecifrato.txt. Di seguito è riportata una descrizione completa dell'implementazione del programma in linguaggio Assembly MIPS e C. Quest'ultimo, data la sua bassa astrazione, ha permesso uno sviluppo semplificato del programma, fornendo inoltre caratteristiche di leggibilità e convertibilità del codice.

2. Descrizione ad alto livello - C

2.1. Variabili globali

La variabile "size_message" contiene la lunghezza del messaggio corrente. È fondamentale preservare il numero di caratteri della stringa perché la molteplice applicazione di algoritmi che modificano il valore ASCII dei caratteri potrebbe trasformare alcuni di essi in '/0', ovvero il carattere NULL. La presenza di NULL all'interno del messaggio non permette di utilizzare tale valore come carattere di fine stringa. L'array "sizes_arrays_E", dall'indice 1 a 4, contiene le lunghezze degli array utilizzati come appoggio per criptare/decrittare il messaggio, mentre all'indice 0 è salvato il numero di interi contenuti in sizes_arrays_E. Tali lunghezze sono utilizzate per effettuare la decrittazione del messaggio tramite l'algoritmo E. Tutti i valori dell'array sono inizializzati a zero, ma ciò vale solo per l'implementazione in C. In Assembly, ogni indice del vettore contiene il valore della lunghezza massima raggiungibile dal messaggio (con tutti i caratteri diversi) dopo esser stato criptato tramite l'algoritmo E. I valori verranno utilizzati per la creazione di blocchi di memoria HEAP al fine di contenere la stringa criptata e in seguito modificati per contenere le lunghezze "reali" delle stringhe criptate.

ESEMPIO

key = "EEE", message = "This is a text string", sizes_arrays_E = {0, 0, 0, 0, 0}.

	sizes_array_E	size_message
1ª Esecuzione di algorithm_encrypt_E	{1, 21, 0, 0, 0}	76
2ª Esecuzione di algorithm_encrypt_E	{2, 21, 76, 0, 0}	283
3ª Esecuzione di algorithm_encrypt_E	{3, 21, 76, 283, 0}	987

	sizes_array_E	size_message
1ª Esecuzione di algorithm_decrypt_E	{2, 21, 76, 283, 0}	283
2ª Esecuzione di algorithm_decrypt_E	{1, 21, 76, 283, 0}	76
3ª Esecuzione di algorithm_decrypt_E	{0, 21, 76, 283, 0}	21

2.2. Strutture dati

Le stringhe lette dai file di input e le stringhe criptate tramite l'algoritmo E vengono memorizzate nella memoria HEAP. Le funzioni che permettono di gestire la memoria dinamica sono malloc() e realloc(), con cui è possibile rispettivamente allocare e riallocare blocchi di bytes, detti chunk. La prima chiamata di allocazione è eseguita tramite malloc(), che, dato un numero di bytes passati come argomento, ricerca un blocco di memoria libero e restituisce l'indirizzo di partenza del chunk allocato. Nel caso sia necessaria aumentare lo spazio di allocazione, una chiamata al metodo realloc() permette di avere a disposizione un nuovo indirizzo corrispondente ad un nuovo blocco di memoria. L'indirizzo del vecchio chunk ed il nuovo numero di bytes da allocare, passati entrambi come argomenti, permettono al metodo di ricercare un nuovo blocco di memoria della grandezza richiesta e copiare all'interno di esso le informazioni del vecchio chunk. Quando lo spazio allocato non serve più, è possibile eliminarlo logicamente tramite la funzione free(), in modo da renderlo disponibile per allocazioni successive. L'utilizzo dell'Heap ha come vantaggio principale un minor spreco di memoria poichè viene allocato dinamicamente il numero di byte necessari per l'esecuzione e, di conseguenza, non vi è la necessità di prevedere uno spazio in memoria di grandezza statica che potrebbe non essere sfruttato a pieno. In Assembly, tale implementazione non può essere totalmente rispettata perché le funzioni di malloc/realloc/free non sono disponibili. Pertanto, la dimensione di alcuni blocchi allocati sarà grande esattamente quanto abbastanza per poter gestire il caso pessimo, ma ovviamente ciò causerà spreco di memoria.

2.3. Funzioni

2.3.1. File

open_f

• Firma: FILE *open_f(const char *name, const char *type);

• Argomenti:

- char **name* = Indirizzo della stringa corrispondente al nome del file da leggere.
- char **type* = Carattere relativo alla modalità di apertura del file ("r" = lettura, "w" = scrittura).

• Return: File descriptor.

• Descrizione: Apertura di un file tramite la funzione fopen() dato un nome *name* ed una modalità di apertura *type*.

read_f

• Firma: unsigned char *read_f(FILE *f, int *length);

• Argomenti:
  - FILE **f* = File descriptor
  - int **length* = Indirizzo del blocco di memoria che, al termine della funzione, conterrà la lunghezza del file.

• Return: Indirizzo del chunk contenente la stringa letta da file.

• Descrizione: Lettura di un file carattere per carattere tramite la funzione fgetc() e memorizzazione dei dati in un chunk. Il valore di *length*, all'inizio della funzione, sarà 0 e verrà incrementato durante il ciclo di lettura del file. Al fine di evitare l'utilizzo di una struct per restituire sia l'indirizzo del chunk, sia la lunghezza del file, viene effetuato l'aggiornamento del valore della lunghezza tramite il puntatore *length*, anche se durante la lettura del file chiave.txt tale operazione non sia necessaria. Al termine del ciclo for, viene inserito il valore di fine stringa nell'ultimo bytes del chunk in modo da poter scorrere la stringa nelle funzioni in cui è richiesto. Tale operazione è strettamente necessaria solo per la lettura del file chiave.txt, dato che per la stringa di messaggio.txt è memorizzato il valore della lunghezza.

write_f

• Firma: void write_f(FILE *f, unsigned char *output_text);

• Argomenti:
  - FILE **f* = File descriptor
  - unsigned char ***output_text* = Indirizzo del chunk contenente la stringa da scrivere sul file.

• Return: /

• Descrizione: Scrittura su un file carattere per carattere della stringa *output_text* tramite la funzione fprintf().

close_f

• Firma: void close_f(FILE *f);

• Argomenti:
  - FILE **f* = File descriptor

• Return: /

• Descrizione: Chiusura del file tramite la funzione fclose();

orc_f (**o**pen-**r**ead-**c**lose)

- Firma: unsigned char *orc_f(const char *name, int *length);
- Argomenti:
  - FILE **f* = File descriptor
  - int **length* = Indirizzo del blocco di memoria che, al termine della funzione, conterrà la lunghezza del file.
- Return: Indirizzo del chunk contenente la stringa letta da file.
- Descrizione: Apertura, lettura e chiusura di un file tramite le funzioni open_f(), read_f(), close_f().

owc_f (**o**pen-**w**rite-**c**lose)

- Firma: void owc_f(const char *name, unsigned char *input_text);
- Argomenti:
  - FILE **f* = File descriptor
  - unsigned char **input_text* = Indirizzo del chunk contenente la stringa da scrivere sul file.
- Return: /
- Descrizione: Apertura, scrittura e chiusura di un file tramite le funzioni open_f(), write_f(), close_f().

2.3.2. Gestione

encrypt_and_decrypt

- Firma: void encrypt_and_decrypt(unsigned char **input_text, char *key);
- Argomenti:
  - unsigned char ***input_text* = Indirizzo del puntatore alla stringa da criptare/decrittare.
  - char **key* = Indirizzo del chunk contenente i caratteri letti dal file chiave.txt
- Return: /
- Descrizione: La funzione scorre i caratteri di *key* tramite un ciclo while e, ad ogni iterazione, richiama la procedura choose_algorithm() per applicare un determinato algoritmo a *input_text*. Data la necessità di eseguire algoritmi prima di criptazione e successivamente di decrittazione, i caratteri di *key* verranno visitati 2 volte. In particolare, una volta raggiunto il carattere di fine stringa '\0', il ciclo while dovrà scorrere i caratteri n direzione opposta, cambiando il valore di increase (da +1 a -1) per decrementare il contatore i. La scrittura sui file di output avviene dopo aver criptato o decrittato il messaggio con tutti gli algoritmi corrispondenti alla chiave, ovvero quando ci troviamo al termine di *key* oppure dopo il ciclo while. L'utilizzo di ***input_text* come indirizzo del puntatore al messaggio e non come indirizzo del messaggio (**input_text*) permette di non perdere il riferimento al messaggio e quindi di poter liberare la memoria al termine del programma.

choose_algorithm

- Firma: unsigned char *choose_algorithm(unsigned char *input_text, int encrypt, char character);
- Argomenti:
  - unsigned char **input_text* = Indirizzo del chunk contenente la stringa da criptare/decrittare.
  - int *encrypt* = Intero che indica quale operazione effettuare tra criptazione (+1) o decrittazione (-1).
  - char *character* = Carattere di chiave.txt relativo all'algoritmo da applicare alla stringa.
- Return: Indirizzo al chunk contenente la stringa criptata o decrittata.
- Descrizione: Un costrutto switch-case permette di applicare un algoritmo a *input_text* in base al valore di *character*. Ogni algoritmo utilizza una sola funzione per le operazioni di criptazione e decrittazione, tranne la E. Le lettere contenute nel file chiave.txt devono essere obbligatoriamente maiuscole altrimenti il valore ascii non verrà riconosciuto e nessun algoritmo verrà eseguito.

2.3.3. Calcolo

change_ascii_value

- Firma: void change_ascii_value(unsigned char *input_text, int encrypt, int remainder);
- Argomenti:
  - unsigned char **input_text* = Indirizzo del chunk contenente la stringa da criptare/decrittare.
  - int *encrypt* = Intero che indica quale operazione effettuare tra criptazione (+1) o decrittazione (-1).
  - int *remainder* = Intero che indica quale valore deve assumere il risultato dell'operazione di modulo tra la posizione di un carattere del messaggio ed il valore 2.
- Return: /
- Descrizione: La funzione aggiunge o sottrae 4 al valore ASCII dei caratteri di *input_text*. La variabile *encrypt* cambia il proprio valore in base all'operazione da effettuare, cioè criptazione (*encrypt* = 4) oppure decrittazione (*encrypt* = -4), mentre *remainder* consente di scegliere quali caratteri modificare secondo un determinato algoritmo (A, B oppure C). L'operazione di Shift left logico permette di moltiplicare per 4 il valore di *encrypt* e non sono necessari controlli di overflow aritmetico perchè il valore di *encrypt* può variare secondo 2 soli valori, ovvero +1 e -1. L'indirizzamento del MIPS è al byte ma il vincolo di allineamento (ogni word deve iniziare ad un indirizzo multiplo di 4) consente di avere un indirizzo multiplo di 4 ogni volta che viene allocato un nuovo byte dell'Heap con una syscall. Perciò, è possibile effettuare l'operazione di modulo direttamente sugli indirizzo dei caratteri senza l'uso di indici di scorrimento. In C il funzionamento è il medesimo. Infine, la funzione non utilizza l'operazione di modulo 256 perché la modifica del valore ASCII dei caratteri avviene su 1 byte senza estensione del segno (l'intervallo in cui il valore può variare è [0, 255]) e l'intervallo è già circolare, infatti 0xFF + 0x01 = 0x100 (signed/unsigned), 0x00 - 0x01 = 0xFF (unsigned), 0x00 - 0x01 =

0xFFFFFFFF (signed). In ogni caso, la circolarità dell'intervallo è garantita perché consideriamo solo il byte meno significativo.

ESEMPIO

	Signed (HEX, DEC)	Unsigned (HEX, DEC)
i	0x0000007F, 127	0x0000007F, 127
i = i + 2	0xFFFFFF81, -127	0x00000081, 129
i = i + 2	0xFFFFFF83, -125	0x00000083, 131

Signed char i = 127, unsigned char j = 127;

insert_index

- Firma: unsigned char *insert_index(unsigned char *input_text, int *length, int index);
- Argomenti:
  - unsigned char **input_text* = Indirizzo del chunk contenente la stringa criptata.
  - int **length* = Numero di caratteri di *input_text*.
  - int *index* = Indice da inserire in *input_text*.
- Return: Indirizzo del chunk contenente la stringa criptata.
- Descrizione: Inserimento dell'indice *index* in *input_text*. Il primo ciclo while conta il numero di cifre dell'indice incrementando il valore di *length*. La chiamata alla funzione realloc permette di allocare i bytes necessari per memorizzare le cifre di *index*. Il ciclo for inserisce *index* in end_text (cioè l'indirizzo dell'ultimo byte allocato) cifra per cifra partendo dalla meno significativa sfruttando l'operazione di modulo 10 e ad ogni cifra viene sommato il valore ascii di zero per trasformarla in un carattere. Successivamente, viene aggiornato il valore di *index* in modo da eliminare la cifra meno significativa.

insert_character

- Firma: unsigned char *insert_character(unsigned char *input_text, int *length, unsigned char character);
- Argomenti:
  - unsigned char **input_text* = Indirizzo del chunk contenente la stringa criptata.
  - int **length* = Numero di caratteri di *input_text*.
  - unsigned char *character* = Carattere da inserire in *input_text*.
- Return: Indirizzo del chunk contenente la stringa criptata.
- Descrizione: Alloca nuovo spazio, inserisce il carattere alla posizione *length*-1 e incrementa la lunghezza di *input_text*.

2.3.4. Algoritmi

algorithm_A

- Firma: void algorithm_A(unsigned char *input_text, int encrypt);
- Argomenti:
  - unsigned char **input_text* = Indirizzo del chunk contenente la stringa da criptare/decrittare.
  - int *encrypt* = Intero che indica quale operazione effettuare tra criptazione (+1) o decrittazione (-1).
- Return: /
- Descrizione: La funzione richiama il metodo change_ascii_value() per applicare l'algoritmo A. L'algoritmo prevede la modifica del valore ascii di tutti i caratteri di *input_text*. Dato che, per convenzione matematica, il resto non può essere negativo, verrà passato come valore dell'argomento *remainder* l'intero -1 in modo da modificare tutti i caratteri di *input_text*.

algorithm_B

- Firma: void algorithm_B(unsigned char *input_text, int encrypt);
- Argomenti:
  - unsigned char **input_text* = Indirizzo del chunk contenente la stringa da criptare/decrittare.
  - int *encrypt* = Intero che indica quale operazione effettuare tra criptazione (+1) o decrittazione (-1).
- Return: /
- Descrizione: La funzione richiama il metodo change_ascii_value() per applicare l'algoritmo B. L'algoritmo prevede la modifica del valore ascii dei caratteri di *input_text* in posizione pari, ovvero i caratteri per cui vale la condizione posizione%2 = 0.

algorithm_C

- Firma: void algorithm_C(unsigned char *input_text, int encrypt);
- Argomenti:
  - unsigned char **input_text* = Indirizzo del chunk contenente la stringa da criptare/decrittare.
  - int *encrypt* = Intero che indica quale operazione effettuare tra criptazione (+1) o decrittazione (-1).
- Return: /
- Descrizione: La funzione richiama il metodo change_ascii_value() per applicare l'algoritmo C. L'algoritmo prevede la modifica del valore ascii dei caratteri di *input_text* in posizione dispari, ovvero i caratteri per cui vale la condizione posizione%2 = 1.

algorithm_D

- Firma: void algorithm_D(unsigned char *input_text);
- Argomenti:
  - unsigned char **input_text* = Indirizzo del chunk contenente la stringa da criptare/decrittare.
- Return: /
- Descrizione: La funzione scambia i caratteri di *input_text* appartenenti a posizioni opposte. Due puntatori al primo (*input_text*) e all'ultimo (end_text) carattere scorrono la stringa in direzioni opposte e i caratteri vengono invertiti finché i puntatori non si incontrano.

algorithm_encrypt_E

- Firma: unsigned char *algorithm_encrypt_E(unsigned char *input_text);
- Argomenti:
  - unsigned char **input_text* = Indirizzo del chunk contenente la stringa da criptare.
- Return: Indirizzo del chunk contenente la stringa criptata.
- Descrizione: La funzione fissa un carattere di indice j nella variabile character e scorre *input_text* per ricercare possibili occorrenze. I valori inseriti in output_text vengono modificati con il primo carattere di *input_text*, in modo da contrassegnare i valori già processati. Ciò significa che, se character è uguale a input_text[0], fatta eccezione per la prima iterazione (j==0), non deve essere inserito in output_text. Quando j==0, il carattere input_text[j] è sicuramente uguale a input_text[0], perciò, se non ci fosse tale condizione, la prima iterazione non inserirebbe nessun carattere e nessun indice in output_text. Dopo aver inserito tutti gli indici delle occorrenze di character, viene richiamata la funzione insert_character per inserire il carattere spazio in output_text, in modo da separare gli indici appartenenti a caratteri differenti. Al termine dei cicli for, la funzione incrementa sizes_arrays_E[0] per far corrispondere tale valore all'indice della nuova lunghezza da inserire, per poi effettuare l'inserimento di size_message e aggiornare quest'ultimo alla lunghezza di output_text. Il valore di length viene decrementato prima di essere assegnato a size_message perché nell'ultima operazione viene inserito un carattere di spazio alla fine di output_text, ma tale carattere deve essere ignorato. Dopo aver liberato lo spazio allocato da *input_text*, la procedura restituisce l'indirizzo del nuovo chunk contenente la stringa criptata.

algorithm_decrypt_E

- Firma: unsigned char *algorithm_decrypt_E(unsigned char *input_text);
- Argomenti:
  - unsigned char **input_text* = Indirizzo del chunk contenente la stringa da decrittare.
- Return: Indirizzo del chunk contenente la stringa decrittata.
- Descrizione: La funzione scorre i caratteri di *input_text* ed effettua delle operazioni in base a 3 casi:

Se *input_text*[i-1] == ' ' && *input_text*[i+1] == '-', significa che *input_text*[i] è la prima occorrenza di un carattere;
Se *input_text*[i] == ' ' || *input_text*[i] == '-', significa che è stato calcolato un indice e il valore di character deve essere salvato in output_text esattamente alla posizione index. Fa eccezione il primo carattere '-' dopo la prima occorrenza perchè non segue a nessun indice;
Se le due condizioni precedenti non sono soddisfatte, significa che il carattere corrente è in realtà una cifra dell'indice. La moltiplicazione per 10 permette di scalare ogni cifra di una unità di grandezza, così da lasciare libera la cifra meno significativa per inserire *input_text*[i]. L'operazione *input_text*[i]-'0' permette di convertire un carattere nel valore ascii corrispondente alla cifra che rappresenta. La variabile *output_text contiene l'indirizzo al chunk che memorizzerà i caratteri della stringa decrittata. La lunghezza del chunk dipende dalle precedenti esecuzioni di algorithm_encrypt_E(). Al termine del ciclo for, la funzione inserisce character in output_text alla posizione index, ovvero l'ultimo indice calcolato, aggiorna il valore di size_message con la lunghezza di output_text (la stessa utilizzata per settare output_text), decrementa sizes_arrays_E[0] per far corrispondere tale valore all'indice della lunghezza da utilizzare alla prossima esecuzione del metodo algorithm_decrypt_E(). Dopo aver liberato lo spazio allocato da *input_text*, la procedura restituisce l'indirizzo del nuovo chunk contenente la stringa decrittata.

2.4. *Codice sorgente*

```
#include <stdio.h>
#include <stdlib.h>

FILE *open_f(const char *name, const char *type);
unsigned char *read_f(FILE *f, int *length);
void write_f(FILE *f, unsigned char *output_text);
void close_f(FILE *f);
unsigned char *orc_f(const char *name, int *length);
void owc_f(const char *name, unsigned char *input_text);

void encrypt_and_decrypt(unsigned char **input_text, char *key);
unsigned char *choose_algorithm(unsigned char *input_text, int encrypt, char character);

void change_ascii_value(unsigned char *input_text, int encrypt, int remainder);
unsigned char *insert_index(unsigned char *input_text, int *length, int index);
unsigned char *insert_character(unsigned char *input_text, int *length, unsigned char
character);

void algorithm_A(unsigned char *input_text, int encrypt);
void algorithm_B(unsigned char *input_text, int encrypt);
void algorithm_C(unsigned char *input_text, int encrypt);
void algorithm_D(unsigned char *input_text);
unsigned char *algorithm_encrypt_E(unsigned char *input_text);
unsigned char *algorithm_decrypt_E(unsigned char *input_text);

void print_text(char name, unsigned char *input_text, int length);
```

```
int size_message = 0;
int sizes_arrays_E[5] = {0, 0, 0, 0, 0};


int main(int argc, const char *argv[]) {
    int length = 0;
    unsigned char *message = orc_f("messaggio.txt", &size_message);
    printf("OUTPUT_MESSAGE: %s \n\n", message);

    char *key = (char*) orc_f("chiave.txt", &length);
    printf("OUTPUT_KEY: %s \n\n", key);

    encrypt_and_decrypt(&message, key);

    free(key);
    free(message);

    return 0;
}


FILE *open_f(const char *name, const char *type) {
    FILE *f = fopen(name, type);

    if(f == NULL) {
        exit(EXIT_FAILURE);
    }

    return f;
}

unsigned char *read_f(FILE *f, int *length) {
    char character;
    unsigned char *input_text = malloc(sizeof(unsigned char));

    for(*length=1; (character = fgetc(f)) != EOF; (*length)++) {
        input_text[*length-1] = character;
        input_text = realloc(input_text, sizeof(unsigned char)*(*length+1));
    }

    (*length)--;
    input_text[*length] = '\0';

    return input_text;
}

void write_f(FILE *f, unsigned char *intput_text) {
    for(int i=0; i<size_message; i++) {
        fprintf(f, "%c", intput_text[i]);
    }
}
```

```
void close_f(FILE *f) {
    fclose(f);
}

unsigned char *orc_f(const char *name, int *length) {
    FILE *f = open_f(name, "r");
    unsigned char *input_text = read_f(f, length);
    close_f(f);

    return input_text;
}

void owc_f(const char *name, unsigned char *input_text) {
    FILE *f = open_f(name, "w");
    write_f(f, input_text);
    close_f(f);
}



void encrypt_and_decrypt(unsigned char **input_text, char *key) {
    int i=0;
    int increase = 1;

    while(i >= 0) {
        if(key[i] == '\0') {
            owc_f("messaggioCifrato.txt", *input_text);
            i += (increase = -1);
        }

        *input_text = choose_algorithm(*input_text, increase, key[i]);
        print_text(key[i], *input_text, size_message);

        i += increase;
    }

    owc_f("messaggioDecifrato.txt", *input_text);
}

unsigned char *choose_algorithm(unsigned char *input_text, int encrypt, char character) {
    switch(character) {
        case 'A': {
            algorithm_A(input_text, encrypt);
            break;
        }

        case 'B': {
            algorithm_B(input_text, encrypt);
            break;
        }
```

```
        case 'C': {
            algorithm_C(input_text, encrypt);
            break;
        }

        case 'D': {
            algorithm_D(input_text);
            break;
        }

        case 'E': {
            input_text = (encrypt == 1)
            ? algorithm_encrypt_E(input_text)
            : algorithm_decrypt_E(input_text);
            break;
        }
    }

    return input_text;
}


void change_ascii_value(unsigned char *input_text, int encrypt, int remainder) {
    unsigned char *end_text = input_text+size_message;
    encrypt <<= 2;

    while(input_text != end_text) {
        if(remainder == -1 || ((int) input_text)%2 == remainder) {
            *input_text += encrypt;
        }

        input_text++;
    }
}


unsigned char *insert_index(unsigned char *output_text, int *length, int index) {
    int temp = index;
    unsigned char *end_text;

    do {
        temp = temp/10;
        (*length)++;

    } while(temp != 0);

    output_text = realloc(output_text, sizeof(unsigned char)*(*length));

    if(output_text == NULL) {
        exit(EXIT_FAILURE);
```

```
    }

    end_text = &output_text[*length-1];

    do {
        *end_text = index%10 + '0';
        index /= 10;

        end_text--;
    } while(index != 0);

    return output_text;
}


unsigned char *insert_character(unsigned char *output_text, int *length, unsigned char
character) {
    (*length)++;
    output_text = realloc(output_text, sizeof(unsigned char)*(*length));

    if(output_text == NULL) {
        exit(EXIT_FAILURE);
    }

    output_text[(*length)-1] = character;

    return output_text;
}


void algorithm_A(unsigned char *input_text, int encrypt) {
    change_ascii_value(input_text, encrypt, -1);
}

void algorithm_B(unsigned char *input_text, int encrypt) {
    change_ascii_value(input_text, encrypt, 0);
}

void algorithm_C(unsigned char *input_text, int encrypt) {
    change_ascii_value(input_text, encrypt, 1);
}

void algorithm_D(unsigned char* input_text) {
    unsigned char temp, *end_text = input_text + (size_message-1);

    while(input_text < end_text) {
        temp = *input_text;
        *input_text = *end_text;
        *end_text = temp;

        input_text++;
```

```
        end_text--;
    }
}



unsigned char *algorithm_encrypt_E(unsigned char *input_text) {
    int length = 0;
    unsigned char character, *output_text = malloc(sizeof(unsigned char));

    if(output_text == NULL) {
        exit(EXIT_FAILURE);
    }

    for(int j=0; j<size_message; j++) {
        character = input_text[j];

        if(j==0 || character != input_text[0]) {
            output_text = insert_character(output_text, &length, character);

            for(int i=j; i<size_message; i++) {
                if(character == input_text[i]){
                    output_text = insert_character(output_text, &length, '-');
                    output_text = insert_index(output_text, &length, i);
                    input_text[i] = input_text[0];
                }
            }

            output_text = insert_character(output_text, &length, ' ');
        }
    }

    sizes_arrays_E[0]++;
    sizes_arrays_E[sizes_arrays_E[0]] = size_message;
    size_message = length-1;
    free(input_text);

    return output_text;
}

unsigned char *algorithm_decrypt_E(unsigned char *input_text) {
    int index = 0;
    unsigned char *output_text = malloc(sizeof(char)*sizes_arrays_E[sizes_arrays_E[0]]),
    character = output_text[0] = input_text[0];

    if(output_text == NULL) {
        exit(EXIT_FAILURE);
    }

    for(int i=1; i<size_message; i++) {
        if(input_text[i-1] == ' ' && input_text[i+1] == '-') {
```

```
            character = input_text[i];
        }

        else if((input_text[i] == ' ' || input_text[i] == '-')) {
            if(index) {
                output_text[index] = character;
                index = 0;
            }
        }

        else {
            index = index*10 + ((int) input_text[i]-'0');
        }
    }

    output_text[index] = character;
    size_message = sizes_arrays_E[sizes_arrays_E[0]];
    sizes_arrays_E[0]--;
    free(input_text);

    return output_text;
}


void print_text(char name, unsigned char *input_text, int length) {
    printf("OUTPUT_%c: ", name);

    for(int i=0; i<length; i++) {
        printf("%c", input_text[i]);
    }

    printf("\n\n");
}
```

3. Descrizione a basso Livello - Assembly MIPS

3.1. Data Segment

Il segmento dati contiene i dati statici utilizzati durante l'esecuzione del programma. Di seguito sono elencati i nomi delle etichette e i dati a cui esse corrispondono:

- **size_message** = Etichetta associata all'indirizzo della lunghezza del messaggio corrente. Un intero è formato da 4 bytes, perciò viene allocata una word per contenere il valore della lunghezza.
- **sizes_arrays_E** = Etichetta associata all'indirizzo di partenza dell'array contenente le lunghezze dei vettori utilizzati per memorizzare il messaggio criptato/decrittato con l'algoritmo E. A differenza dell'implementazione in C, l'array è inizializzato con le lunghezze massime raggiungibili dal messaggio con

128 caratteri (tutti differenti) criptato con l'algoritmo E. Ogni indice corrisponde al numero di applicazioni dell'algoritmo effettuate sul messaggio.
- **jump_table** = Etichetta associata all'indirizzo di partenza dell'array contenente gli indirizzi dei CASE implementati nel costrutto switch/case del metodo choose_algorithm(). Ogni CASE è rappresentato con un' etichetta denominata con la lettera dell'algoritmo a cui è associata preceduta da una L (LABEL), e occupa uno spazio di 4 bytes perchè gli indirizzi sono a 32 bit.
- **file_not_found** = Etichetta associata all'indirizzo di partenza della stringa utilizzata per comunicare un errore di FILE NOT FOUND durante l'esecuzione. Termina con un carattere di fine stringa.
- **sbrk_error** = Etichetta associata all'indirizzo di partenza della stringa utilizzata per comunicare un errore di relativo all'allocazione di nuova memoria nell'heap. Termina con un carattere di fine stringa.
- **chiave** = Etichetta associata all'indirizzo di partenza della stringa contenente il path del file chiave.txt Termina con un carattere di fine stringa.
- **messaggio** = Etichetta associata all'indirizzo di partenza della stringa contenente il path del file messaggio.txt Termina con un carattere di fine stringa.
- **messaggioCifrato** = Etichetta associata all'indirizzo di partenza della stringa contenente il path del file messaggioCifrato.txt Termina con un carattere di fine stringa.
- **messaggioDecifrato** = Etichetta associata all'indirizzo di partenza della stringa contenente il path del file messaggioDecifrato.txt Termina con un carattere di fine stringa.

3.2. Strutture dati

Le stringhe lette dai file di input e le stringhe criptate tramite l'algoritmo E vengono memorizzate nella memoria HEAP. A differenza del linguaggio C, non sono disponibili funzioni per allocare e riallocare dinamicamente la memoria, bensì è presente solo una chiamata a sistema SBRK di allocazione che restituisce l'indirizzo del blocco di bytes richiesto. Ciò significa che, non essendoci una procedura per riallocare i dati, l'indirizzo restituito dalla syscall SBRK corrisponderà alla prima word libera nell'HEAP. In generale, però, non è possibile assumere che chiamate successive di allocazione restituiscano indirizzi consecutivi, perciò non è possibile sfruttare al massimo la potenzialità della memoria dinamica. I blocchi utilizzati avranno una dimensione tale da poter contenere stringhe di lunghezza massima possibile e perciò ci sarà inevitabilmente spreco di memoria. Il contenuto dei registri permanenti e del registro $ra verrà preservato temporaneamente nello stack nel caso in cui una procedura faccia uso di questi registri, o, nel caso di $ra, se richiamerà un'altra procedura.

3.3. Procedure

L'implementazione delle procedure segue il corrispondente codice C, ma per alcune funzioni ci sono delle differenze, soprattutto riguardo all'allocazione dinamica di bytes nella memoria HEAP.

3.3.1. File

open_f

- Argomenti:
  # *$a0* = Indirizzo di partenza del nome del file.
  # *$a1* = Flag di modalità (0 = lettura, 1 = scrittura).
- Return:
  # $v0 = File descriptor.
- Descrizione: Apertura del file *$a0* tramite la syscall 13. In caso di errore, il branch richiama la procedura di errore.

read_f

- Argomenti:
  # *$a0* = File Descriptor.
  # *$a1* = Numero massimo di bytes da leggere dal file.
- Return:
  # *$v0* = Indirizzo di partenza del messaggio letto dal file.
  # *$v1* = Numero di bytes effettivamente letti da file.
- Descrizione: Lettura del file *$a0* tramite la syscall 14 e memorizzazione dei dati in un chunk allocato con una syscall 9. A differenza dell'implementazione in C, la procedura ha due valori di ritorno, uno dei quali è la lunghezza del messaggio letto *$v1*, perciò non è necessario utilizzare un puntatore al blocco di memoria della lunghezza come argomento della procedura. Il valore della lunghezza viene memorizzato durante la chiamata di orc_f() del file messaggio.txt, mentre per quanto riguarda il file chiave.txt, semplicemente viene ignorato il valore di ritorno del registro $v1. In caso di errore durante l'allocazione di memoria, il branch richiama la procedura di errore.

write_f

- Argomenti:
  # *$a0* = File Descriptor.
  # *$a1* = Indirizzo di partenza del messaggio da scrivere nel file.
- Return: /
- Descrizione: Scrittura del messaggio *$a1* sul file *$a0* tramite la syscall 15.

close_f

- Argomenti:
  # *$a0* = File Descriptor
- Return: /
- Descrizione: Chiusura del file *$a0* tramite la syscall 16.

orc_f (**o**pen-**r**ead-**c**lose)

- Argomenti:
  # *$a0* = Indirizzo di partenza del nome del file.
  # *$a1* = Numero massimo di byte da leggere dal file.
- Return:
  # *$v0* = Numero di bytes effettivamente letti da file.
  # *$v1* = Indirizzo di partenza del messaggio letto dal file.
- Descrizione: Apertura, lettura e chiusura del file *$a0* tramite le procedure open_f(), read_f(), close_f(). L'utilizzo di sotto-procedure obbliga a memorizzare alcune informazioni per fare in modo che non vengano sovrascritte, come il file descriptor, in registri permanenti $s che, per convenzione, mantengono il valore impostato dal chiamante di una procedura. Per questo, orc_f(), essendo anch'essa una procedura, ha l'obbligo di preservare il contenuto dei registri permanenti prima di utilizzarli. Inoltre, è necessario salvare anche il valore del registro $ra perché l'uso di sotto-procedure compromette il contenuto di tale registro, che viene usato per ritornare alla procedura chiamante dopo aver eseguito tutte le istruzioni. Le prime istruzioni memorizzano nella memoria Stack il contenuto dei registri $s e $ra che verranno utilizzati successivamente, mentre le ultime istruzioni ne ripristinano il valore.

owc_f (**o**pen-**w**rite-**c**lose)

- Argomenti:
  # *$a0* = Indirizzo di partenza del nome del file.
  # *$a1* = Indirizzo di partenza del messaggio memorizzato nell'Heap.
- Return: /
- Descrizione: Apertura di $a0, scrittura del messaggio *$a1* nel file *$a0* e chiusura di $a0 tramite le procedure open_f(), write_f(), close_f(). Anche in questo caso, i valori dei registri $s e $ra vengono memorizzati e poi ripristinati per poter richiamare sotto-procedure.

3.3.2. Gestione

encrypt_and_decrypt

- Argomenti:
  # *$a0* = Indirizzo della stringa da criptare/decrittare.
  # *$a1* = Indirizzo del chunk contenente i caratteri letti dal file chiave.txt

- Return: /
- Descrizione: La funzione scorre i caratteri di *$a1* tramite un ciclo while e, ad ogni iterazione, richiama la procedura choose_algorithm() per applicare un determinato algoritmo a *$a0*. Il costrutto while è stato convertito in un ciclo do-while perchè, sapendo che il file chiave.txt è ben formato (quindi non vuoto), ci sarà almeno un carattere che corrisponderà ad uno degli algoritmi del programma, quindi sicuramente sarà necessario eseguire almeno una iterazione del ciclo. Non essendoci una procedura per liberare la memoria Heap, non è necessario usare l'indirizzo del puntatore al messaggio come nel codice C, perciò *$a0* contiene l'indirizzo del messaggio. Come nelle procedure della sezione 3.3.1, i valori dei registri $s e $ra vengono memorizzati e poi ripristinati per poter richiamare sotto-procedure.

choose_algorithm

- Argomenti:
  # *$a0* = Indirizzo del chunk contenente la stringa da criptare/decrittare.
  # *$a1* = Intero che indica quale operazione effettuare tra criptazione (+1) o decrittazione (-1).
  # *$a2* = Carattere di chiave.txt relativo all'algoritmo da applicare alla stringa.
- Return:
  # *$v0* = Indirizzo al chunk contenente la stringa criptata o decrittata.
- Descrizione: Un costrutto switch-case permette di applicare un algoritmo ad *$a0* in base al valore di *$a2*. Lo switch/case viene tradotto in quanto in MIPS non è presente tale costrutto. Nel segmento dati è presente un array il cui indirizzo di partenza è associato all'etichetta jump_table. Ogni word dell'array contiene un indirizzo che rappresenta un case dello switch. Per saltare alla porzione di codice desiderata (cioè entrare in un case), è necessario sommare all'indirizzo di jump_table l'indice della posizione moltiplicato per 4 (gli indirizzi di jump_table occupano 4 bytes). L'operazione di shif left logico permette di moltiplicare per 2 il valore dell'indice ed è utilizzabile al posto della normale moltiplicazione perchè viene eseguita più velocemente dal processore ed il controllo di overflow non è necessario. I case si differenziano per il carattere dell'algoritmo a cui corrispondono, perciò basterà sommare a jump_table un numero sapendo che:
  jump_table[0] = case 'A';
  jump_table[1] = case 'B';
  jump_table[2] = case 'C';
  jump_table[3] = case 'D';
  jump_table[4] = case 'E';

3.3.3. Calcolo

change_ascii_value

- Argomenti:
  # *$a0* = Indirizzo del chunk contenente la stringa da criptare/decrittare.
  # *$a1* = Intero che indica quale operazione effettuare tra criptazione (+1)

o decrittazione (-1).
$a2 = Intero che indica quale valore deve assumere il risultato dell'operazione di modulo tra la posizione di un carattere del messaggio ed il valore 2.

- Return: /
- Descrizione: La funzione aggiunge o sottrae 4 al valore ASCII dei caratteri di *$a0*. Dato che per le specifiche del progetto il file messaggio.txt è ben formato, è possibile tradurre il ciclo while in un do-while perché il messaggio avrà almeno un carattere da criptare. Per il resto, il codice segue l'implementazione di C.

insert_index

- Argomenti:
  # *$a0* = Indirizzo dell'ultimo carattere della stringa criptata.
  # *$a1* = Numero di caratteri di *$a0*.
  # *$a2* = Indice da inserire in *$a0*.
- Return:
  # *$v0* = Indirizzo dell'ultimo carattere della stringa criptata.
  # *$v1* = Numero di caratteri di *$v0.*
- Descrizione: Inserimento dell'indice *$a2* in *$a0*. All'interno del primo loop, viene incrementato il valore di una variabile che non è presente nel codice C, che chiameremo digits_index. La variabile contiene il numero di cifre di *$a2* e viene utilizzata per poter calcolare *end_text.* A differenza dell'implementazione in C, *$a0* contiene l'indirizzo dell'ultimo carattere della stringa criptata invece dell'indirizzo iniziale poichè quest'ultimo, dato che in Assembly non è presente la funzione di deallocazione dell'HEAP, non verrà utilizzato alla fine della procedura di algorithm_encrypt_E. L'uso dell'indirizzo dell'ultimo carattere permette di calcolare *end_text* senza necessità di una istruzione di load word per estrarre dalla memoria il valore della lunghezza.
  Inoltre, la procedura restituisce anche la lunghezza del chunk criptato, a differenza del C in cui viene modificato il valore del puntatore length passato come argomento.

insert_character

- Argomenti:
  # *$a0* = Indirizzo dell'ultimo carattere della stringa criptata.
  # *$a1* = Numero di caratteri di *input_text*.
  # *$a2* = Carattere da inserire in *input_text*.
- Return:
  # *$v0* = Indirizzo dell'ultimo carattere della stringa criptata.
  # *$v1* = Numero di caratteri di *$v0.*
- Descrizione: Alloca nuovo spazio, inserisce il carattere *$a2* all'indirizzo *$a0*+1 e incrementa *$a1*. Come già spiegato precedentemente per la procedura insert_index, *$a0* contiene l'indirizzo dell'ultimo carattere della stringa criptata invece dell'indirizzo iniziale data l'impossibilità di deallocare. Inoltre, la procedura restituisce anche la lunghezza del chunk criptato, a differenza del C in cui viene modificato il valore del puntatore length passato come argomento.

3.3.4. Algoritmi

algorithm_A

- Argomenti:
  # *$a0* = Indirizzo del chunk contenente la stringa da criptare/decrittare.
  # *$a1* = Intero che indica quale operazione effettuare tra criptazione (+1) o decrittazione (-1).
- Return: /
- Descrizione: La procedura richiama il metodo change_ascii_value() per applicare l'algoritmo A. Il valore del registro $ra viene memorizzati e poi ripristinato per poter richiamare sotto-procedure.

algorithm_B

- Argomenti:
  # *$a0* = Indirizzo del chunk contenente la stringa da criptare/decrittare.
  # *$a1* = Intero che indica quale operazione effettuare tra criptazione (+1) o decrittazione (-1).
- Return: /
- Descrizione: La funzione richiama il metodo change_ascii_value() per applicare l'algoritmo B. Il valore del registro $ra viene memorizzati e poi ripristinato per poter richiamare sotto-procedure.

algorithm_C

- Argomenti:
  # *$a0* = Indirizzo del chunk contenente la stringa da criptare/decrittare.
  # *$a1* = Intero che indica quale operazione effettuare tra criptazione (+1) o decrittazione (-1).
- Return: /
- Descrizione: La funzione richiama il metodo change_ascii_value() per applicare l'algoritmo C. Il valore del registro $ra viene memorizzati e poi ripristinato per poter richiamare sotto-procedure.

algorithm_D

- Argomenti:
  # *$a0* = Indirizzo del chunk contenente la stringa da criptare/decrittare.
- Return: /
- Descrizione: La funzione scambia i caratteri di *$a0* appartenenti a posizioni opposte. Dato che per le specifiche del progetto il file messaggio.txt è ben formato, è possibile tradurre il ciclo while in un do-while perché il messaggio avrà almeno un carattere da criptare.

algorithm_encrypt_E

- Argomenti:
  # *$a0* = Indirizzo del chunk contenente la stringa da criptare.
- Return:
  # *$v0* = Indirizzo del chunk contenente la stringa criptata.
  # $v1 = Numero di caratteri di *$v0.*
- Descrizione: La procedura fissa un carattere di indice j nella variabile character e scorre *$a0* per ricercare possibili occorrenze. Il messaggio cifrato verrà memorizzato nel chunk allocato all'inizio della procedura. La lunghezza del blocco viene prelevata dall'array sizes_array_E ed è determinata da quante esecuzioni dell'algoritmo E sono già state effettuate. Infatti, il primo valore di sizes_array_E identifica l'indice della lunghezza da prelevare. Questa implementazione differisce dal codice C a causa della mancanza di funzioni di gestione della memoria dinamica. All'interno dei loop vengono sfruttati direttamente i registri *$v0* e *$v1* per memorizzare rispettivamente l'indirizzo del messaggio e la lunghezza di *$v0* perché i valori di ritorno di una procedura sono gli argomenti della procedura successiva, quindi le convenzioni sui registri temporanei sono garantite. A differenza dell'implementazione in C, in cui output_text contiene l'indirizzo di partenza della stringa criptata, in MIPS il suo valore viene modificato ad ogni inserimento di carattere o indice, in modo da contenere sempre l'indirizzo dell'ultimo carattere inserito. Ciò permette di non dover eseguire in ogni procedura di inserimento una istruzione di load word per ottenere il valore della lunghezza e la somma output_text + length per trovare l'indirizzo della prima posizione libera. L'incremento continuo del valore di output_text però comporta la necessità di eseguire una differenza (output_text - (length + 1)) per riportare output_text all'inizio della stringa criptata e restituire il suo valore tramite *$v0.* Inoltre, nella sezione iniziale della procedura viene decrementato l'indirizzo della stringa criptata perché altrimenti la prima istruzione della procedura insert_character non permetterebbe di inserire caratteri nel primo byte del chunk. Come nelle procedure della sezione 3.3.1, i valori dei registri $s e $ra vengono memorizzati e poi ripristinati per poter richiamare sotto-procedure. Infine, dato che per le specifiche del progetto il file messaggio.txt è ben formato, è possibile tradurre il ciclo while in un do-while perché il messaggio avrà almeno un carattere da criptare.

algorithm_decrypt_E

- Argomenti:
  # *$a0* = Indirizzo del chunk contenente la stringa da decrittare.
- Return:
  # *$v0* = Indirizzo del chunk contenente la stringa decrittata.
- Descrizione: La procedura esegue la decrittazione dell'algoritmo E della stringa criptata e segue fedelmente l'implementazione del C. Dato che per le specifiche del progetto il file messaggio.txt è ben formato, è possibile tradurre il ciclo while in un do-while perché il messaggio avrà almeno un carattere da decrittare.

3.4. Test

```
PC        = 4000b4
EPC       = 0
Cause     = 0
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 0

R0   [r0] = 0
R1   [at] = 10010000
R2   [v0] = 12
R3   [v1] = 0
R4   [a0] = 10010099
R5   [a1] = 0
R6   [a2] = 0
```

orc_f (open_f)

$a0 = 0x10010099 -> Indirizzo di partenza del nome di chiave.txt.
$a1 = 0 -> Flag di modalità lettura.
$v0 = 0x12 -> File descriptor di chiave.txt
Sinistra: registri, in basso: path del file a partire dall'indirizzo di $a0

```
[10010090]    72652067  2e726f72  65735500  672f7372    g   e r r o r . .  U s e r s / g
[100100a0]    6c656f69  6d696465  61746c69  7365442f    i o e l e d i m  i l t a / D e s
[100100b0]    706f746b  6573452f  7a696372  63732069    k t o p / E s e r c i z i   s c
[100100c0]    616c6f75  7373412f  6c626d65  74512f79    u o l a / A s s e m b l y / Q t
[100100d0]    6d697053  6f72502f  74746567  68632f6f    S p i m / P r o g e t t o / c h
[100100e0]    65766169  7478742e  65735500  672f7372    i a v e . t x t . U s e r s / g
```

```
PC        = 4000ec
EPC       = 0
Cause     = 0
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 0

R0   [r0] = 0
R1   [at] = 10010000
R2   [v0] = 1
R3   [v1] = 10040000
R4   [a0] = 12
R5   [a1] = 10040000
R6   [a2] = 5
R7   [a3] = 0
R8   [t0] = 12
R9   [t1] = 5
R10  [t2] = 10040000
```

orc_f (read_f)

$t0 = 0x12 -> File descriptor di chiave.txt
$t1 = 5 -> Numero massimo di byte da leggere dal file chiave.txt
$t2 = 0x10040000 -> Indirizzo di partenza del chunk allocato nell'Heap. L'indirizzo di partenza della memoria dinamica è proprio 0x10040000, infatti è il primo inserimento di dati che viene effettuato nell'Heap.
$v0 = 1 -> Numero di bytes effettivamente letti da file
$v1 = 0x10040000 -> Indirizzo del messaggio letto dal file chiave.txt.
A sinistra: registri, in basso: primo carattere di chiave.txt.

```
[10040000]    00000041  00000000                          A . . . . . . .
```

```
PC        = 40016c
EPC       = 0
Cause     = 0
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 0

R0   [r0] = 0
R1   [at] = 10010000
R2   [v0] = 1
R3   [v1] = 10040000
R4   [a0] = 12
```

orc_f (close_f)

Il valore di PC corrisponde all'istruzione "jr $ra" della procedura close_f.
$a0 = 0x12 -> File descriptor di chiave.txt
$v0 = 1 -> Numero di bytes effettivamente letti dal file chiave.txt
$v1 = 0x10040000 -> Indirizzo di partenza della stringa di chiave.txt
A sinistra: registri.

```
PC        = 40021c
EPC       = 400064
Cause     = 24
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 0

R0   [r0] = 0
R1   [at] = 10010000
R2   [v0] = 15
R3   [v1] = 10040008
R4   [a0] = 10040008
R5   [a1] = 1
R6   [a2] = 41
R7   [a3] = 0
R8   [t0] = 10040000
R9   [t1] = 41
R10  [t2] = 10040008
R11  [t3] = 0
R12  [t4] = 0
R13  [t5] = 0
R14  [t6] = 0
R15  [t7] = 0
R16  [s0] = 10040008
R17  [s1] = 10040000
R18  [s2] = 0
R19  [s3] = 1
```

encrypt_and_decrypt (criptazione)

$s0 = 0x10040008 -> Indirizzo della stringa di messaggio.txt. L'intervallo [0x10040000, 0x10040007] è occupato dai caratteri di chiave.txt (4 caratteri MAX + '\0' + 3 bytes per il vincolo di allineamento).
$s1 = 0x10040000 -> Indirizzo del chunk contenente i caratteri letti dal file chiave.txt
$s2 = 0 -> Contatore i. È la prima iterazione del ciclo, quindi i non è stata ancora incrementata.
$s3 = 1 -> Variabile increase usata per modificare il contatore i e per assegnare un valore a encrypt.
$t1 = 0x41 = 'A' -> key[i]
$a0 = $s0, $a1 = $s3, $a2 = $t1
A sinistra: registri.

```
PC        = 400294
EPC       = 40021c
Cause     = 24
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 0

R0   [r0] = 0
R1   [at] = 10010000
R2   [v0] = 15
R3   [v1] = 10040008
R4   [a0] = 10040008
R5   [a1] = 1
R6   [a2] = 41
R7   [a3] = 0
R8   [t0] = 10010018
R9   [t1] = 400294
```

Choose_algorithm

$a2 = 0x41 = ‘A’ -> key[i] (i = 0)
$t0 = 0x10010018 -> &(jump_table[0]), ovvero l’indirizzo della word che contiene l’indirizzo del case ‘A’.
jump_table + ($a2 - ‘A’) = 0x10010018 + 0 = 0x10010018.
$t1 = 0x400294 -> jump_table[0], ovvero l’indirizzo del case ‘A’. In questo caso, il PC corrisponde a jump_table[0] perché stiamo eseguendo la prima istruzione del case ‘A’.
$a0 = 0x10040008 -> Indirizzo del chunk contenente la stringa di messaggio.txt
$a1 = 1 -> valore di encrypt.
A sinistra: registri, in basso: indirizzi contenuti in jump_table.

```
[10010000]   00000015  00000000  00000000  00000000    . . . . . . . . . . . . . . . .
[10010010]   00000000  00000000  00400294  0040029c    . . . . . . . . . @ . . . @ . .
[10010020]   004002a4  004002ac  004002b4  63657845    . . @ . . . @ . . . @ . E x e c
```

```
PC        = 400314
EPC       = 40021c
Cause     = 24
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 8020004

R0   [r0] = 0
R1   [at] = ffffffff
R2   [v0] = 15
R3   [v1] = 10040008
R4   [a0] = 10040008
R5   [a1] = 4
R6   [a2] = ffffffff
R7   [a3] = 0
R8   [t0] = 1004001d
R9   [t1] = 58
```

Change_ascii_value

$a0 = 0x10040008 -> Indirizzo del chunk contenente la stringa di messaggio.txt
$a1 = 4 -> valore di encrypt << 2
$a2 = 0xFFFFFFFF = -1 -> valore di remainder.
$t0 = 0x1004001d -> end_text
$t1 = 0x58 = 0x54 + 4 = ’T’ + 4 = ‘X’
Prima iterazione del ciclo della procedura, quindi viene processato il primo carattere del messaggio (’T’) all’indirizzo 0x10040008.
A sinistra: registri, in basso: messaggio con il primo carattere criptato.

```
[10040000]   00000041  00000000  73696858  20736920    A . . . . . . . X h i s   i s
[10040010]   65742061  73207478  6e697274  00000067    a   t e x t   s t r i n g . . .
```

```
PC        = 4003f8
EPC       = 40031c
Cause     = 24
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 802000e

R0   [r0] = 0
R1   [at] = ffffffff
R2   [v0] = 15
R3   [v1] = 10040008
R4   [a0] = 1004001d
R5   [a1] = 4
R6   [a2] = ffffffff
R7   [a3] = 0
R8   [t0] = 1004001d
R9   [t1] = 6b
```

Algorithm_A (criptazione)

$a0 = 0x1004001d -> L’indirizzo di partenza del messaggio è stato incrementato durante il ciclo per processare ogni carattere, quindi adesso equivale al valore di end_text.
$t1 = 0x6b = 0x67 + 4 = ’g’ + 4 = ‘k’
Ogni carattere del messaggio è stato criptato con la stessa operazione di somma, infatti si può notare che i caratteri che si ripetono all’interno della stringa rimangono uguali. Ad esempio, gli spazi sono diventati ‘$’, le lettere ‘i’ sono diventate ‘m’, etc.
A sinstra: registri, in basso: messaggio criptato con l’algoritmo A.

```
[10040000]   00000041  00000000  776d6c58  24776d24    A . . . . . . . X l m w $ m w $
[10040010]   69782465  7724787c  726d7678  0000006b    e $ x i | x $ w x v m r k . . .
```

```
PC       = 4001f8
EPC      = 40031c
Cause    = 24
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 802000e

R0  [r0] = 0
R1  [at] = 10010000
R2  [v0] = 10040008
R3  [v1] = 10040008
R4  [a0] = 1001013c
R5  [a1] = 10040008
R6  [a2] = ffffffff
R7  [a3] = 0
R8  [t0] = 10040001
R9  [t1] = 0
R10 [t2] = 10040008
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 10040008
R17 [s1] = 10040000
R18 [s2] = 1
R19 [s3] = 1
```

encrypt_and_decrypt (scrittura su messaggioCifrato.txt)

$s1 = 0x10040000 -> Indirizzo del chunk contenente i caratteri letti dal file chiave.txt
$s2 = 1 -> Contatore i. È la seconda iterazione del ciclo.
$s3 = 1 -> Variabile increase usata per modificare il contatore i e per assegnare un valore a encrypt.
$t1 = \0 -> key[i]
$a0 = 0x1001013c -> Indirizzo di partenza del nome di messaggioCifrato.txt
$a1 = 0x10040008 -> Indirizzo di partenza del messaggio criptato.
A sinistra: registri, in basso: path di messaggioDecifrato.txt e messaggio cifrato.

```
[10010130]    67617373  2e6f6967  00747874  72657355    s s a g g i o . t x t . U s e r
[10010140]    69672f73  656c656f  696d6964  2f61746c    s / g i o e l e d i m i l t a /
[10010150]    6b736544  2f706f74  72657345  697a6963    D e s k t o p / E s e r c i z i
[10010160]    75637320  2f616c6f  65737341  796c626d      s c u o l a / A s s e m b l y
[10010170]    5374512f  2f6d6970  676f7250  6f747465    / Q t S p i m / P r o g e t t o
[10010180]    73656d2f  67676173  69436f69  74617266    / m e s s a g g i o C i f r a t
[10010190]    78742e6f  73550074  2f737265  656f6967    o . t x t . U s e r s / g i o e
[100101a0]    6964656c  746c696d  65442f61  6f746b73    l e d i m i l t a / D e s k t o
[100101b0]    73452f70  69637265  7320697a  6c6f7563    p / E s e r c i z i   s c u o l
[100101c0]    73412f61  626d6573  512f796c  69705374    a / A s s e m b l y / Q t S p i
[100101d0]    72502f6d  7465676f  6d2f6f74  61737365    m / P r o g e t t o / m e s s a
[100101e0]    6f696767  69636544  74617266  78742e6f    g g i o D e c i f r a t o . t x
[100101f0]    00000074  00000000  00000000  00000000    t . . . . . .
[10010200]..[1003ffff]  00000000
[10040000]    00000041  00000000  776d6c58  24776d24    A . . . . . X l m w $ m w $
[10040010]    69782465  7724787c  726d7678  0000006b    e $ x i | x $ w x v m r k . . .
```

```
PC       = 400100
EPC      = 4000fc
Cause    = 24
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 802000e

R0  [r0] = 0
R1  [at] = 10010000
R2  [v0] = 15
R3  [v1] = 10040008
R4  [a0] = e
R5  [a1] = 10040008
R6  [a2] = 15
```

owc_f (write_f)

$a0 = 0xe -> File Descriptor di messaggioCifrato.txt
$a1 = 0x10040008 -> Indirizzo di partenza del messaggio cifrato.
$a2 = 0x15 -> Numero di caratteri della stringa da scrivere sul file.
A sinitra: registri, in basso: messaggio cifrato (messaggioCifrato.txt e data segment)

```
Xlmw$mw$e$xi|x$wxvmrk
```

```
[10040000]    00000041  00000000  776d6c58  24776d24    A . . . . . X l m w $ m w $
[10040010]    69782465  7724787c  726d7678  0000006b    e $ x i | x $ w x v m r k . . .
```

```
PC       = 40021c
EPC      = 4001b0
Cause    = 24
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 802000e

R0  [r0] = 0
R1  [at] = ffff0000
R2  [v0] = 0
R3  [v1] = 10040008
R4  [a0] = 10040008
R5  [a1] = ffffffff
R6  [a2] = 41
R7  [a3] = 0
R8  [t0] = 10040000
R9  [t1] = 41
R10 [t2] = 10040008
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 10040008
R17 [s1] = 10040000
R18 [s2] = 0
R19 [s3] = ffffffff
```

encrypt_and_decrypt (decrittazione)

$s0 = 0x10040008 -> Indirizzo della stringa da decifrare
$s1 = 0x10040000 -> Indirizzo del chunk contenente i caratteri letti dal file chiave.txt
$s3 = -1 -> increase.
$s2 = 0 -> Contatore i. Durante la scrittura su file i = 1, ma è stato modificato il valore di increase da +1 a -1, perciò: i - 1 = 1 - 1 = 0
$t1 = 0x41 = 'A' -> key[i]
$a0 = $s0, $a1 = $s3, $a2 = $t1
A sinistra: registri.

Algorithm_A (decrittazione)

La stringa del messaggio è stata decifrata e corrisponde al messaggio letto all'inizio dell'esecuzione. In basso: messaggioDecifrato.txt e data segment)

```
[10040000]    00000041  00000000  73696854  20736920    A . . . . . . . T h i s   i s
[10040010]    65742061  73207478  6e697274  00000067    a   t e x t   s t r i n g . . .
```

encrypt_and_decrypt (scrittura su messaggioDecifrato.txt)

La chiamata alla procedura owc_f() ha permesso la scrittura su messaggioDecifrato.txt della stringa decifrata.

```
This is a text string
```

Algorithm_B (criptazione)

La stringa letta dal file messaggio.txt è "This is a text string" e di seguito sono riportati tutti gli indici e le codifiche in esadecimale dei singoli caratteri. Le posizioni di indice pari sono contrassegnate con il colore rosso.

54	68	69	73	20	69	73	20	61	20	74	65	78	74	20	73	74	72	69	6e	67
T	h	i	s		i	s		a		t	e	x	t		s	t	r	i	n	g
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Dopo la criptazione tramite l'algoritmo B, che modifica il valore ascii dei caratteri in posizione pari, la stringa appare come segue (In basso: messaggioCifrato.txt e data segment):

```
Xhms$iw e xe|t$sxrmnk
```

```
[10040000]    00000042  00000000  736d6858  20776924    B . . . . . . . X h m s $ i w
[10040010]    65782065  7324747c  6e6d7278  0000006b    e   x e | t $ s x r m n k . . .
```

Algorithm_B (decrittazione)

La stringa convertita risulta essere uguale alla stringa iniziale, perciò l'algoritmo di decrittazione è corretto (In basso: messaggioCifrato.txt e data segment).

```
This is a text string
```

```
[10040000]    00000042  00000000  73696854  20736920    B . . . . . . . T h i s   i s
[10040010]    65742061  73207478  6e697274  00000067    a   t e x t   s t r i n g . . .
```

Algorithm_C (criptazione)

La stringa letta dal file messaggio.txt è "This is a text string" e di seguito sono riportati tutti gli indici e le codifiche in esadecimale dei singoli caratteri. Le posizioni di indice dispari sono contrassegnate con il colore blu.

54	68	69	73	20	69	73	20	61	20	74	65	78	74	20	73	74	72	69	6e	67
T	h	i	s		i	s		a		t	e	x	t		s	t	r	i	n	g
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Dopo la criptazione tramite l'algoritmo C, che modifica il valore ascii dei caratteri in posizione dispari, la stringa appare come segue (In basso: messaggioCifrato.txt e data segment):

```
Tliw ms$a$tixx wtvirg
```

```
[10040000]  00000043  00000000  77696c54  24736d20  C . . . . . T l i w   m s $
[10040010]  69742461  77207878  72697674  00000067  a $ t i x x   w t v i r g . . .
```

Algorithm_C (decrittazione)

La stringa convertita risulta essere uguale alla stringa iniziale, perciò l'algoritmo di decrittazione è corretto (In basso: messaggioDecifrato.txt e data segment).

```
This is a text string
```

```
[10040000]  00000043  00000000  73696854  20736920  C . . . . . T h i s   i s
[10040010]  65742061  73207478  6e697274  00000067  a   t e x t   s t r i n g . . .
```

Algorithm_D (criptazione)

```
PC       = 400464
EPC      = 400460
Cause    = 24
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 10010000
R2  [v0] = 15
R3  [v1] = 10040008
R4  [a0] = 10040009
R5  [a1] = 1
R6  [a2] = 44
R7  [a3] = 0
R8  [t0] = 1004001b
R9  [t1] = 54
R10 [t2] = 67
```

La stringa letta dal file messaggio.txt è "This is a text string".
Lo stato dei registri è relativo alla prima iterazione del ciclo.
$a0 = 0x10040008 -> Indirizzo del secondo carattere della stringa di messaggio.txt
$t0 = 1004001b -> end_text
$t1 = 0x54 -> Codice ascii in esadecimale del carattere 'T'
$t2 = 0x67 -> Codice ascii in esadecimale del carattere 'g'
Il primo e l'ultimo carattere della stringa sono stati cambiati di posizione e sono stati modificati i valori di $a0 e $t0 per poter invertire, al ciclo successivo, il secondo con il penultimo carattere del messaggio.
A sinistra: registri, in basso: messaggio con primo e ultimo carattere criptati.

```
[10040000]  00000044  00000000  73696867  20736920  D . . . . . g h i s   i s
[10040010]  65742061  73207478  6e697274  00000054  a   t e x t   s t r i n T . . .
```

```
PC       = 40046c
EPC      = 400468
Cause    = 24
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 15
R3  [v1] = 10040008
R4  [a0] = 10040012
R5  [a1] = 1
R6  [a2] = 44
R7  [a3] = 0
R8  [t0] = 10040012
R9  [t1] = 20
R10 [t2] = 65
```

Il ciclo di criptazione è terminato, infatti $a0 = 0x10040012 non è più strettamente minore di $t0 = 10040012.
$t1 = 0x20 -> Codice ascii in esadecimale del carattere spazio
$t2 = 0x65 -> Codice ascii in esadecimale del carattere 'e'

La stringa ha un numero dispari di carattere, perciò nell'ultima iterazione i registri $a0 e $t0 contengono lo stesso valore, ovvero l'indirizzo del carattere che sta tra 0x20 e 0x65 (0x74 = 't').
A sinistra: registri, in basso: messaggio criptato (messaggioCifrato.txt e data segment).

```
gnirts txet a si sihT
```

```
[10040000]  00000044  00000000  72696e67  74207374  D . . . . . g n i r t s   t
[10040010]  20746578  69732061  68697320  00000054  x e t   a   s i   s i h T . . .
```

Algorithm_D (decrittazione)

Al termine dell'esecuzione del programma, la stringa risulta essere uguale al messaggio iniziale, perciò l'algoritmo di decifratura funziona correttamente.

```
This is a text string
```

```
[10040000]    00000044  00000000  73696854  20736920    D . . . . . . . T h i s   i s
[10040010]    65742061  73207478  6e697274  00000067    a   t e x t   s t r i n g . . .
```

Algorithm_encrypt_E

La stringa letta dal file messaggio.txt è "This is a text string". La criptazione del messaggio tramite l'algoritmo E fa uso di sue sotto-procedure, ovvero insert_character e insert_index.

```
PC        = 400378
EPC       = 400374
Cause     = 24
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 0

R0   [r0] = 0
R1   [at] = 10010000
R2   [v0] = 10040088
R3   [v1] = 1
R4   [a0] = 10040088
R5   [a1] = 1
R6   [a2] = 54
```

Insert_character

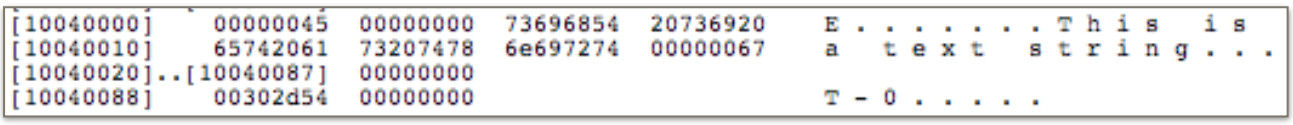
$a0 = 0x10040088 -> indirizzo dell'ultimo carattere inserito in output_text.
$a1 = 1 -> Numero di caratteri di output_text, che dopo un l'inserimento di $a2 ha lunghezza 1.
$a2 = 0x54 = 'T' -> Primo carattere del messaggio.
$v0 = 0x10040088 -> Indirizzo dell'ultimo carattere inserito in output_text.
$v1 = 1 -> Numero di caratteri di output_text.
A sinistra: registri, in basso: primo carattere del messaggio cifrato.

```
[10040000]    00000045  00000000  73696854  20736920    E . . . . . . . T h i s   i s
[10040010]    65742061  73207478  6e697274  00000067    a   t e x t   s t r i n g . . .
[10040020]..[10040087]  00000000
[10040088]    00000054  00000000                        T . . . . . . .
```

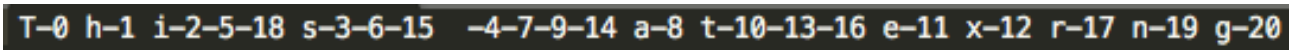
Insert_index

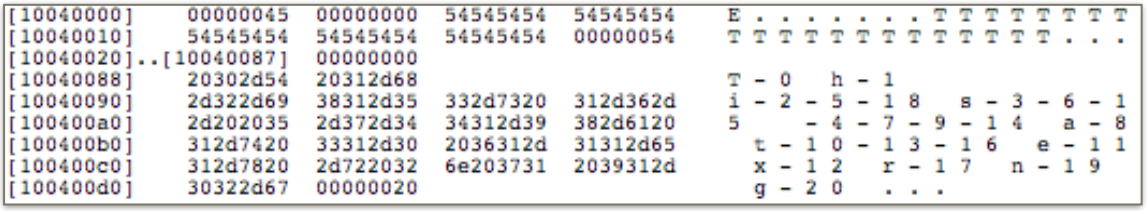
```
PC        = 400360
EPC       = 40035c
Cause     = 24
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 0

R0   [r0] = 0
R1   [at] = a
R2   [v0] = 1004008a
R3   [v1] = 3
R4   [a0] = 1004008a
R5   [a1] = 2
R6   [a2] = 0
R7   [a3] = 0
R8   [t0] = 10040089
R9   [t1] = 1
R10  [t2] = a
R11  [t3] = 30
```

$a0 = 0x1004008a -> Indirizzo dell'ultimo carattere di output_text. Prima di inserire l'indice, sono stati effettuati 2 inserimenti, uno dei quali è stato menzionato nella descrizione della procedura insert_character. L'indirizzo di partenza di output_text è 0x10040088 ('T'), il secondo byte è 0x10040089 ('-'). Il contenuto di $a0 inizialmente era 0x10040089 ed è stato modificato durante l'esecuzione della procedura per calcolare end_text.
$a1 = 2 -> Numero di caratteri di output_text prima dell'inserimento dell'indice.
$a2 = 0 -> Contiene la codifica dell'indice da inserire in output_text.
$t0 = 0x10040089 -> end_text decrementato.
$t1 = 1 -> Numero di cifre che compongono l'indice (digits_index).
$t2 = a = 10 -> Numero utilizzato per l'operazione di modulo.
$t2 = 0x30 -> Codifica dell'indice inserito in output_text.
$v0 = 0x1004008a -> Indirizzo dell'ultimo carattere di output_text dopo l'inserimento dell'indice.
$v1 = 3 -> Lunghezza di output_text dopo l'inserimento dell'indice.
A sinistra: registri, in basso: messaggio criptato parzialmente.



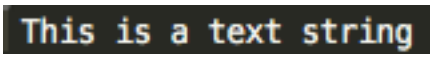
Dopo l'esecuzione dell'algoritmo di criptazione E, ogni carattere del messaggio [0x10040008, 0x1004001c] è stato sostituito con il primo carattere del messaggio stesso. Il messaggio criptato è stato memorizzato a partire dall'indirizzo 0x10040088 (in basso: messaggioCifrato.txt e data segment).

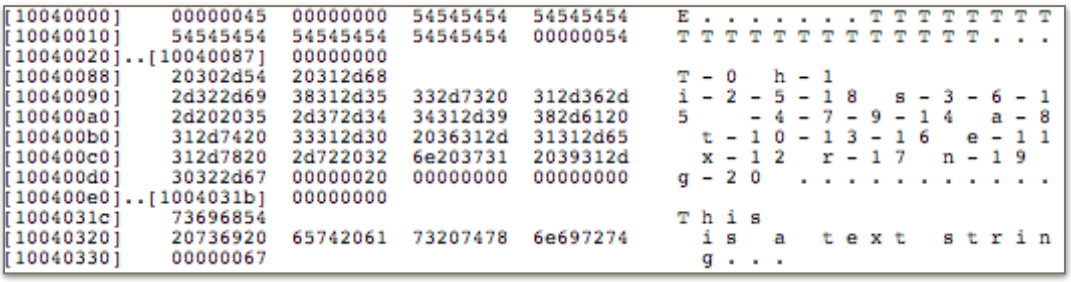




Algorithm_decrypt_E

Il valore di PC corrisponde all'istruzione "jr $ra" della procedura algorithm_decrypt_E. Dopo aver allocato un chunk di dimensione equivalente al messaggio iniziale [0x100400d8, 0x100400ec], l'algoritmo di decifratura E traduce il messaggio cifrato e riporta la stringa al suo stato iniziale.





3.5. Codice Sorgente

```
# PROGETTO MIPS ASSEMBLY PER IL CORSO DI ARCHITETTURE DEGLI ELABORATORI
# - A.A. 2018/2019 -

# Title: Messaggi Cifrati
# Author: Gioele Dimilta - Alberto Brogi
# Email: gioele.dimilta@stud.unifi.it - alberto.brogi@stud.unifi.it
# Description: Progetto MIPS Assembly - Architetture degli Elaboratori
# Input: messaggio.txt, chiave.txt
# Output: messaggioDecifrato.txt, messaggioCifrato.txt
# Filename: Progetto_AE.asm
# Date: 29/05/2019


##########################
#        INDICE          #
##########################
```

```
# open_f                        - line 120 #
# read_f                        - line 132 #
# write_f                       - line 154 #
# close_f                       - line 164 #
# orc_f                         - line 172 #
# owc_f                         - line 207 #
#                                          #
# encrypt_and_decrypt           - line 243 #
# choose_algorithm              - line 296 #
#                                          #
# change_ascii_value            - line 359 #
# insert_index                  - line 387 #
# insert_character              - line 421 #
#                                          #
# algorithm_A                   - line 440 #
# algorithm_B                   - line 457 #
# algorithm_C                   - line 473 #
# algorithm_D                   - line 489 #
# algorithm_encrypt_E           - line 509 #
# algorithm_decrypt_E           - line 619 #
# # # # # # # # # # # # # # # # # # # # # #


# # # # # # # # # # # # # # # # # # # #
#              DATA SEGMENT             #
# # # # # # # # # # # # # # # # # # # #
```

.data

```
size_message:           .word   0
sizes_arrays_E:         .word   0 657 2773 13010 67205

jump_table:             .word   L_A, L_B, L_C, L_D, L_E

file_not_found:         .asciiz "Execution terminated with errors - File not found."
sbrk_error:             .asciiz "Execution terminated with errors - Heap allocating error."

chiave:                 .asciiz "BRO-DIM-2019-05-29/chiave.txt"
messaggio:              .asciiz "BRO-DIM-2019-05-29/messaggio.txt"
messaggioCifrato:       .asciiz "BRO-DIM-2019-05-29/messaggioCifrato.txt"
messaggioDecifrato:     .asciiz "BRO-DIM-2019-05-29/messaggioDecifrato.txt"


# # # # # # # # # # # # # # # # # # # #
#             CODE SEGMENT             #
# # # # # # # # # # # # # # # # # # # #
```

.text

```
.globl main
```

main:

```
                                # # # # # # # # # # # # #
        addi $sp, $sp, -8       #   Memorizzo il valore   #
        sw $s0, 4($sp)          # dei registri permanenti #
        sw $ra, 0($sp)          #     e del registro $ra     #
                                # # # # # # # # # # # # #

        la $a0, chiave          # $a0 = &(chiave)
        li $a1, 5               # La chiave ha un massimo di 4 caratteri ma allocando 5 bytes
                                # sono sicuro che la stringa terminerà con il carattere '\0'
        jal orc_f
        move $s0, $v1           # $s0 = key
                                # $v0 = lunghezza dell'array chiave

        la $a0, messaggio       # $a0 = &(chiave)
        li $a1, 128             # Il messaggio ha un massimo di 1
                                # 28 caratteri
```

```
        jal orc_f
        sw $v0, size_message        #*size_message = $v0
                                    # $v1 = input_text

        move $a0, $v1               # $a0 = input_text
        move $a1, $s0               # $a1 = key
        jal encrypt_and_decrypt
```

end_program:

```
                                    ##############
        lw $ra, 0($sp)              # Ripristino il contenuto  #
        lw $s0, 4($sp)              # dei registri permanenti #
        addi $sp, $sp, 8            #     e del registro $ra     #
                                    ##############

        li $v0, 10                  # L'intero 10 identifica l'operazione di chiusura del programma
         syscall


######################
#          ERRORI          #
######################
```

error:

```
                                    # $a0 = Indirizzo della stringa da stampare
        li $v0, 4                   # L'intero 4 identifica l'operazione di stampa su stdin di una stringa
        syscall

        j end_program


######################
#            FILE            #
######################
```

open_f:

```
                                    # $a0 = Indirizzo di partenza del nome del file
                                    # $a1 = Flag di modalità (0 = lettura, 1 = scrittura)

        li $v0, 13                  # L'intero 13 identifica l'operazione di apertura del file
        li $a2, 0                   # Nel caso di creazione automatica di un file,
                                    # $a2 contiene i permessi da attribuire a tale file
        syscall
                                    # $v0 = File descriptor
        la $a0, file_not_found      # $a0 = Indirizzo della stringa di errore da stampare
        bltz $v0, error             # In caso di errore durante l'apertura del file, la syscall restituisce -1

        jr $ra
```

read_f:

```
        move $t0, $a0               # $t0 = File Descriptor
        move $t1, $a1               # $t1 = Numero massimo di bytes da leggere dal file

        li      $v0, 9              # L'intero 9 identifica l'operazione di allocazione di bytes nell'Heap
        move    $a0, $t1            # Numero di bytes da allocare
        syscall

        la $a0, sbrk_error          # $a0 = Indirizzo della stringa di errore da stampare
        bltz $v0, error             # Se $v0 = -1, significa che c`è stato un errore durante
                                    # l'allocazione di memoria
        move $t2, $v0               # $t2 = Indirizzo di partenza del chunk allocato nell'Heap

        li      $v0, 14             # L'intero 14 identifica l'operazione di lettura da file
        move    $a0, $t0            # $a0 = File Descriptor
```

```
move    $a1, $t2            # $a1 = Indirizzo di partenza del chunk allocato nell'Heap
move    $a2, $t1            # $a2 = Numero massimo di byte da leggere dal file
syscall
                            # $v0 = Numero di bytes effettivamente letti da file
move $v1, $t2               # $v1 = Indirizzo di partenza del messaggio letto dal file

jr $ra
```

write_f:

```
                            # $a0 = File Descriptor
                            # $a1 = Indirizzo di partenza del messaggio da scrivere nel file

li $v0, 15                  # L'intero 15 identifica l'operazione di scrittura su file
lw $a2, size_message        # Numero di bytes da estrapolare dall'Heap e scrivere nel file
syscall

jr $ra
```

close_f:

```
                            # $a0 = File Descriptor

li $v0, 16                  # L'intero 16 identifica l'operazione di chiusura del file
syscall

jr $ra
```

orc_f:

```
addi $sp, $sp, -16          # # # # # # # # # # # # #
sw $s2, 12($sp)             #   Memorizzo il valore   #
sw $s1, 8($sp)              # dei registri permanenti #
sw $s0, 4($sp)              #     e del registro $ra     #
sw $ra, 0($sp)              # # # # # # # # # # # # #

                            # $a0 = Indirizzo di partenza del nome del file
move $s0, $a1               # $a1 = Numero massimo di byte da leggere dal file

move $a1, $zero             # Flag di modalità (0 = lettura, 1 = scrittura)
jal open_f
move $s1, $v0               # Memorizzo temporaneamente il valore del File Descriptor
                            # perchè servirà più avanti per chiudere il file

move $a0, $s1               # $a0 = File Descriptor
move $a1, $s0               # $a1 = Numero massimo di byte da leggere dal file
jal read_f
move $s2, $v0               # $s2 = Numero di bytes effettivamente letti da file
move $s0, $v1               # $s0 = Indirizzo di partenza del messaggio letto dal file

move $a0, $s1               # $a0 = File Descriptor
jal close_f

move $v0, $s2               # $v0 = Numero di bytes effettivamente letti da file
move $v1, $s0               # $v1 = Indirizzo di partenza del messaggio letto dal file

lw $ra, 0($sp)              # # # # # # # # # # # # #
lw $s0, 4($sp)              # Ripristino il contenuto #
lw $s1, 8($sp)              # dei registri permanenti #
lw $s2, 12($sp)             #     e del registro $ra     #
addi $sp, $sp, 16           # # # # # # # # # # # # #

jr $ra
```

owc_f:

```
        addi $sp, $sp, -12          ##############
        sw $s1, 8($sp)              #  Memorizzo il valore  #
        sw $s0, 4($sp)              # dei registri permanenti #
        sw $ra, 0($sp)              #    e del registro $ra    #
                                    ##############

                                    # $a0 = Indirizzo di partenza del nome del file
        move $s0, $a1               # $s0 = Indirizzo di partenza del messaggio memorizzato nell'Heap

        li $a1, 1                   # Flag di modalità (0 = lettura, 1 = scrittura)
        jal open_f
        move $s1, $v0               # Memorizzo temporaneamente il valore del File Descriptor
                                    # perchè servirà più avanti per chiudere il file.

        move $a0, $s1               # $a0 = File Descriptor
        move $a1, $s0               # $a1 = Indirizzo di partenza del messaggio da scrivere nel file
        jal write_f

        move $a0, $s1               # $a0 = File Descriptor
        jal close_f

                                    ##############
        lw $ra, 0($sp)              # Ripristino il contenuto #
        lw $s0, 4($sp)              # dei registri permanenti #
        lw $s1, 8($sp)              #    e del registro $ra    #
        addi $sp, $sp, 12           ##############

        jr $ra


######################
#      GESTIONE      #
######################
```

encrypt_and_decrypt:

```
        addi $sp, $sp, -20          ##############
        sw $s3, 16($sp)             #              #
        sw $s2, 12($sp)             #  Memorizzo il valore  #
        sw $s1, 8($sp)              # dei registri permanenti #
        sw $s0, 4($sp)              #    e del registro $ra    #
        sw $ra, 0($sp)              #              #
                                    ##############

        move $s0, $a0               # $s0 = input_text
        move $s1, $a1               # $s1 = key
        move $s2, $zero             # $s2 = i
        li $s3, 1                   # $s3 = increase

        loop__encrypt_and_decrypt:
                add $t0, $s1, $s2               # $t0 = key + i
                lb $t1, 0($t0)                  # $t0 = key[i]
                bnez $t1, else__encrypt_and_decrypt   # IF (key[i] != '\0')

                la $a0, messaggioCifrato        # $a0 = &(messaggioCifrato)
                move $a1, $s0                   # $a1 = input_text
                jal owc_f

                li $s3, -1                      # increase = -1
                add $s2, $s2, $s3               # i = i + increase

                else__encrypt_and_decrypt:
                add $t0, $s1, $s2               # $t0 = key + i
                lb $t1, 0($t0)                  # $t1 = key[i]
```

```
        move $a0, $s0                       # $a0 = input_text
        move $a1, $s3                       # $a1 = encrypt
        move $a2, $t1                       # $a2 = key[i]
        jal choose_algorithm
        move $s0, $v0                       # $v0 = input_text

        add $s2, $s2, $s3                   # i = i + increase
        bgez $s2, loop__encrypt_and_decrypt # while(i >= 0)

la $a0, messaggioDecifrato                  # $a0 = &(messaggioDecifrato)
move $a1, $s0                               # $a1 = input_text
jal owc_f

                                            # # # # # # # # # # # # # #
lw $ra, 0($sp)                              #                         #
lw $s0, 4($sp)                              # Ripristino il contenuto #
lw $s1, 8($sp)                              # dei registri permanenti #
lw $s2, 12($sp)                             #     e del registro $ra  #
lw $s3, 16($sp)                             #                         #
addi $sp, $sp, 20                           # # # # # # # # # # # # # #


jr $ra
```

choose_algorithm:

```
                                            # # # # # # # # # # # # # #
addi $sp, $sp, -8                           #   Memorizzo il valore   #
sw $s0, 4($sp)                              # dei registri permanenti #
sw $ra, 0($sp)                              #     e del registro $ra    #
                                            # # # # # # # # # # # # # #

                                            # $a0 = input_text
                                            # $a1 = encrypt
                                            # $a2 = character

blt $a2, 'A', exit_switch                   # IF (character < 'A')
bgt $a2, 'E', exit_switch                   # IF (character > 'E')

addi $t0, $a2, -0x41                        # $t0 = character - 'A'
sll $t0, $t0, 2                             # $t0 = (character - 'A') * 4 (without overflow)
la $t1, jump_table                          # $t1 = &(jump_table)
add $t0, $t1, $t0                           # $t0 = ((character - 'A') * 4) + &(jump_table)
lw $t1, 0($t0)                              # $t1 = jump_table[(character - 'A')]

move $s0, $a0                               # $s0 = input_text
jr $t1

L_A:                                        # L_A = jump_table[0]
        jal algorithm_A
        j exit_switch
L_B:                                        # L_B = jump_table[1]
        jal algorithm_B
        j exit_switch
L_C:                                        # L_C = jump_table[2]
        jal algorithm_C
        j exit_switch
L_D:                                        # L_D = jump_table[3]
        jal algorithm_D
        j exit_switch
L_E:                                        # L_E = jump_table[4]
        bltz $a1, L_E_decrypt               # IF (encrypt < 0)
        jal algorithm_encrypt_E
        j end_L_E
```

```
            L_E_decrypt:
                jal algorithm_decrypt_E

            end_L_E:
                move $s0, $v0                   # $s0 = input_text

        exit_switch:
        move $v0, $s0                           # $v0 = input_text

                                                # # # # # # # # # # # # #
        lw $ra, 0($sp)                          # Ripristino il contenuto #
        lw $s0, 4($sp)                          # dei registri permanenti #
        addi $sp, $sp, 8                        #     e del registro $ra     #
                                                # # # # # # # # # # # # #

        jr $ra



# # # # # # # # # # # # # # # # # # # #
#               CALCOLO                 #
# # # # # # # # # # # # # # # # # # # #
```

***change_ascii_value**:*

```
                                                # $a0 = input_text
                                                # $a1 = encrypt
                                                # $a2 = reminder

        lw $t0, size_message                    # $t0 = *size_message
        add $t0, $a0, $t0                       # end_text = input_text + *size_message
        sll $a1, $a1, 2                         # encrypt = encrypt * 4

        loop__change_ascii_value:
                li $t1, 2                       # $t1 = 2
                div $a0, $t1                    # input_text / 2
                mfhi $t1                        # $t1 = input_text % 2

                beq $t1, $a2, true_if__change_ascii_value       # IF ((input_text % 2) == reminder)
                bne $a2, -1, end_loop__change_ascii_value       # IF (remainder != -1)

                true_if__change_ascii_value:
                lbu $t1, 0($a0)                 # $t1 = *input_text
                add $t1, $t1, $a1               # $t1 = $t1 + encrypt
                sb $t1, 0($a0)                  # *input_text = $t1

    end_loop__change_ascii_value:
    addi $a0, $a0, 1                            # input_text = input_text + 1
    bne $a0, $t0, loop__change_ascii_value      # IF (input_text != end_text)

        jr $ra
```

***insert_index**:*

```
                                                # $a0 = &(output_text[length-1])
                                                # $a1 = length
                                                # $a2 = index

        move $t0, $a2                           # temp = index
        move $t1, $zero                         # digits_index = 0

        count_digits__insert_index:
                div $t0, $t0, 10                # temp = temp / 10;
                add $t1, $t1, 1                 # digits_index = digits_index + 1
                bnez $t0, count_digits__insert_index    # WHILE (temp != 0)
```

```
        add     $a0, $a0, $t1                   # $a0 = &(output_text[length-1]) + digits_index =
                                                # &(output_text[(length-1) + digits_index])
        move $t0, $a0                           # end_text = &(output_text[(length-1) +
                                                # digits_index])

        li      $t2, 10                         # $t2 = 10

        loop__insert_index:
                div $a2, $t2                    # index / 10
                mflo $a2                        # index = index / 10
                mfhi $t3                        # $t3 = index % 10

                add $t3, $t3, '0'               # $t3 = (index % 10) + '0'
                sb $t3, 0($t0)                  # *output_text = (index % 10) + '0'

                sub $t0, $t0, 1                 # end_text = end_text - 1
                bnez $a2, loop__insert_index    # WHILE (index != 0)

        move $v0, $a0                           # $v0 = &(output_text[(length-1) + digits_index])
        add $v1, $a1, $t1                       # $v1 = length + digits_index

        jr $ra
```

insert_character:

```
                                                # $a0 = &(output_text[length-1])
                                                # $a1 = length
                                                # $a2 = character

        addi $a0, $a0, 1                        # $a0 = &(output_text[(length-1) + 1]) =
                                                # &(output_text[length])

        sb $a2, 0($a0)                          # output_text[length] = character
        addi $a1, $a1, 1                        # length = length + 1

        move $v0, $a0                           # $v0 = output_text[length - 1]
        move $v1, $a1                           # $v1 = length

        jr $ra
```

```
####################
#    ALGORITMI     #
####################
```

algorithm_A:

```
                                                ##############
        addi $sp, $sp, -4                       #  Memorizzo il valore  #
        sw $ra, 0($sp)                          #     del registro $ra.    #
                                                ##############

                                                # $a0 = input_text
                                                # $a1 = encrypt
        li $a2, -1                              # $a2 = reminder
        jal change_ascii_value

                                                ##############
        lw $ra, 0($sp)                          # Ripristino il contenuto #
        addi $sp, $sp, 4                        #     del registro $ra.    #
                                                ##############
    jr $ra
```

algorithm_B:

```
                                # # # # # # # # # # # # #
addi $sp, $sp, -4               #  Memorizzo il valore  #
sw $ra, 0($sp)                  #     del registro $ra.    #
                                # # # # # # # # # # # # #

                                # $a0 = input_text
                                # $a1 = encrypt
li $a2, 0                       # $a2 = reminder
jal change_ascii_value
                                # # # # # # # # # # # # #
lw $ra, 0($sp)                  # Ripristino il contenuto #
addi $sp, $sp, 4                #     del registro $ra.    #
                                # # # # # # # # # # # # #
jr $ra
```

algorithm_C:

```
                                # # # # # # # # # # # # #
addi $sp, $sp, -4               #  Memorizzo il valore  #
sw $ra, 0($sp)                  #     del registro $ra.    #
                                # # # # # # # # # # # # #

                                # $a0 = input_text
                                # $a1 = encrypt
li $a2, 1                       # $a2 = reminder
jal change_ascii_value
                                # # # # # # # # # # # # #
lw $ra, 0($sp)                  # Ripristino il contenuto #
addi $sp, $sp, 4                #     del registro $ra.    #
                                # # # # # # # # # # # # #
jr $ra
```

algorithm_D:

```
                                # $a0 = input_text

lw $t0, size_message            # $t0 = *size_message
addi $t0, $t0, -1               # $t0 = *size_message - 1
add $t0, $a0, $t0               # end_text = input_text + (*size_message - 1)

loop__D:
        lbu $t1, 0($a0)         # temp = *input_text
        lbu $t2, 0($t0)         # $t2 = *end_text

        sb $t2, 0($a0)          # *input_text = *end_text
        sb $t1, 0($t0)          # *end_text = temp

        addi $a0, $a0, 1        # input_text = input_text + 1
        addi $t0, $t0, -1       # end_text = end_text - 1
        blt $a0, $t0, loop__D   # IF (input_text < end_text)

jr $ra
```

algorithm_encrypt_E:

```
addi $sp, $sp, -24              # # # # # # # # # # # # #
sw $s4, 20($sp)                 #                       #
sw $s3, 16($sp)                 #  Memorizzo il valore  #
sw $s2, 12($sp)                 # dei registri permanenti #
sw $s1, 8($sp)                  #    e del registro $ra.  #
sw $s0, 4($sp)                  #                       #
```

```
sw $ra, 0(sp)                                  # # # # # # # # # # # # #

                                               # $a0 = input_text

move $s0, $a0                                  # $s0 = input_text
move $s1, $zero                                # $s1 = i
move $s2, $zero                                # $s2 = j
lw $s3, size_message                           # $s3 = *size_message
move $s4, $zero                                # $s4 = character
move $v1, $zero                                # $v1 = length

la $t0, sizes_arrays_E                         # $t0 = sizes_arrays_E
lw $t1, 0($t0)                                 # $t1 = sizes_arrays_E[0]
addi $t1, $t1, 1                               # $t1 = sizes_arrays_E[0] + 1
sll $t1, $t1, 2                                # $t1 = (sizes_arrays_E[0] + 1) * 4
add $t1, $t0, $t1                              # $t1 = sizes_arrays_E +
                                               # ((size_arrays_E[0] + 1) * 4) =
                                               # sizes_arrays_E[size_arrays_E[0] + 1]
lw $t2, 0($t1)                                 # $t2 = sizes_arrays_E[size_arrays_E[0]+1]

li $v0, 9                                      # L'intero 9 identifica l'operazione di
                                               # allocazione di bytes nell'Heap
move $a0, $t2                                  # Numero di bytes da allocare
syscall

la $a0, sbrk_error                             # $a0 = Indirizzo della stringa di errore da
                                               # stampare
bltz $v0, error                                # Se $v0 = -1, significa che c`è stato un
                                               # errore durante l'allocazione di memoria

addi $v0, $v0, -1                              # output_text = output_text - 1

loop_1__encrypt_E:
        lbu $t0, 0($s0)                        # $t0 = input_text[0]
        add $t1, $s0, $s2                      # $t1 = &(input_text[j])
        lb $s4, 0($t1)                         # character = input_text[j]

        beqz $s2, true_if__encrypt_E           # IF (j == 0)
        beq $s4, $t0, end_loop_1__encrypt_E    # IF (character == input_text[0])

        true_if__encrypt_E:
        move $a0, $v0                          # $a0 = output_text
        move $a1, $v1                          # $a1 = length
        move $a2, $s4                          # $a2 = character
        jal insert_character

        move $s1, $s2                          # i = j

        loop_2__encrypt_E:
                add $t0, $s0, $s1              # $t0 = &(input_text[i])
                lb $t1, 0($t0)                 # $t1 = input_text[i]
                bne $s4, $t1, end_loop_2__encrypt_E   # IF (character != input_text[i])

                move $a0, $v0                  # $a0 = output_text
                move $a1, $v1                  # $a1 = length
                li $a2, '-'                    # $a2 = '-'
                jal insert_character

                move $a0, $v0                  # $a0 = output_text
                move $a1, $v1                  # $a1 = length
                move $a2, $s1                  # $a2 = i
                jal insert_index

                lbu $t0, 0($s0)                # $t0 = input_text[0]
                add $t1, $s0, $s1              # $t1 = &(input_text[i])
```

```
                sb $t0, 0($t1)                  # input_text[i] = input_text[0]

        end_loop_2__encrypt_E:
                addi $s1, $s1, 1                # i = i + 1
                blt $s1, $s3, loop_2__encrypt_E # IF (i < *size_message)

                move $a0, $v0                   # $a0 = output_text
                move $a1, $v1                   # $a1 = length
                li $a2, ' '                     # $a2 = '-'
                jal insert_character

end_loop_1__encrypt_E:
        addi $s2, $s2, 1                        # j = j + 1
        blt $s2, $s3, loop_1__encrypt_E         # IF (j < *size_message)

la $t0, sizes_arrays_E                          # $t0 = size_arrays_E
lw $t1, 0($t0)                                  # $t1 = size_arrays_E[0]
add $t1, $t1, 1                                 # $t1 = size_arrays_E[0] + 1
sw $t1, 0($t0)                                  # size_arrays_E[0] = $t1

sll $t1, $t1, 2                                 # $t1 = size_arrays_E[0] * 4
add $t0, $t0, $t1                               # sizes_arrays_E = sizes_arrays_E + $t1
sw $s3, 0($t0)                                  # sizes_arrays_E + $t1 = size_message

la $t0, size_message                            # $t0 = &(size_message)

addi $t1, $v1, -1                               # $t1 = length - 1
sw $t1, 0($t0)                                  # *size_message = length - 1

sub $v0, $v0, $v1                               # output_text = output_text - length
addi $v0, $v0, 1                                # output_text = output_text + 1


lw $ra, 0($sp)                                  # # # # # # # # # # # # # # #
lw $s0, 4($sp)                                  #                         #
lw $s1, 8($sp)                                  # Ripristino il contenuto #
lw $s2, 12($sp)                                 # dei registri permanenti #
lw $s3, 16($sp)                                 #     e del registro $ra. #
lw $s4, 20($sp)                                 #                         #
addi $sp, $sp, 24                               # # # # # # # # # # # # # # #

jr $ra
```

algorithm_decrypt_E:

```
                                                # $a0 = input_text

move $t0, $a0                                   # $t0 = input_text
li $t1, 1                                       # $t1 = i
lw $t2, size_message                            # $t2 = *size_message
li $t3, 0                                       # $t3 = index

la $t4, sizes_arrays_E                          # $t4 = &(sizes_arrays_E)
lw $t5, 0($t4)                                  # $t5 = sizes_arrays_E[0]
sll $t5, $t5, 2                                 # $t5 = sizes_arrays_E[0] * 4
add $t4, $t4, $t5                               # $t4 = &(sizes_arrays_E) +
                                                # (sizes_arrays_E[0] * 4)
lw $t5, 0($t4)                                  # $t5 = sizes_arrays_E[sizes_arrays_E[0]]

li $v0, 9                                       # L'intero 9 identifica l'operazione di
                                                # allocazione di bytes nell'Heap
move $a0, $t5                                   # Numero bytes da allocare
syscall

la $a0, sbrk_error                              # $a0 = Indirizzo della stringa di errore da
```

```
                                        # stampare
bltz $v0, error                         # Se $v0 = -1, significa che c`è stato un
                                        # errore durante l'allocazione di memoria
move $t4, $v0                           # $t4 = output_text

lbu $t5, 0($t0)                         # character = input_text[0]
sb $t5, 0($t4)                          # output_text[0] = character


loop__decrypt_E:
        add $t6, $t0, $t1               # $t6 = input_text + i
        addi $t7, $t6, -1               # $t7 = (input_text + i) - 1
        lbu $t8, 0($t7)                 # $t8 = input_text[i-1]
        bne $t8, ' ', else_if__decrypt_E # IF (input_text[i-1] != ' ')
        addi $t7, $t6, 1                # $t7 = (input_text + i) + 1
        lbu $t8, 0($t7)                 # $t8 = input_text[i+1]
        bne $t8, '-', else_if__decrypt_E # IF (input_text[i+1] != '-')

        lbu $t5, 0($t6)                 # $t5 = input_text[i]
        j end_loop__decrypt_E

        else_if__decrypt_E:
                lbu $t7, 0($t6)         # $t7 = input_text[i]

                beq $t7, ' ', if_true__decrypt_E # IF (input_text[i] == ' ')
                bne $t7, '-', else__decrypt_E    # IF (input_text[i] != '-') != 0)

                if_true__decrypt_E:
                beqz $t3, end_loop__decrypt_E    # IF (index == 0)

                add $t7, $t4, $t3       # $t7 = output_text + index
                sb $t5, 0($t7)          # output_text[index] = character
                move $t3, $zero         # index = 0
                j end_loop__decrypt_E

        else__decrypt_E:
                lbu $t7, 0($t6)         # $t7 = input_text[i]
                addi $t7, $t7, -48      # $t7 = input_text[i] - '0'
                mul $t8, $t3, 10        # $t8 = index * 10
                add $t3, $t8, $t7       # index = (index * 10) +
                                        # (input_text[i] - '0')


end_loop__decrypt_E:
        addi $t1, $t1, 1                # i = i + 1
        blt $t1, $t2, loop__decrypt_E   # IF (i < *size_message)

add $t0, $t4, $t3                       # $t0 = output_text + index
sb $t5, 0($t0)                          # output_text[index] = character;

la $t0, sizes_arrays_E                  # $t0 = &(size_message)
lw $t1, 0($t0)                          # $t1 = size_message[0]
sll $t2, $t1, 2                         # $t2 = size_message[0] * 4
add $t2, $t0, $t2                       # $t2 = &(size_message) +
                                        # (size_message[0] * 4)
lw $t3, 0($t2)                          # $t3 = sizes_arrays_E[sizes_arrays_E[0]]
sw $t3, size_message                    # *size_message =
                                        # sizes_arrays_E[sizes_arrays_E[0]]


addi $t1, $t1, -1                       # $t1 = sizes_arrays_E[0] - 1;
sw $t1, 0($t0)                          # sizes_arrays_E[0] = sizes_arrays_E[0] - 1

move $v0, $t4                           # $v0 = output_text

jr $ra
```