RELAZIONE PROGETTO

Data di consegna: 29/05/2019

Autori:

- Bruno Andreuccetti, bruno.andreuccetti@stud.unifi.it

Descrizione della soluzione adottata

Per realizzare il progetto ho sviluppato alcune procedure, il meno possibili ripetitive, per eseguire le funzioni principali richieste: leggere il messaggio, leggere la chiave di cifratura, cifrare la stringa con gli algoritmi specificati, decifrare la stringa cifrata e scrivere i file di output.

La procedura ***main***, pertanto, si limita a richiamare le varie procedure che ho realizzato, nelle modalità e nell'ordine corretti.

Per salvare i dati utilizzati ho usato vari spazi di memoria:

- ***bufferMessage***, di 200.000 byte, nel quale viene salvata la stringa via via che viene cifrata e poi successivamente decifrata;
- ***bufferKey***, nel quale viene salvata la chiave di cifratura
- ***bufferMessageSupport***, di 200.000 byte, che viene utilizzato di appoggio per l'algoritmo E
- ***jumpTable***, di 20 byte ossia 5 word, nel quale viene salvata la jump table per lo switch degli algoritmi
- ***bufferCifre***, di 10 byte, utilizzato per gestire e salvare alcune cifre dell'algoritmo E

Le procedure sviluppate sono le seguenti:

- ***main***: Richiama le altre procedure con le giuste modalità e nel giusto ordine per far sì che il programma compia il suo lavoro correttamente
- ***switch***: Crea la jump table per lo switch delle procedure, scorre la chiave da destra a sinistra in base a se deve cifrare o decifrare (in base al valore di *$a0*) e per ogni lettera della chiave esegue lo switch per chiamare la corretta procedura di cifratura o decifratura
- **dimensioneBuffer:** Restituisce l'indice dell'ultimo elemento dello spazio di memoria che gli viene passato in *$a0*

- ***letturaFile***: Viene utilizzata per leggere la chiave e il messaggio da cifrare. La procedura legge il file indicato in *$a0* e salva nel buffer di memoria in base al valore di *$a1:*
    - 0: bufferKey
    - 1: bufferMessage
- **scritturaFile:** Scrive il contenuto di bufferMessage nel file il cui nome viene passato in *$a0*
- ***algA***: Questa procedura si occupa del funzionamento dell'algoritmo A sia per cifratura che per la decifratura, in base al valore passato in *$a0:*
    - 4: cifra
    - -4: decifra
- ***algBC***: Questa procedura si occupa del funzionamento degli algoritmi B e C, sia per cifratura che per la decifratura, in base al valore passato in *$a0:*
    - 4: cifra
    - -4: decifra

  in base al valore passato in *$a1*:

    - 0: usa algoritmo B
    - 1: usa algoritmo C
- **algD:** Questa procedura si occupa del funzionamento dell'algoritmo A sia per cifratura che per la decifratura, che funziona allo stesso modo in entrambi i casi, senza bisogno quindi di dati in input
- **algCifraturaE:** Questa procedura si occupa del funzionamento dell'algoritmo E solo per la cifratura
- **algDecifraturaE:** Questa procedura si occupa del funzionamento dell'algoritmo E solo per la decifratura

Ho principalmente utilizzato i registri **t** all'interno delle procedure, poiché avevo bisogno di variabili temporanee, come per esempio per i contatori.

Talvolta ho usato i registri **s** nelle procedure per mantenere dati più duraturi e per ovviare alla mancanza di registri **t**.

Mi sono avvalso anche dei registri **a**,**v** e **$ra** per rispettare le convenzioni sulle chiamate alle procedure e per le syscall.

Ho usufruito dello stack per preservare correttamente i registri **s** e salvare **$ra**, utilizzando quindi il registro **$sp** (puntatore allo stack).

Per salvare i dati a lungo termine ho utilizzato solo memoria statica.

Scelte implementative:

Ho deciso di avvalermi di un'unica procedura per la lettura dei due file dato che, seppur diversi tra loro, ho ritenuto non fosse molto differente l'implementazione della stessa.

Ho utilizzato una sola procedura sia per lo switch delle procedure di cifratura che per lo switch delle procedure di decifratura, che decide se cifrare o decifrare in base ad un parametro che gli viene passato, iniziando a leggere la chiave da sinistra o da destra in base ad esso.

Per quanto riguarda l'algoritmo A ho creato una procedura che facesse sia cifratura che decifratura, in base ad un parametro dato.

Invece ho unito gli algoritmi B e C in un'unica procedura per entrambi, che ha due parametri di ingresso per fargli capire quale dei due algoritmi utilizzare e se deve cifrare/decifrare.

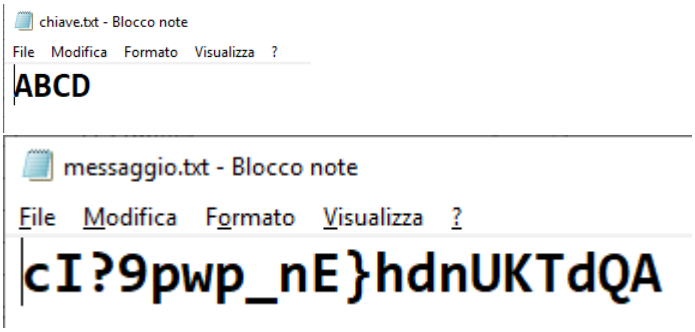
L'algoritmo D è uguale nel funzionamento sia in cifratura che decifratura, pertanto ho creato un'unica procedura che non richiede parametri di ingresso.

L'algoritmo E è suddiviso in due procedure, una per cifratura e una per decifratura.

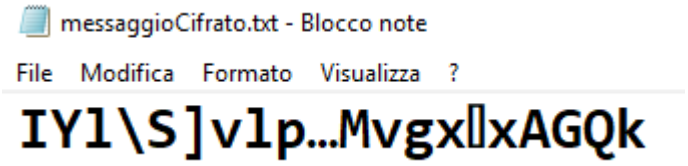
Test di corretto funzionamento

Per verificare la correttezza del mio codice, ho creato un programma in un linguaggio di codice ad alto livello che genera file di testo di input randomici (coerenti con il testo e sensati allo scopo di verificare il maggior numero di diversi possibili valori di ingresso) e controlla poi la correttezza dei 2 file di output generati dal programma assembly.
All'interno dell'archivio, come indicato dalle modalità di consegna, sono stati inclusi i file di input e output di uno dei tanti test eseguiti. Riporto qua sotto le foto di due test che includono tutti gli algoritmi di cifratura

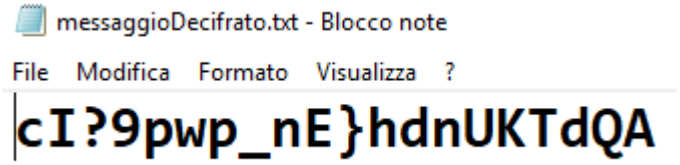
FILE DI INPUT



FILE CON LA STRINGA CIFRATA



FILE CON LA STRINGA DECIFRATA



CODICE

```
# Progetto di Assembly realizzato da Bruno Andreuccetti
# bruno.andreuccetti@stud.unifi.it
# Data consegna: 29/05/2019

.data
jumpTable:  .space 20           # Spazio di memoria riservato alla Jump Table dello switch
bufferKey:    .space 5          # Spazio di memoria riservato per salvare la chiave
bufferCifre:  .space 10         # Spazio di memoria riservato per salvare alcune cifre durante il funzionamento dell'algoritmo E
bufferMessage:          .space 200000   # Spazio di memoria riservato per salvare la stringa
bufferMessageSupport:   .space 200000   # Spazio di memoria riservato per il corretto funzionamento dell'algoritmo E

fnf:                    .asciiz  "\nThe file was not found: "
fileINmessaggio:    .asciiz "messaggio.txt"
fileINchiave:       .asciiz "chiave.txt"
fileOUTcifrato:             .asciiz "messaggioCifrato.txt"
fileOUTdecifrato:   .asciiz "messaggioDecifrato.txt"

.text
.globl main  # Il programma comincia richiamando la procedura main


main:    # Procedura main

        addi $sp, $sp, -4    # Posizionamento dello stack pointer per poter fare un push
        sw $ra, 0($sp)              # Salvataggio del precedente $ra nello stack per poterlo ripristinare a fine procedura
```

```
        la $a0, fileINmessaggio   # Nome del file che contiene il messaggio
        li $a1, 0                 # Identificatore che deve salvare in bufferMessage
        jal letturaFile           # Chiamata della procedura per leggere il file indicato


        la $a0, fileINchiave      # Nome del file che contiene la chiave
        li $a1, 1                 # Identificatore che deve salvare in bufferKey
        jal letturaFile           # Chiamata della procedura per leggere il file indicato

        li $a0, 1                 # Indico che voglio cifrare il messaggio
        jal switch                # Chiamo procedura che switcha la chiave per cifrare

        la $a0, fileOUTcifrato    # Nome del file in cui scrivere il messaggio cifrato
        jal scritturaFile         # Chiamata della procedura per scrivere il messaggio
decifrato

        li $a0, -1                # Indico che voglio decifrare il messaggio
        jal switch                # Chiamo procedura che switcha la chiave per decifrare

        la $a0, fileOUTdecifrato  # Nome del file in cui scrivere il messaggio decifrato
        jal scritturaFile         # Chiamata della procedura per scrivere il messaggio
decifrato


        lw $ra, 0($sp)            # Ripristino del vecchio $ra dallo stack
        addi $sp, $sp, 4          # Risistemazione dello stack pointer dopo aver estratto un
dato

#done: li $v0, 10          #per mars
#syscall            #per mars
jr $ra                  #per qtspim # Termine della procedura main
```

PROCEDURE

Procedura che fa lo switch di ogni elemento della chiave
per chiamare nel giusto ordine gli algoritmi
di cifratura e decifratura del messaggio

switch:
	addi $sp, $sp, -4	# Posizionamento dello stack pointer per poter fare un push
	sw $ra, 0($sp)		# Salvataggio di $ra nello stack per poterlo ripristinare a fine procedura

	addi $sp, $sp, -4	# Posizionamento dello stack pointer per poter fare un push
	sw $s0, 0($sp)		# Salvataggio del precedente $s0 nello stack per poterlo ripristinare a fine procedura
	addi $sp, $sp, -4	# Posizionamento dello stack pointer per poter fare un push
	sw $s1, 0($sp)		# Salvataggio del precedente $s1 nello stack per poterlo ripristinare a fine procedura
	addi $sp, $sp, -4	# Posizionamento dello stack pointer per poter fare un push
	sw $s2, 0($sp)		# Salvataggio del precedente $s2 nello stack per poterlo ripristinare a fine procedura

	move $s2, $a0		# Mi copio in $s2 il valore che mi dice se devo cifrare o decifrare

	# Creazione della jump table con i 5 casi relativi ai 5 algoritmi
	la $t0, SwitchAlgA
	li $t1, 0
	sw $t0, jumpTable($t1)
	la $t0, SwitchAlgB
	addi $t1, $t1, 4
	sw $t0, jumpTable($t1)

```
la $t0, SwitchAlgC
addi $t1, $t1, 4
sw $t0, jumpTable($t1)
la $t0, SwitchAlgD
addi $t1, $t1, 4
sw $t0, jumpTable($t1)
la $t0, SwitchAlgE
addi $t1, $t1, 4
sw $t0, jumpTable($t1)



beq $s2, 1, switchCifratura
# Caso decifratura: leggo la chiave da destra a sinistra
    la $a0, bufferKey
    jal dimensioneBuffer      # Prelevo il numero di elementi presenti nella chiave
    move $s1, $v0             # Il contatore del buffer della stringa viene posizionato sull'ultimo elemento della chiave
    j forStringaChiave

switchCifratura:    # Caso decifratura: leggo la chiave da sinistra a destra
    li $s1, 0       # Il contatore del buffer della stringa viene posizionato sul primo elemento della chiave

forStringaChiave:  # Ciclo di tutti i caratteri della chiave
    lb $s0, bufferKey($s1)              # Carattere attuale da elaborare
    beq $s0, $zero, FineSwitch          # Controllo fine della stringa e del ciclo
    add $s1, $s1, $s2                   # Incremento/Decremento il contatore del buffer per passare al valore successivo

    li $t9, 4               # Devo moltiplicare per 4 per saltare alla giusta posizione della Jump Table
```

```
        li $t8, 65              # Valore da sottrarre per trasformare le lettere A/B/C/D/E in 0/1/2/3/4
        sub $s0, $s0, $t8       # Calcolo dell'indice della Jump Table per reperire l'indirizzo a cui saltare
        mul $s0, $s0, $t9

        lw $t4, jumpTable($s0)  # Prendo dalla Jump Table l'indirizzo a cui devo saltare
        jr $t4

    SwitchAlgA:
        mul $a0, $s2, 4     # Imposto se voglio cifrare o decifrare in base a quel che ho in $s2
        jal algA        # Chiamo la procedura per cifrare o decifrare con algoritmo A
    j forStringaChiave  # Iterazione successiva

    SwitchAlgB:
        mul $a0, $s2, 4     # Imposto se voglio cifrare o decifrare in base a quel che ho in $s2
        li $a1, 0       # Imposto che voglio utilizzare l'algoritmo B
        jal algBC       # Chiamo la procedura per cifrare o decifrare con algoritmo B
    j forStringaChiave  # Iterazione successiva

    SwitchAlgC:
        mul $a0, $s2, 4     # Imposto se voglio cifrare o decifrare in base a quel che ho in $s2
        li $a1, 1       # Imposto che voglio utilizzare l'algoritmo C
        jal algBC       # Chiamo la procedura per cifrare o decifrare con algoritmo C
    j forStringaChiave  # Iterazione successiva

    SwitchAlgD:
```

```
            jal algD        # Chiamo la procedura per cifrare o decifrare con algoritmo
D
        j forStringaChiave  # Iterazione successiva

        SwitchAlgE:
            beq $s2, 1, SwitchAlgECif       # Controllo se devo chiamare la cifratura
o la decifratura
                jal algDecifraturaE # Cifratura con algoritmo E
                j forStringaChiave
            SwitchAlgECif:
                jal algCifraturaE   # Decifratura con algoritmo E
        j forStringaChiave  # Iterazione successiva

        FineSwitch:
            lw $s2, 0($sp)              # Ripristino del vecchio $s2 dallo stack
            addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver
estratto un dato
            lw $s1, 0($sp)              # Ripristino del vecchio $s1 dallo stack
            addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver
estratto un dato
            lw $s0, 0($sp)              # Ripristino del vecchio $s0 dallo stack
            addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver
estratto un dato

            lw $ra, 0($sp)              # Ripristino del vecchio $ra dallo stack
            addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver
estratto un dato
jr $ra
```

```
# ALGORITMI DI CIFRATURA/DECIFRATURA

# Procedura che cifra/decifra una stringa con l'Algoritmo A in base al valore passato in $a0
algA:
	li $t3, 0	# Contatore del buffer della stringa

	forStringaAlgA:	# Ciclo di tutti i caratteri della stringa
		lb $t0, bufferMessage($t3)		# Carattere attuale da elaborare
		beq $t0, $zero, fineForStringaAlgA	# Controllo fine della stringa e del ciclo

		add $t0, $t0, $a0		# Applico l'algoritmo sul carattere
		sb $t0, bufferMessage($t3)	# Salvo il carattere cifrato

		addi $t3, $t3, 1		# Incremento del contatore del buffer per passare ai valori successivi
	j forStringaAlgA	# Iterazione successiva

	fineForStringaAlgA:
jr $ra			# Termine della procedura

# Procedura che cifra/decifra una stringa con l'Algoritmo B o C in base al valore passato in $a0
# $a1 = 0 -> algoritmo B, $a1 = 1 -> algoritmo C
algBC:
	li $t3, 0	# Contatore del buffer della stringa
	move $t1, $a1	# Flag per indicare se devo applicare algoritmo o no

	forStringaAlgBC:	# Ciclo di tutti i caratteri della stringa
		lb $t0, bufferMessage($t3)		# Carattere attuale da elaborare
		beq $t0, $zero, fineForStringaAlgBC	# Controllo fine della stringa e del ciclo
```

```
            beq $t1, 0, applyAlgBC    # Controllo il flag per verificare se devo applicare l'algoritmo

            doNotApplyAlgBC:
                li $t1, 0                   # Indico che al prossimo ciclo dovrà essere applicato l'algoritmo
                j goAwayAlgBC

            applyAlgBC:
                add $t0, $t0, $a0           # Applico l'algoritmo sul carattere
                sb $t0, bufferMessage($t3)      # Salvo il carattere cifrato
                li $t1, 1                   # Indico che al prossimo ciclo non dovrà essere applicato l'algoritmo

            goAwayAlgBC:
                addi $t3, $t3, 1            # Incremento del contatore del buffer per passare ai valori successivi
        j forStringaAlgBC   # Iterazione successiva

        fineForStringaAlgBC:
jr $ra              # Termine della procedura

# Procedura che cifra/decifra una stringa con l'Algoritmo D
algD:
        addi $sp, $sp, -4    # Posizionamento dello stack pointer per poter fare un push
        sw $ra, 0($sp)              # Salvataggio di $ra nello stack per poterlo ripristinare a fine procedura

        la $a0, bufferMessage
        jal dimensioneBuffer
        move $t0, $v0       # Valore dell'indice dell'ltimo elemento della stringa
```

```
        li $t1, 0       # Contatore del buffer della stringa

        forStringaAlgD:     # Ciclo di tutti i caratteri della stringa
                sub $t3, $t0, $t1
                bge $t1, $t3, fineForStringaAlgD

                lb $t2, bufferMessage($t1)      # Carattere attuale
                lb $t4, bufferMessage($t3)      # Carattere "opposto" all'attuale

                sb $t4, bufferMessage($t1)      # Scambio i due valori nello spazio di memoria
                sb $t2, bufferMessage($t3)

                addi $t1, $t1, 1            # Incremento del contatore del buffer per passare ai valori successivi
        j forStringaAlgD    # Iterazione successiva

        fineForStringaAlgD:
                lw $ra, 0($sp)              # Ripristino del vecchio $ra dallo stack
                addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver estratto un dato
jr $ra                  # Termine della procedura

algCifraturaE:
        addi $sp, $sp, -4   # Posizionamento dello stack pointer per poter fare un push
        sw $ra, 0($sp)              # Salvataggio di $ra nello stack per poterlo ripristinare a fine procedura

        addi $sp, $sp, -4   # Posizionamento dello stack pointer per poter fare un push
        sw $s0, 0($sp)              # Salvataggio del precedente $s0 nello stack per poterlo ripristinare a fine procedura
        addi $sp, $sp, -4   # Posizionamento dello stack pointer per poter fare un push
        sw $s1, 0($sp)              # Salvataggio del precedente $s1 nello stack per poterlo ripristinare a fine procedura
```

```
    addi $sp, $sp, -4    # Posizionamento dello stack pointer per poter fare un push
    sw $s2, 0($sp)       # Salvataggio del precedente $s2 nello stack per poterlo ripristinare a fine procedura
    addi $sp, $sp, -4    # Posizionamento dello stack pointer per poter fare un push
    sw $s3, 0($sp)       # Salvataggio del precedente $s3 nello stack per poterlo ripristinare a fine procedura


    li $t1, 0       # Contatore del buffer della stringa
    li $s1, 32      # Valore ASCII dello spazio
    li $s2, 45      # Valore ASCII del simbolo -


    # Copio bufferMessage to bufferMessageSupport
    forCopiaBufferToSupport:
        lb $t0, bufferMessage($t1)              # Carattere attuale da copiare
        beq $t0, $zero, fineForCopiaBuffer      # Controllo fine della stringa e del ciclo
        sb $t0, bufferMessageSupport($t1)       # Effettuo la copia del carattere
        addi $t1, $t1, 1                        # Incremento del contatore del buffer per passare ai valori successivi
    j forCopiaBufferToSupport       # Iterazione successiva


    fineForCopiaBuffer:
        move $s0, $t1       # Mi salvo la lunghezza del buffer
        li $t1, 0       # Resetto il contatore del buffer della stringa per usarlo su bufferMessageSupport
        li $t2, 0       # Contatore di scrittura per bufferMessage

    forStringaAlgCifE:  # Ciclo di tutti i caratteri della stringa
        bge $t1, $s0, fineForStringaAlgCifE     # Controllo fine della stringa e del ciclo
        lb $t0, bufferMessageSupport($t1)    # Carattere attuale da elaborare
```

```
        beq $t0, $zero, goAwayAlgE          # Se il carattere è già stato elaborato, vado al successivo

        beq $t2, 0, stampaCarattereAlgE
        sb $s1, bufferMessage($t2)      # Scrivo lo spazio nella stringa finale
        addi $t2, $t2, 1            # Incremento il contatore di scrittura

        stampaCarattereAlgE:
            beq $t0, 32, trovaSuccessiveRicorrenze
            sb $t0, bufferMessage($t2)      # Scrivo il carattere nella stringa finale
            addi $t2, $t2, 1            # Incremento il contatore di scrittura


        trovaSuccessiveRicorrenze:      # Cerco le successive ricorrenze del carattere trovato
            move $t3, $t1
            sub $t3, $t3, 1

        forSuccessiveRicorrenze:
            addi $t3, $t3, 1
            bgt $t3, $s0, fineForSuccessiveRicorrenze    # Controllo fine della stringa e del ciclo
            lb $t4, bufferMessageSupport($t3)           # Carico il carattere da controllare
            bne $t0, $t4, forSuccessiveRicorrenze               # Se il carattere non mi interessa, passo al successivo

            # Se invece il carattere mi interessa
            sb $zero, bufferMessageSupport($t3)         # Lo cancello, per non elaborarlo nuovamente in seguito
            sb $s2, bufferMessage($t2)              # Scrivo il separatore nella stringa finale
```

```
                addi $t2, $t2, 1                # Incremento il contatore di
scrittura

                li $s3, 10      # Numero per cui dividere se voglio scorrere le cifre di
un numero
                move $t6, $t3

            li $t7, 0       # Contatore scrittura su bufferCifre

                forScorroCifre:     # Scorro le cifre della posizione
                    div $t6, $s3
                    mfhi $t5        # resto della divisione per 10
                    mflo $t6        # quoziente della divisione per 10

                addi $t5, $t5, 48

                    sb $t5, bufferCifre($t7)    # Scrivo la sua posizione nella
stringa finale
                    addi $t7, $t7, 1            # Incremento il contatore di
scrittura

                    beq $t6, 0, fineForScorroCifre
                j forScorroCifre

                fineForScorroCifre:
                    li $t9, 1
                    sub $t7, $t7, $t9
                    move $t5, $t7
                    li $t6, 0       # Contatore bufferCifre

                forInvertoCifre:    # Ciclo di tutti i caratteri della stringa per
invertirli
                    sub $t7, $t5, $t6
```

```
                bge $t6, $t7, fineForInvertoCifre

                lb $t8, bufferCifre($t6)    # Carattere attuale
                lb $t9, bufferCifre($t7)    # Carattere "opposto" all'attuale
                sb $t9, bufferCifre($t6)    # Scambio i due valori
                sb $t8, bufferCifre($t7)

                addi $t6, $t6, 1            # Incremento del contatore del
buffer per passare ai valori successivi
            j forInvertoCifre     # Iterazione successiva

            fineForInvertoCifre:
                li $t6, 0                   # Contatore bufferCifre
                addi $t5, $t5, 1

            forRicopioBufferCifre:     # Copio la cifra della posizione nel
messaggio
                beq $t5, $t6, forSuccessiveRicorrenze

                lb $t7, bufferCifre($t6)    # Carattere da copiare
                sb $t7, bufferMessage($t2)      # Scrivo il carattere nella
stringa finale
                addi $t2, $t2, 1            # Incremento il contatore di
scrittura

                addi $t6, $t6, 1
            j forRicopioBufferCifre

        j forSuccessiveRicorrenze

        fineForSuccessiveRicorrenze:

        goAwayAlgE:
```

```
                addi $t1, $t1, 1            # Incremento del contatore del buffer per passare ai valori successivi

                j forStringaAlgCifE # Iterazione successiva

        fineForStringaAlgCifE:
        lw $s3, 0($sp)              # Ripristino del vecchio $s3 dallo stack
        addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver estratto un dato
        lw $s2, 0($sp)              # Ripristino del vecchio $s2 dallo stack
        addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver estratto un dato
        lw $s1, 0($sp)              # Ripristino del vecchio $s1 dallo stack
        addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver estratto un dato
        lw $s0, 0($sp)              # Ripristino del vecchio $s0 dallo stack
        addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver estratto un dato

        lw $ra, 0($sp)              # Ripristino del vecchio $ra dallo stack
        addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver estratto un dato
jr $ra              # Termine della procedura

algDecifraturaE:

        addi $sp, $sp, -4   # Posizionamento dello stack pointer per poter fare un push
        sw $ra, 0($sp)              # Salvataggio di $ra nello stack per poterlo ripristinare a fine procedura

        addi $sp, $sp, -4   # Posizionamento dello stack pointer per poter fare un push
        sw $s0, 0($sp)              # Salvataggio del precedente $s0 nello stack per poterlo ripristinare a fine procedura
        addi $sp, $sp, -4   # Posizionamento dello stack pointer per poter fare un push
```

```
    sw $s1, 0($sp)              # Salvataggio del precedente $s1 nello stack per poterlo ripristinare a fine procedura
    addi $sp, $sp, -4   # Posizionamento dello stack pointer per poter fare un push
    sw $s2, 0($sp)              # Salvataggio del precedente $s2 nello stack per poterlo ripristinare a fine procedura
    addi $sp, $sp, -4   # Posizionamento dello stack pointer per poter fare un push
    sw $s3, 0($sp)              # Salvataggio del precedente $s3 nello stack per poterlo ripristinare a fine procedura


    li $t1, 0       # Contatore del buffer della stringa
    li $s1, 32      # Valore ASCII dello spazio
    li $s2, 45      # Valore ASCII del simbolo -


    # Copio bufferMessage to bufferMessageSupport
    forCopiaBufferToSupportDec:
        lb $t0, bufferMessage($t1)              # Carattere attuale da copiare
        beq $t0, $zero, fineForCopiaBufferDec       # Controllo fine della stringa e del ciclo
        sb $t0, bufferMessageSupport($t1)   # Effettuo la copia del carattere
        sb $zero, bufferMessage($t1)            # Svuoto lo spazio su cui scriverò
        addi $t1, $t1, 1                    # Incremento del contatore del buffer per passare ai valori successivi
    j forCopiaBufferToSupportDec  # Iterazione successiva


    fineForCopiaBufferDec:
        move $s0, $t1       # Mi salvo la lunghezza del buffer
        li $t1, -1      # Resetto il contatore del buffer della stringa per usarlo su bufferMessageSupport
        li $t2, 0       # Contatore di scrittura per bufferMessage

    forStringaAlgDecE:      # Ciclo di tutti i caratteri della stringa
```

```
        addi $t1, $t1, 1
        bge $t1, $s0, fineForStringaAlgDecE   # Controllo fine della stringa e del ciclo
        lb $t0, bufferMessageSupport($t1)     # Carattere attuale da elaborare
        beq $t0, $zero, fineForStringaAlgDecE   # Controllo fine della stringa e del ciclo


        elaboraCarattere: # Il carattere da elaborare sta in $t0
            li $t3, 0 # Contatore per svuotare lo spazio di memoria delle cifre
            forSvuotoSpazioCifre:
                lb $t4, bufferCifre($t3)            # Carico il carattere da controllare
                beq $t4, $zero, trovaSuccessiveCifre  # Se ho cancellato tutte le cifre, proseguo
                sb $zero, bufferCifre($t3)              # Cancello la cifra
                addi $t3, $t3, 1
            j forSvuotoSpazioCifre


            trovaSuccessiveCifre:      # Cerco le successive cifre del carattere trovato
                addi $t1, $t1, 1
                li $t5, 0

            forSuccessiveCifre:
                addi $t1, $t1, 1
                bgt $t1, $s0, fineDellaCifra              # Controllo fine della stringa e del ciclo
                lb $t4, bufferMessageSupport($t1)           # Carico il carattere da controllare

                beq $t4, $s1, fineDellaCifra                # Se il carattere è uno spazio conclude una cifra (e tutte le cifre del carattere in corso)
```

```
                beq $t4, $s2, fineDellaCifra            # Se il carattere
è un trattino conclude una cifra
                beq $t4, $zero, fineDellaCifra          # Se il carattere
è un trattino conclude una cifra

                # Se ricevo un numero
                sb $t4, bufferCifre($t5)
                addi $t5, $t5, 1
                j forSuccessiveCifre


                fineDellaCifra: # Se trovo un trattino o uno spazio
                li $t3, 0
                li $t8, 0
                scorroLaCifra:
                        lb $t6, bufferCifre($t3)        # Carico il carattere da
controllare
                        beq $t6, $zero, scrivoIlCarattere       # Se ho letto
tutte le cifre, proseguo
                        sb $zero, bufferCifre($t3)              # Cancello la
cifra dal bufferCifre
                        addi $t3, $t3, 1

                        addi $t6, $t6, -48

                        li $t7, 1
                        li $t9, 1
                        forPotenzaDieci:
                                beq $t7, $t5, fineForPotenzaDieci
                                addi $t7, $t7, 1
                                mul $t9, $t9, 10
                        j forPotenzaDieci
```

```
                fineForPotenzaDieci:
                addi $t5, $t5, -1

                mul $t6, $t6, $t9
                add $t8, $t8, $t6

            j scorroLaCifra

            scrivoIlCarattere:
                sb $t0, bufferMessage($t8)

                beq $t4, $s1, forStringaAlgDecE          # Se il carattere è uno spazio conclude il carattere
                beq $t4, $zero, fineForStringaAlgDecE    # Se il carattere è la fine, conclude tutto
            j forSuccessiveCifre

            fineCifraTotale:
                li $t5, 0

        j forStringaAlgDecE     # Iterazione successiva

    fineForStringaAlgDecE:
    lw $s3, 0($sp)          # Ripristino del vecchio $s3 dallo stack
    addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver estratto un dato
    lw $s2, 0($sp)          # Ripristino del vecchio $s2 dallo stack
    addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver estratto un dato
    lw $s1, 0($sp)          # Ripristino del vecchio $s1 dallo stack
    addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver estratto un dato
    lw $s0, 0($sp)          # Ripristino del vecchio $s0 dallo stack
```

```
        addi $sp, $sp, 4      # Risistemazione dello stack pointer dopo aver estratto un dato

        lw $ra, 0($sp)              # Ripristino del vecchio $ra dallo stack
        addi $sp, $sp, 4      # Risistemazione dello stack pointer dopo aver estratto un dato
jr $ra                  # Termine della procedura


# Procedura che calcola l'indice dell'ultimo elemento presente in uno spazio di memoria
dimensioneBuffer:
        li $t1, -1 #counter rows
        forDimensioneBuffer:
                lb $t0, 0($a0)
                beq $t0, $zero, fineDimensioneBuffer          # Controllo fine della stringa e del ciclo
                addi $t1, $t1, 1
                addi $a0, $a0, 1
        j forDimensioneBuffer

        fineDimensioneBuffer:
        move $v0, $t1
jr $ra

# Procedura "letturaFile" che viene utilizzata per leggere un file e salvarne il contenuto nel giusto spazio di memoria
letturaFile:
        addi $sp, $sp, -4     # Posizionamento dello stack pointer per poter fare un push
        sw $s0, 0($sp)              # Salvataggio del precedente $s0 nello stack per poterlo ripristinare a fine procedura
        addi $sp, $sp, -4     # Posizionamento dello stack pointer per poter fare un push
        sw $s1, 0($sp)              # Salvataggio del precedente $s1 nello stack per poterlo ripristinare a fine procedura
```

```
    # Apertura File
    # Il nome del file viene già passato in $a0
    move $s0, $a1           # Salvataggio del registro $a1, che viene passato alla procedura, prima che venga sostituito
    li $v0, 13              # Syscall per aprire un file
    li $a1, 0               # Flag che indica l'intenzione di leggere nel file
    li $a2, 0               # (ignorato)
    syscall
    move $s1, $v0           # Salvataggio del descrittore del file
    blt $v0, 0, errorReadFile # In caso di errore di apertura nel file

    # Lettura File
    li $v0, 14              # Syscall per leggere in un file
    move $a0, $s1           # Descrittore del file da cui leggere

    beq $s0, 0, openMessageFile

    openKeyFile:
        la $a1, bufferKey   # Indirizzo del buffer in cui mettere i dati letti dal file
        li $a2, 16          # Numero di caratteri da leggere
        j openFile

    openMessageFile:
        la $a1, bufferMessage     # Indirizzo del buffer in cui mettere i dati letti dal file
        li $a2, 1024        # Numero di caratteri da leggere

    openFile:
        syscall

    move $t1, $v0       # Numero di caratteri letti nel file
```

```
        closeFile:      # Fine del file
        # Chiusura file
        li      $v0, 16             # Syscall per chiudere un file
        move $a0, $s1       # Descrittore del file da chiudere
        syscall

        j fineLettura

        # In caso di errore di apertura del file
        errorReadFile:
                move $t0, $a0       # Salvataggio del nome del file da stampare
                li $v0, 4       # Syscall per stampare
                la $a0, fnf     # Stringa di errore da stampare
                syscall
                move $a0, $t0       # Nome del file da stampare dopo la stringa di errore
                syscall

        fineLettura:
                lw $s1, 0($sp)              # Ripristino del vecchio $s1 dallo stack
                addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver
estratto un dato
                lw $s0, 0($sp)              # Ripristino del vecchio $s0 dallo stack
                addi $sp, $sp, 4    # Risistemazione dello stack pointer dopo aver
estratto un dato

jr $ra  # Termine della procedura

scritturaFile:
        # Il nome del file viene già passato in $a0
        #OPEN FILE
                li      $v0, 13             # Open File Syscall
                li      $a1, 1          # Write-only Flag
```

```
        li      $a2, 0          # (ignored)
        syscall
        move $t4, $v0       # Save File Descriptor
        blt     $v0, 0, erroreWriteFile    # Goto Error
    #WRITE FILE
        li      $v0, 15                    # Write File Syscall
        move $a0, $t4               # Load File Descriptor
        la      $a1, bufferMessage        # Load Buffer Address
        li      $a2, 200000                # Buffer Size
        syscall
    #CLOSE FILE
        li      $v0, 16                # Close File Syscall
        move $a0, $t4       # Load File Descriptor
        syscall
        j fineScrittura

    # Error
    erroreWriteFile:
        li      $v0, 4          # Print String Syscall
        la      $a0, fnf        # Load Error String
        syscall

    fineScrittura:
jr $ra # Termine della procedura
```