



UNIVERSITÀ
DEGLI STUDI
FIRENZE

Progetto Architetture Degli Elaboratori

Autori:

Marco Vignini – 5444851 – marco.vignini@stud.unifi.it

Manuel Ronca – 5094658 – manuel.ronca@stud.unifi.it

Luigi Favaro - 5251492 – luigi.favaro@stud.unifi.it

Data di consegna: 25/ 05/ 2015

Esercizio 1

Utilizzando QtSpim, scrivere e provare un programma che realizzi un identificatore di sotto-sequenze di caratteri. Il programma legge in input da file una sequenza di N caratteri, con $10 \leq N \leq 100$, dove ogni carattere può essere o '0' o '1'. Il file di input deve chiamarsi "sequenza.txt" e deve risiedere nella stessa cartella in cui è presente l'eseguibile QtSpim. Il programma permette anche di inserire in input da tastiera un numero intero X compreso nell'intervallo $-511 \leq X \leq 511$.

Sia $[C_1 \dots C_N]$ la sequenza di caratteri memorizzati su file, dove C_1 è il primo carattere memorizzato nel file stesso, e sia $\text{Seq}(k)$ una sua sotto-sequenza di 10 caratteri, ovvero $\text{Seq}(k) = [C_k \dots C_{k+9}]$ per qualche $1 \leq k \leq N-9$.

Il programma:

- Visualizza su console l'insieme degli indici k tali che $V([C_k \dots C_{k+9}]_{2,MS}) = X$, ovvero l'insieme degli indici k tali che $\text{Seq}(k)$ corrisponde alla codifica in **Modulo e Segno (MS)** su 10 bit del numero intero X inserito in input da tastiera.

- Visualizza su console l'insieme degli indici k tali che $V([C_k \dots C_{k+9}]_{2,C2}) = X$, ovvero l'insieme degli indici k tali che $\text{Seq}(k)$ corrisponde alla codifica in **Complemento a Due (C2)** su 10 bit del numero intero X inserito in input da tastiera.

- Visualizza su console l'insieme degli indici k tali che $V([C_k \dots C_{k+9}]_{2,C1}) = X$, ovvero l'insieme degli indici k tali che $\text{Seq}(k)$ corrisponde alla codifica in **Complemento a Uno (C1)** su 10 bit del numero intero X inserito in input da tastiera.

Ad esempio, sia $X = -6$ e sia $[C_1 \dots C_N] = [1000000110111111101000000110]$ (quindi $N=28$); il programma deve produrre il seguente output su console:

Caso MS - Elenco indici di inizio occorrenza: 1, 19

Caso C2 - Elenco indici di inizio occorrenza: 11

Caso C1 - Elenco indici di inizio occorrenza: nessuna occorrenza

Infatti:

- Caso MS: $V([C_1 \dots C_{10}]_{2,MS}) = V([C_{19} \dots C_{28}]_{2,MS}) = V([1000000110]_{2,MS}) = X = -6$;
- Caso C2: $V([C_{11} \dots C_{20}]_{2,C2}) = V([1111111010]_{2,C2}) = X = -6$;
- Caso C1: nessuna occorrenza della sotto-stringa $[1111111001]_{2,C1}$ (che è la codifica in C1 del numero -6).

Descrizione degli algoritmi utilizzati

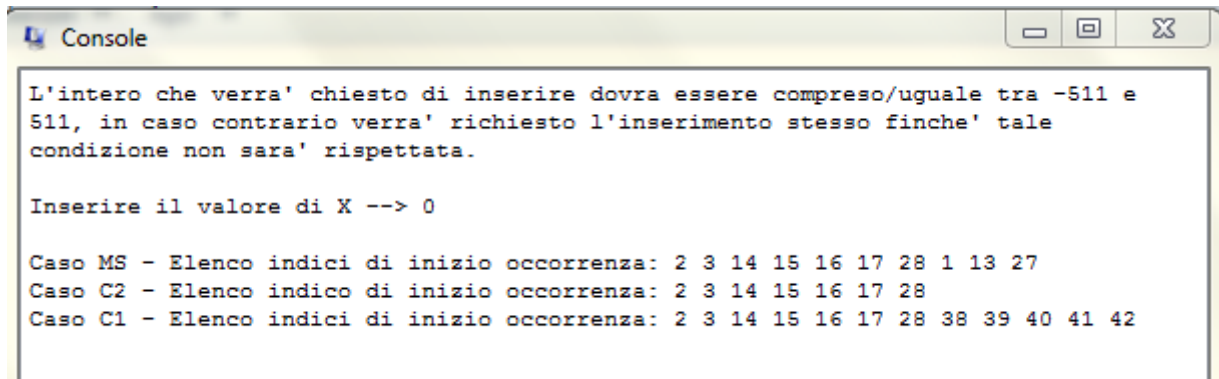
Alla x (inserita da tastiera dall'utente) vengono applicati tre algoritmi diversi per il calcolo delle varie codifiche: MS, C2 e C1. Per memorizzare di volta in volta la X codificata usiamo un buffer (chiamato `codificaX`) lungo 10 byte dato che questa è la lunghezza della sotto-sequenza da esaminare nella stringa presa da file (che invece è memorizzata nel buffer chiamato `sequenzaDaFile`). Per la codifica in modulo e segno usiamo il metodo delle divisioni ripetute inserendo di volta in volta il resto della divisione in `codificaX`, per il segno si fa un controllo all'inserimento della X (0 per il caso positivo ed 1 per quello negativo). Per il complemento a 2 il metodo applicato è lo stesso del modulo e segno con la

differenza che il bit più significativo ovviamente va a contribuire nel valore del numero. Per il caso negativo il metodo applicato è il seguente metodo: si scorre la parola da destra a sinistra complementando a partire dal primo 1 trovato. Per il complemento a 1 se il nostro numero è positivo si applica lo stesso metodo visto per il complemento a 2, se invece è negativo si complementano tutti i bit. Per quanto riguarda l'algoritmo di ricerca degli indici, usiamo due contatori all'interno di un doppio ciclo for (il primo contatore arriva alla lunghezza totale della stringa contenuta nel file meno nove in modo da riconoscere l'ultima possibile sotto-sequenza, rispettando quindi la condizione fornita dalla specifica $1 \leq k \leq N-9$) che aumenta il primo contatore solo quando è stata trovata una sotto - sequenza di lunghezza 10 nel buffer che contiene la sequenza da file corrispondente con la codifica della X, oppure i bit delle due sequenze confrontate (quella da file e quella che rappresenta la X inserita) non coincidono. Nel caso in cui X sia uguale a 0 c'è subito un controllo che fa saltare a un'altra parte del programma che si occupa di trovare le occorrenze dello zero in ogni codifica visto che per il modulo e segno e complemento a uno c'è una doppia codifica (attenzione: prima vengono stampati gli zeri positivi e poi quelli negativi). Nella procedura main usiamo vari registri preservati che prima di essere utilizzati devo essere salvati nello stack (e il return address per l'Exception Handler) per convenzione dato che il main è un chiamato. Alla fine della procedura ripristinati i registri caller - saved.

Motivazione delle scelte implementative

La scelta del metodo usato è stata determinata dalla maggiore efficienza rispetto a quello che consisteva nel prendere tutte le sottostringhe del file trasformandole in numero naturale e di confrontarle con la nostra X inserita da tastiera. Il nostro algoritmo una volta tradotta la X permette di scartare subito una determinata sottostringa appena non c'è corrispondenza di bit. Nel caso della codifica dello zero il nostro algoritmo raggiunge un picco di efficienza perché cerca solo specifiche codifiche su 10 bit. Usiamo soltanto un buffer di lunghezza 10 per le sottosequenze (invece che un buffer per ogni codifica) perché lo sovrascriviamo di volta in volta, dopo aver controllato le occorrenze della X nel caso della codifica precedente. Ciò ci permette di risparmiare sulla memoria usata (vengono usati 10 byte invece che 30). **Peculiarità:** Le varie codifiche della X nel buffer codificaX sono scritte al contrario.

Simulazioni



```
Console

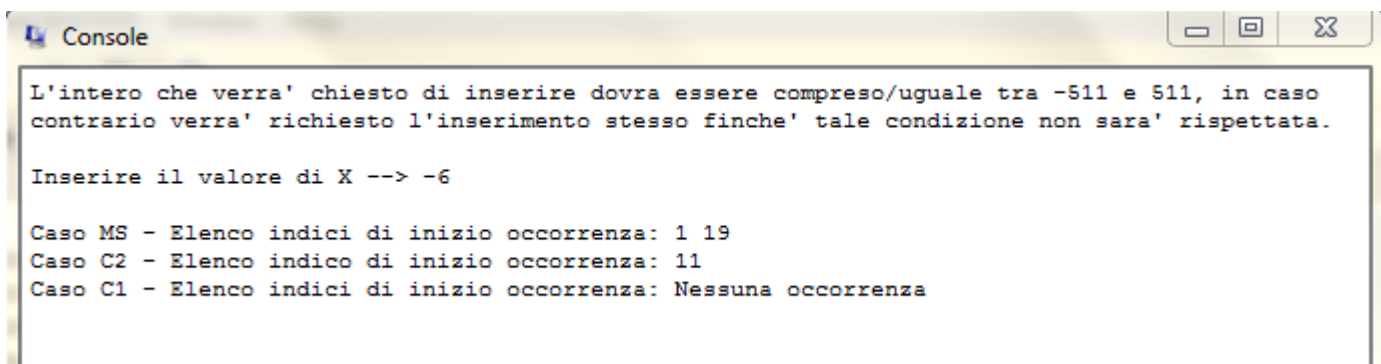
L'intero che verra' chiesto di inserire dovra essere compreso/uguale tra -511 e 511, in caso contrario verra' richiesto l'inserimento stesso finche' tale condizione non sara' rispettata.

Inserire il valore di X --> 0

Caso MS - Elenco indici di inizio occorrenza: 2 3 14 15 16 17 28 1 13 27
Caso C2 - Elenco indico di inizio occorrenza: 2 3 14 15 16 17 28
Caso C1 - Elenco indici di inizio occorrenza: 2 3 14 15 16 17 28 38 39 40 41 42
```

Con sequenza:

100000000000100000000000001000000000011111111111111



```
Console

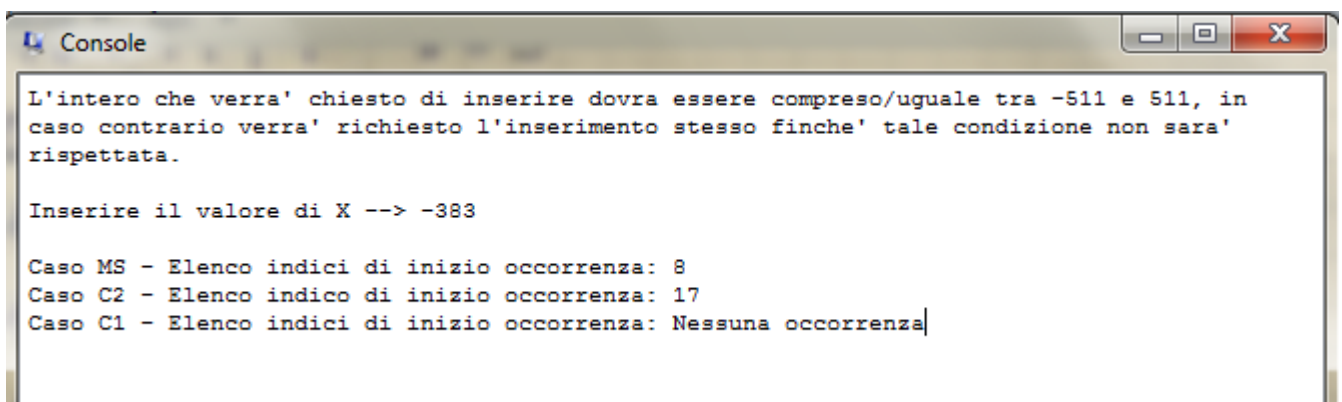
L'intero che verra' chiesto di inserire dovra essere compreso/uguale tra -511 e 511, in caso contrario verra' richiesto l'inserimento stesso finche' tale condizione non sara' rispettata.

Inserire il valore di X --> -6

Caso MS - Elenco indici di inizio occorrenza: 1 19
Caso C2 - Elenco indico di inizio occorrenza: 11
Caso C1 - Elenco indici di inizio occorrenza: Nessuna occorrenza
```

Con sequenza:

10000001101111111101000000110



```
Console

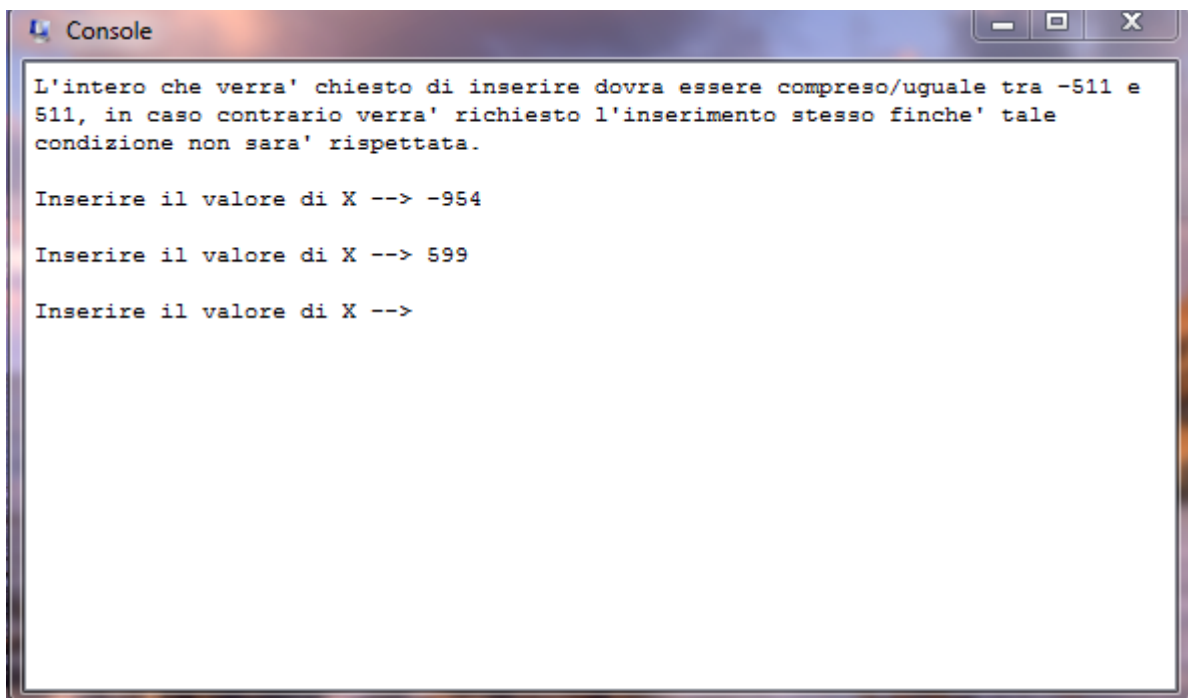
L'intero che verra' chiesto di inserire dovra essere compreso/uguale tra -511 e 511, in caso contrario verra' richiesto l'inserimento stesso finche' tale condizione non sara' rispettata.

Inserire il valore di X --> -383

Caso MS - Elenco indici di inizio occorrenza: 8
Caso C2 - Elenco indico di inizio occorrenza: 17
Caso C1 - Elenco indici di inizio occorrenza: Nessuna occorrenza
```

Con sequenza:

10000001101111111101000000110



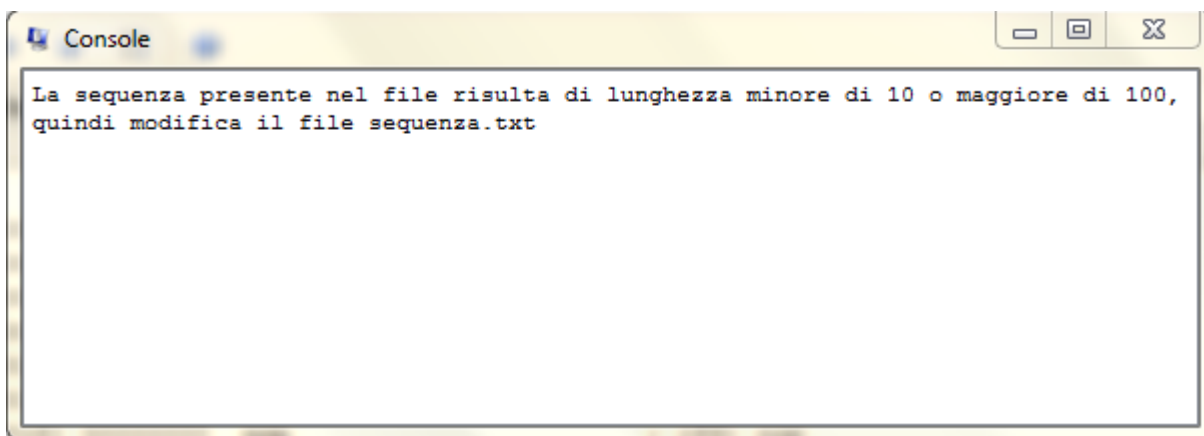
```
Console
L'intero che verra' chiesto di inserire dovra essere compreso/uguale tra -511 e
511, in caso contrario verra' richiesto l'inserimento stesso finche' tale
condizione non sara' rispettata.

Inserire il valore di X --> -954

Inserire il valore di X --> 599

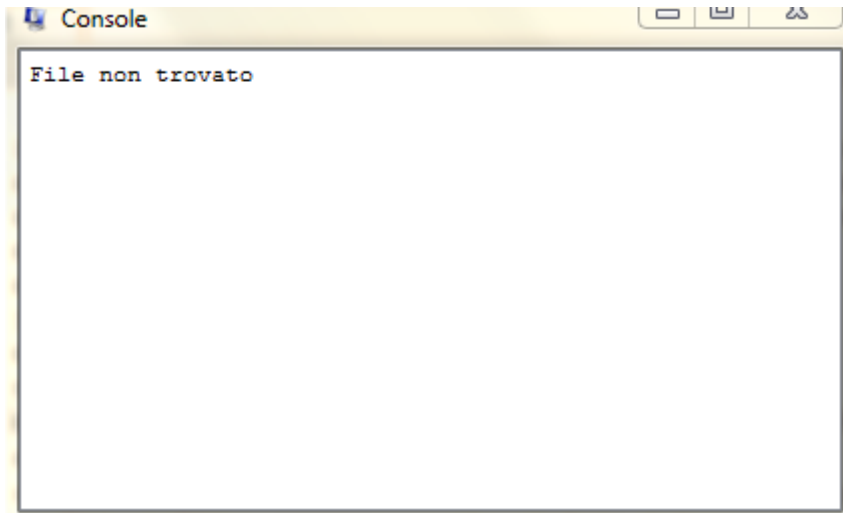
Inserire il valore di X -->
```

Simulazione con input sbagliati, fuori da range $-511 \leq X \leq 511$



```
Console
La sequenza presente nel file risulta di lunghezza minore di 10 o maggiore di 100,
quindi modifica il file sequenza.txt
```

Simulazione con una sequenza da file che non rispetta i vincoli di lunghezza specificati nel testo dell'esercizio.



Simulazione in caso sequenza.txt non sia presente nella cartella dove è installato QtSpim.

Evoluzione dell'User Data Segment:

```
User data segment [10000000]..[10040000]
[10000000]..[10010007] 00000000
[10010008] 0808517632 0808464432 . . 1 0 0 0 0 0
[10010010] 0808530224 0825307441 0808530225 0808464433 0 1 1 0 1 1 1 1 1 1 0 1 0 0 0
[10010020] 0825241648 0000012337 0000000000 0000000000 0 0 0 1 1 0 . . . . .
[10010030]..[1001006b] 00000000
```

Stato dell'user data segment dopo la lettura della sequenza da file.

```
User data segment [10000000]..[10040000]
[10000000]..[1000ffff] 00000000
[10010000] 0808530224 0808464432 0808530224 0808464432 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0
[10010010] 0808530224 0825307441 0808530225 0808464433 0 1 1 0 1 1 1 1 1 1 1 0 1 0 0 0
[10010020] 0825241648 0000012337 0000000000 0000000000 0 0 0 1 1 0 . . . . .
[10010030]..[1001006b] 00000000
```

Evoluzione dell'User Data Segment dopo la codifica in MS(es: X = -6 evidenziato in giallo)

```
User data segment [10000000]..[10040000]
[10000000]..[1000ffff] 00000000
[10010000] 0825241904 0825307441 0808530225 0808464432 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0
[10010010] 0808530224 0825307441 0808530225 0808464433 0 1 1 0 1 1 1 1 1 1 1 0 1 0 0 0
[10010020] 0825241648 0000012337 0000000000 0000000000 0 0 0 1 1 0 . . . . .
[10010030]..[1001006b] 00000000
```

Evoluzione dell'User Data Segment dopo la codifica in C2 (es: X = -6 evidenziato in giallo)

```
User data segment [10000000]..[10040000]
[10000000]..[1000ffff] 00000000
[10010000] 0825241649 0825307441 0808530225 0808464432 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0
[10010010] 0808530224 0825307441 0808530225 0808464433 0 1 1 0 1 1 1 1 1 1 1 0 1 0 0 0
[10010020] 0825241648 0000012337 0000000000 0000000000 0 0 0 1 1 0 . . . . .
[10010030]..[1001006b] 00000000
```

Evoluzione dell'User Data Segment dopo la codifica in C1 (es: X = -6 evidenziato in giallo)

Codice:

Esercizio 1: Identificatore di sotto-sequenze.

.data

codificaX: .space 10

Array di codifica per la

X. Gli space vanno messi in alto per non farli sfalzare a non multipli di 4.

sequenzaDaFile: .space 100

Posso

leggere fino a 100 caratteri da file.

welcome: .asciiiz "L'intero che verra' chiesto di inserire dovra essere compreso/uguale tra -511 e 511, in caso contrario verra' richiesto l'inserimento stesso finche' tale condizione non sara' rispettata.\n"

input: .asciiiz "\nInserire il valore di X --> "

file: .asciiiz "sequenza.txt"

fnt: .asciiiz "File non trovato"

indiciMS: .asciiiz "\nCaso MS - Elenco indici di inizio occorrenza: "

indiciC1: .asciiiz "\nCaso C1 - Elenco indici di inizio occorrenza: "

indiciC2: .asciiiz "\nCaso C2 - Elenco indico di inizio occorrenza: "

noOccorrenze: .asciiiz "Nessuna occorrenza"

spazio: .asciiiz " "

erroreFile2: .asciiiz "La sequenza presente nel file risulta di lunghezza minore di 10 o maggiore di 100, quindi modifica il file sequenza.txt"

.text

.globl main

main:

addi \$sp, \$sp, -32

Il main è un chiamato (QtSpim fa jal main)

e quindi alloco nello stack i registri preservati che voglio usare e il return address.

sw \$s0, 0(\$sp)

sw \$s1, 4(\$sp)

sw \$s2, 8(\$sp)

sw \$s3, 12(\$sp)

sw \$s4, 16(\$sp)

sw \$s5, 20(\$sp)

sw \$s6, 24(\$sp)

sw \$ra, 28(\$sp)

li \$s5, 1

li \$s6, 0 delle occorrenze, servira' in fondo al programma.	# Contatore per la stampa
li \$v0, 13 system call).	# Apertura del file(
la \$a0, file questione.	# Carico il nome del file in
li \$a1, 0 scrittura file.	# 0 = lettura file. 1 =
li \$a2, 0 in questo caso = 0).	# Lunghezza file (ignorata
syscall	
move \$t6, \$v0 verificarne la correttezza.	# Salvo il file per
blt \$v0, 0, errore	
li \$v0, 14	# Leggo il file.
move \$a0, \$t6	
la \$a1, sequenzaDaFile	# Indirizzo della sequenza da file.
li \$a2, 100 lunghezza che mi interessa. La sequenza da file può essere lunga massimo 100 caratteri.	# Ora specifico la
syscall	
controlloLunghezza:	
la \$s1, sequenzaDaFile	# Carico l'indirizzo della sequenza da file.
move \$s3, \$s1 sequenza da File.	# Mi servira' una copia dell'indirizzo della
li \$s4, 0 sequenzaDaFile.	# Contatore della lunghezza del
nextCh: lb \$t3,(\$s1)	# Leggo un carattere della stringa (pseudoistruzione).
beqz \$t3, controlloErroreLunghezza	# Se è zero ho finito (pseudoistruzione).
add \$s4, \$s4, 1	# Incremento il contatore (lunghezza della stringa).
add \$s1, \$s1, 1	# incremento la posizione sulla stringa(indirizzo).
j nextCh	

controlloErroreLunghezza:
essere compresa/uguale tra 10 e 100.

La lunghezza della stringa da file deve

li \$t1, 10

bge \$s4, \$t1, controlloLunghezza2

j errore2

controlloLunghezza2:

blt \$s4, 100, chiusuraFile
chiudo il file.

Dopo aver fatto anche il secondo controllo

j errore2

chiusuraFile:

li \$v0, 16

Chiusura del file.

move \$a0, \$t6

syscall

j inizio

errore:

li \$v0, 4
stringa che se ne occupa).

File non trovato (stampa la

la \$a0, fnt

syscall

j exit

errore2:

li \$v0, 4

la \$a0, erroreFile2
riguardante la lunghezza della stringa contenuta nel file.

Stringa che avverte che il tipo di errore e' quello

syscall

li \$v0, 16

Chiusura del file.

move \$a0, \$t6

syscall

j exit

inizio:

li \$v0, 4

Istruzioni per l'uso.

la \$a0, welcome

syscall

inserimento:

li \$v0, 4

Richiesta dell'inserimento dell'input.

la \$a0, input

syscall

li \$v0, 5

Lettura di X.

syscall

slti \$t0, \$v0, -511 # Se \$v0 < -511 metto \$t0 = 1. Controllo sul valore
inserito(deve essere compreso tra -511 e 511). Principio di progetto: uso direttamente le costanti per incrementare la velocità
d'esecuzione.

li \$t1, 1

Mi serve il controllo con la branch if equal.

beq \$t0, \$t1, inserimento
= 1 --> \$v0 < -511.

Input sbagliato, faccio reinserire all'utente l'intero. Perché se \$t0

li \$t2, 511

Riuso \$t1 per risparmiare registri.

slt \$t0, \$t2, \$v0

Controllo che l'intero sia <= di 511.

beq \$t0, \$t1, inserimento
= 1 --> \$v0 > 511.

Input sbagliato, faccio reinserire all'utente l'intero. Perché se \$t0

move \$t0, \$a1
scorrerlo.

Mi serve l'indirizzo del sequenzaDaFile per

```

move $t1, $v0                                # X.

la $t2, codificaX                            # Indirizzo dell'array codificaX.

move $s2, $t2

Start:

addi $s4, $s4, -9                            # Lunghezza = Lunghezza - 9, dato che l'ultima sotto-sequenza di lunghezza 10 puo' cominciare da
LunghezzaSequenza - 9.

li $t3, 2                                    # Si usa nella divisione per 2.

li $t7, 9                                    # Fine dell'array.

move $s0, $t1                                # Lo salvo perche' nel corso del programma
mi riservira'.

bne $t1, $zero, NonZero

j inizAlgoritmoZeri                          # 0 --> Ha due codifiche in C1 ed in MS quindi va
trattato a parte.

NonZero:                                     # Tutti gli altri casi.

codificaMS:                                 # Metodo delle divisioni ripetute.

div $t1, $t3                                # Divido X per 2.

mflo $t1                                     # Metto il quoziente in $t1 per accumularlo.

mfhi $t4                                     # Metto il resto in $t4.

bne $t4, $zero, codifica1

li $t8, 48                                  # Codifiche ascii dello 0 e dell'1.

sb $t8, ($t2)                               # Scrittura nell'array codificaX.

addi $t9, $t9, 1                            # Contatore per arrivare alla fine della stringa.

addi $t2, $t2, 1                            # Incremento dell'indirizzo.

beq $t9, $t7, codificaSegno                # Fine delle divisioni ripetute perche' il quoziente e' zero.

j codificaMS

codifica1:

li $t8, 49

```

```

sb $t8, ($t2)                # Scrittura nell'array codificaX.

addi $t9,$t9, 1              # Contatore per arrivare alla fine della stringa.

addi $t2, $t2, 1             # Incremento dell'indirizzo.

beq $t9, $t7, codificaSegno  # Fine delle divisioni ripetute perche' il quoziente e' zero.

j codificaMS

codificaSegno:

blt $s0, $zero, segnoArrayRestiNeg    # Segno del numero.

li $t8, 48                             # Per il caso positivo.

sb $t8, ($t2)                          # Segno positivo.


li $v0, 4                             # Caso MS - Elenco indici di inizio occorrenza.
la $a0, indiciMS
syscall

j reinizializzazionePerRicercaIndici    # Vado a ricercare gli indici.

segnoArrayRestiNeg:

li $t8, 49                             # Corrispondente di 1 in ascii.

sb $t8, ($t2)                          # Segno negativo.


li $v0, 4                             # Caso MS - Elenco indici di inizio occorrenza.
la $a0, indiciMS
syscall

j reinizializzazionePerRicercaIndici    # Vado a ricercare gli indici.

reinizializzazioneC2:                # Per fare il complemento a due

addi $s6, $s6, 1                     # $s6 = 1 --> Sto facendo il C2.

li $v0, 4                             # Caso C2 - Elenco indici di inizio occorrenza.

```

la \$a0, indiciC2

syscall

move \$t1, \$s0 # X.

li \$t3, 2

move \$t2, \$s2

li \$t9, 1 # Contatore per arrivare a fine della stringa.

li \$t0, 0 # Contatore che mi dice se trovo ho trovato l'1 oppure no.

li \$t7, 10

blt \$t1, \$zero, C2negativo # Controllo sul segno.

codificaC2:

div \$t1, \$t3 # Divido X per 2.

mflo \$t1 # Metto il quoziente in \$t1 per accumularlo.

mfhi \$t4 # Metto il resto in \$t4.

bne \$t4, \$zero, codifica2C2

li \$t8, 48 # Codifiche ascii dello 0 e dell'1.

sb \$t8, (\$t2) # Scrittura nell'array di codificaX(codifica).

addi \$t9, \$t9, 1 # Contatore per arrivare alla fine della stringa.

addi \$t2, \$t2, 1 # Decremento dell'indirizzo, devo scorrere la parola al contrario(metodo 3 per il C2).

beq \$t9, \$t7, reinizializzazionePerRicercaIndici zero. # Fine delle divisioni ripetute perche' il quoziente e'

j codificaC2

codifica2C2: # Scrive 1 nel caso positivo.

li \$t8, 49

sb \$t8, (\$t2) # Scrittura nell'array di codificaX(codifica).

addi \$t9, \$t9, 1 # Contatore per arrivare alla fine della stringa.

addi \$t2, \$t2, 1 # Decremento dell'indirizzo.

beq \$t9, \$t7, reinizializzazionePerRicercaIndici # Fine delle divisioni ripetute perche' il quoziente e' zero.

j codificaC2

C2negativo:

Caso Negativo del C2

div \$t1, \$t3

Divido X per 2.

mflo \$t1

Metto il quoziente in \$t1 per accumularlo.

mfhi \$t4

Metto il resto in \$t4.

beq \$t4, \$zero, controllo0

j controllo1

scriviUno:

li \$t8, 49

Codifiche ascii dello 0 e dell'1.

sb \$t8, (\$t2)

Scrittura nell'array di codificaX(codifica).

addi \$t0, \$t0, 1

Segno di aver trovato almeno un 1.

addi \$t9, \$t9, 1

Contatore per arrivare alla fine della stringa.

addi \$t2, \$t2, 1

Incremento dell'indirizzo.

beq \$t9, \$t7, reinizializzazionePerRicercaIndici

Fine delle divisioni ripetute perche' il quoziente e' zero.

j C2negativo

scriviZero:

li \$t8, 48

sb \$t8, (\$t2)

Scrittura nell'array di codificaX(codifica).

addi \$t9, \$t9, 1

Contatore per arrivare alla fine della stringa.

addi \$t2, \$t2, 1

Decremento dell'indirizzo.

beq \$t9, \$t7, reinizializzazionePerRicercaIndici
zero.

Fine delle divisioni ripetute perche' il quoziente e'

j C2negativo

controllo0:

beq \$t0, \$zero, scriviZero

\$t0 = 0 --> Non ho ancora trovato un 1 quindi scrivo i bit normali.

j scriviUno

Complemento i bit.

controllo1:

beq \$t0, \$zero, scriviUno # \$t0 = 0 --> Non ho ancora trovato degli 1 quindi scrivo i bit normali.

j scriviZero # Complemento i bit.

reinizializzazioneC1:

addi \$s6, \$s6, 1 # \$s6 = 2 --> Sto facendo il C1.

li \$v0, 4 # Caso C1 - Elenco indici di inizio occorrenza.

la \$a0, indiciC1

syscall

li \$t7, 10

move \$t1, \$s0

li \$t3, 2

move \$t2, \$s2

li \$t9, 1

blt \$t1, \$zero, C1negativo

codificaC1:

div \$t1, \$t3 # Divido X per 2.

mflo \$t1 # Metto il quoziente in \$t1 per
accumularlo.

mfhi \$t4 # Metto il resto in \$t4.

bne \$t4, \$zero, codifica1C1

li \$t8, 48 # Codifiche ascii dello 0 e dell'1.

sb \$t8, (\$t2) # Scrittura nell'array di
codificaX(codifica).

addi \$t9, \$t9, 1 # Contatore per arrivare alla fine
della stringa.

addi \$t2, \$t2, 1 # Incremento dell'indirizzo.

beq \$t9, \$t7, reinizializzazionePerRicercaIndici # Fine delle divisioni ripetute perche' il quoziente e'
zero.

j codificaC1

codifica1C1:

li \$t8, 49

sb \$t8, (\$t2) # Scrittura nell'array di
codificaX(codifica).

addi \$t9, \$t9, 1 # Contatore per arrivare alla fine
della stringa.

addi \$t2, \$t2, 1 # Incremento dell'indirizzo.

beq \$t9, \$t7, reinizializzazionePerRicercaIndici # Fine delle divisioni ripetute perche' il quoziente e' zero.

j codificaC1

C1negativo:

Metodo delle divisioni ripetute.

div \$t1, \$t3 # Divido X per 2.

mflo \$t1 # Metto il quoziente in \$t1 per
accumularlo.

mfhi \$t4 # Metto il resto in \$t4.

bne \$t4, \$zero, codifica0

li \$t8, 49 # Codifiche ascii dello 0 e dell'1.

sb \$t8, (\$t2) # Scrittura nell'array di
codificaX(codifica).

addi \$t9, \$t9, 1 # Contatore per arrivare alla fine della stringa.

addi \$t2, \$t2, 1 # Incremento dell'indirizzo.

beq \$t9, \$t7, reinizializzazionePerRicercaIndici # Fine delle divisioni ripetute perche' il quoziente e' zero.

j C1negativo

codifica0:

li \$t8, 48

sb \$t8, (\$t2) # Scrittura nell'array di
codificaX(codifica).

addi \$t9, \$t9, 1 # Contatore per arrivare alla fine della
stringa.

addi \$t2, \$t2, 1 # Incremento dell'indirizzo.

beq \$t9, \$t7, reinizializzazionePerRicercaIndici # Fine delle divisioni ripetute perche' il quoziente e' zero.

j C1negativo

reinizializzazionePerRicercaIndici:

```
move $s1, $s3                # Riporto sequenzaDaFile al suo indirizzo originale.
li $t3, 1                    # Indice di partenza sequenzaDaFile.
li $t4, 10                   # Fine sotto-sequenza.
li $t7, 0                    # Contatore che serve ad indicare
se ci sono state occorrenze.
```

w1:

```
bgt $t3, $s4, nessunaOccorrenza    # Fine scorrimento SequenzaDaFile.
li $t1, 0                          # $t1 = contatoregiusti.
```

w2:

```
beq $t1, $t4, stampaOccorrenze
lb $t5, ($t2)                  # Prende l'ultimo carattere poichè la codifica è memorizzata al contrario.
lb $t6, ($s1)                  # Primo carattere della sequenza da file.
bne $t5, $t6, AumentaIndiceOcc # Se non sono uguali vado ad aumentare l'indice.
addi $t1, $t1, 1               # Aumento del contatoreGiusti.
addi $t2, $t2, -1              # Decremento l'indice della sotto-sequenza.
addi $s1, $s1, 1               # Punto al carattere successivo della sequenza da file
```

j w2

AumentaIndiceOcc:

```
sub $s1, $s1, $t1              # Aggiornamento indirizzo del
sequenzaDaFile.
addi $s1, $s1, 1
addi $t3, $t3, 1               # Incremento l'indice della sequenza da file.
move $t2, $s2
addi $t2, $t2, 9
```

j w1

stampaOccorrenze:

addi \$t7, \$t7, 1 # N° occorrenze, per capire se stampare la stringa che indica che non ci sono occorrenze.

move \$a0, \$t3 # Stampo l' occorrenza corrente.

li \$v0, 1

syscall

li \$v0, 4 # Spazio.

la \$a0, spazio

syscall

addi \$t3, \$t3, 1 # reinizializzazionePerRicercaIndici e indirizzi.

addi \$t2, \$s2, 9

sub \$s1, \$s1, \$t1

addi \$s1, \$s1, 1 # Scorro la sequenza da file.

j w1 # Continua a cercare
occorrenze.

nessunaOccorrenza:

beq \$t7, \$zero, controlloStampaOcc # Questo controllo va fatto altrimenti in caso di C2 senza occorrenza non
stampa la mancanza della occorrenza appunto.

beq \$s6, \$zero, reinizializzazioneC2 # Da MS a C2.

beq \$s6, \$s5, reinizializzazioneC1 # Da C2 a C1.

bgt \$s6, \$s5, exit # Ho gia' fatto C1.

controlloStampaOcc:

li \$v0, 4 # Nessuna occorrenza

la \$a0, noOccorrenze

syscall

\$s6 = 0 --> MS

\$s6 = 1 --> C2

\$s6 = 2 --> C1

beq \$s6, \$zero, reinizializzazioneC2

beq \$s6, \$s5, reinizializzazioneC1

Per C1 non ho bisogno di indicare il salto perche' devo finire.

exit:

lw \$s0, 0(\$sp) # Il chiamato ripristina i registri caller - saved che ha utilizzato e torna al chiamante(QtSpim).

lw \$s1, 4(\$sp)

lw \$s2, 8(\$sp)

lw \$s3, 12(\$sp)

lw \$s4, 16(\$sp)

lw \$s5, 20(\$sp)

lw \$s6, 24(\$sp)

lw \$ra, 28(\$sp)

addi \$sp, \$sp, 32

jr \$ra

inizAlgoritmoZeri:

li \$s5, 0

li \$s6, 0

li \$t1, 0 # Contatore per nessuna occorrenza.

algoritmoZeri:

li \$s6, 2

li \$t2, 1 # Contatore fisso.

li \$t3, 0 # Contatore dei Giusti.

li \$t6, 10

```

move $s1, $s3                                # Riporto l'indirizzo della sequenzaDaFile all'inizio.
li $t7, 48                                    # Zero in Ascii.

beq $s5, $s6, stampaC2occorrenze
bgt $s5, $t2, stampaC1occorrenze

li $v0, 4                                     # Indici Occorrenze MS.
la $a0, indiciMS
syscall

j controllo

stampaC2occorrenze:

li $v0, 4
la $a0, indiciC2                             # Indici Occorrenze C2.
syscall

j controllo

stampaC1occorrenze:

li $v0, 4
la $a0, indiciC1                             # Indici Occorrenze C1.
syscall

controllo:

bgt $t2, $s4, controlloAlg

loopZeroMS:

beq $t3, $t6, stampaIndiceCercato
lb $t5, ($s1)                                # Prendo il byte successivo dalla sequenza da File.
bne $t5, $t7, riazzera

```

```

addi $t3, $t3, 1          # Aumento il contatore dei giusti.

addi $s1, $s1, 1          # Incremento l'indirizzo della sequenzaDaFile.


j loopZeroMS


riazzera:


addi $t2, $t2, 1          # Incremento l'indice fisso.

sub $s1, $s1, $t3         # Riporto indietro l'indirizzo.

li $t3, 0                 # Reinizializzo il contatore dei giusti.

addi $s1, $s1, 1          # Incremento l'indirizzo della SequenzaDaFile.


j controllo


stampaIndiceCercato:


addi $t1, $t1, 1          # Controllo se ho stampato occorrenze oppure no, per restituire su console "Nessuna
Occorrenza" se necessario.


move $a0, $t2


li $v0, 1                 # Stampa occorrenza.

syscall


li $v0, 4                 # Spazio.

la $a0, spazio

syscall


addi $s1, $s1, -9          # Ricomincio a scansionare dall'indice fisso successivo a quello stampato per cercare una
nuova sottosequenza.

li $t3, 0                 # Riazzero il contatore dei giusti.

addi $t2, $t2, 1          # Incremento l'indice fisso.


j controllo


controlloAlg:

```

li \$t4, 3

li \$t2, 1

li \$t3, 2

beq \$s5, \$zero, reinizializzazioneLoopZeroNegativoMS # Da MS a MSNegativo.

beq \$s5, \$t2, reinizializzazioneZeroC2 # Da MSNegativo a C2.

beq \$s5, \$t3, reinizializzazioneLoopZeroPosC1 # Da C2 a C1positivo.

beq \$s5, \$t4, reinizializzazioneLoopZeroNegC1 # Da C1positivo a C1negativo.

reinizializzazioneLoopZeroNegativoMS:

addi \$s5, \$s5, 1

li \$t2, 1 # Contatore fisso.

li \$t3, 0 # Contatore dei Giusti.

move \$s1, \$s3 # Riporto l'indirizzo della sequenzaDaFile all'inizio.

li \$t7, 48 # Zero in Ascii.

li \$t8, 49 # Uno in Ascii.

li \$t9, 9

primoControllo: # Perche' se la sequenza non comincia con un 1, non potra' mai essere uno
zero negativo nel MS.

lb \$t5, (\$s1) # Prelevo il carattere da file.

beq \$t5, \$t8, loopZeroNegativoMS

addi \$s1, \$s1, 1 # Esito di ricerca dell'1 negativo --> Incremento l'indirizzo e il contatore fisso e riazzero il contatore dei
giusti.

addi \$t2, \$t2, 1

li \$t3, 0

j primoControllo

loopZeroNegativoMS:

bgt \$t2, \$s4, controlloAlg # Controllo che serve per quando sono arrivato in fondo a k - 9 della sequenza da file.

beq \$t3, \$t9, stampaOccorrenzeZeroNegativoMS

```

addi $s1, $s1, 1                # Incremento dell'indirizzo.

lb $t5, ($s1)                   # Leggo dalla sequenza da file.

bne $t5, $t7, riazzerazero2

addi $t3, $t3, 1                # Incremento il contatore dei giusti.

j loopZeroNegativoMS

riazzerazero2:

addi $t2, $t2, 1                # Incremento l'indice fisso.

sub $s1, $s1, $t3               # Riporto indietro l'indirizzo.

li $t3, 0                       # Reinizializzo il contatore dei giusti.

j primoControllo

stampaOccorrenzeZeroNegativoMS:

addi $t1, $t1, 1                # Controllo se ho stampato occorrenze oppure no, per restituire su console "Nessuna
Occorrenza" se necessario.

move $a0, $t2

li $v0, 1                       # Stampa occorrenza.

syscall

li $v0, 4

la $a0, spazio

syscall

addi $s1, $s1, -8               # Ricomincio a scansionare dall'indice fisso successivo a quello stampato.

li $t3, 0                       # Riazzero il contatore dei giusti.

addi $t2, $t2, 1                # Incremento l'indice fisso.

j primoControllo

```

reinizializzazioneZeroC2:

addi \$s5, \$s5, 1 # ---> Sto facendo C2

bne \$t1, \$zero, dopo1

li \$v0, 4 # Nel caso in cui non siano state trovate occorrenze nel caso precedente.

la \$a0, noOccorrenze

syscall

dopo1:

li \$t1, 0 # Riazzero il contatore che mi serve ad indicare se sono state stampare occorrenze
oppure no.

j algoritmoZeri

reinizializzazioneLoopZeroPosC1:

addi \$s5, \$s5, 1 # \$s5 = 3 --> Sto facendo il C1.

bne \$t1, \$zero, dopo2

li \$v0, 4 # Nel caso in cui non siano state trovate occorrenze nel caso precedente.

la \$a0, noOccorrenze

syscall

dopo2:

li \$t1, 0 # Riazzero il contatore che mi serve ad indicare se sono state stampare occorrenze
oppure no.

j algoritmoZeri

reinizializzazioneLoopZeroNegC1:


```
move $a0, $t2
```

```
li $v0, 1                                # Stampa occorrenza.
```

```
syscall
```

```
li $v0, 4                                # Spazio.
```

```
la $a0, spazio
```

```
syscall
```

```
addi $s1, $s1, -9                        # Ricomincio a scansionare dall'indice fisso successivo a quello stampato.
```

```
li $t3, 0                                # Reinizializzo il contatore dei giusti.
```

```
addi $t2, $t2, 1                          # Incremento l'indice fisso.
```

```
j controlloC1Neg
```

```
controlloFinale:                          # Per capire se devo stampare "Nessuna occorrenza" prima di uscire dal programma  
oppure no.
```

```
bne $t1, $zero, exit
```

```
li $v0, 4
```

```
la $a0, noOccorrenze
```

```
syscall
```

```
j exit
```

Esercizio 2

Siano G e F due procedure definite come segue (in pseudo-linguaggio di alto livello):

```
Procedure  $G(n)$   
begin  
     $b := 0$   
    for  $k := 0, 1, 2, \dots, n$  do  
        begin  
             $u := F(k)$   
             $b := b^2 + u$   
        end
```

```

        return b
    end

    Procedure F(n)
    begin
        if n = 0
            then return 1
            else return 2*F(n - 1) + n
        end
    end

```

Utilizzando QtSpim, scrivere e provare un programma che legga inizialmente un numero naturale n , e che visualizzi su console:

- il valore restituito dalla procedura $G(n)$, implementando G e F come descritto precedentemente. Le chiamate alle due procedure G ed F devono essere realizzate utilizzando l'istruzione jal (jump and link).
- la traccia con la sequenza delle chiamate annidate (con argomento fra parentesi) ed i valori restituiti dalle varie chiamate annidate (valore restituito fra parentesi), sia per G che per F .
Mostrare e discutere nella relazione l'evoluzione dello stack nel caso specifico in cui $n=2$.

Esempio di output su console nel caso in cui $n=1$:

Risultato: $G(1)=4$

Traccia:

$G(1) \rightarrow F(0) \rightarrow F\text{-return}(1) \rightarrow F(1) \rightarrow F(0) \rightarrow F\text{-return}(1) \rightarrow F\text{-return}(3) \rightarrow G\text{-return}(4)$

Infatti: $G(1)$ richiama $F(0)$, che ritorna il valore 1; quindi viene richiamata $F(1)$, che a sua volta richiama $F(0)$; $F(0)$ ritorna il valore 1, quindi $F(1)$ ritorna il valore 3, ed infine $G(1)$ ritorna il valore 4 che è il risultato finale.

Algoritmi utilizzati e scelte implementative:

Nello svolgimento dell'esercizio è stato seguito fedelmente lo pseudocodice indicato nella specifica dell'esercizio stesso. Si è deciso di fare prima la stampa della traccia e poi del risultato delle procedure, tale scelta è dovuta all'efficienza del metodo. Nel nostro algoritmo piazzando accuratamente le stampe otteniamo la traccia senza dover scomodare strutture dati aggiuntive e senza generare un codice più grande di quello attuale. Se la stampa della traccia fosse stata fatta dopo quella del risultato avremmo dovuto utilizzare memoria dinamica in più per memorizzare le informazioni che ci sarebbero servite per la traccia stessa (come ad esempio liste o stack dato che non conosciamo a priori la lunghezza della traccia) e l'algoritmo sarebbe risultato molto più lento.

Simulazioni

```
Console
Inserire il valore del numero naturale n --> 1
Traccia: G(1)->F(0)->F-return(1)->F(1)->F(0)->F-return(1)->F-return(3)->G-return(4)
Risultato: G(1)=4
```

Caso $n = 1$

```
Console
Inserire il valore del numero naturale n --> 2
Traccia: G(2)->F(0)->F-return(1)->F(1)->F(0)->F-return(1)->F-return(3)->F(2)->F(1)->F(0)->F-return(1)->F-return(3)->F-return(8)->G-return(24)
Risultato: G(2)=24
```

Caso $n = 2$

```
Console

Inserire il valore del numero naturale n --> 3

Traccia: G(3)->F(0)->F-return(1)->F(1)->F(0)->F-return(1)->F-return(3)->F(2)->F(1)->F(0)->F-return(1)->F-return(3)->F-return(8)->F(3)->F(2)->F(1)->F(0)->F-return(1)->F-return(3)->F-return(8)->F-return(19)->G-return(595)

Risultato: G(3)=595
```

Caso $n = 3$

```
Console

Inserire il valore del numero naturale n --> -5
Inserire il valore del numero naturale n --> -9
Inserire il valore del numero naturale n -->
```

Simulazione con input sbagliato (solo numeri naturali vanno bene).

Evoluzione dello stack nel caso specifico di $n = 2$

User Stack [7ffff950]..[80000000]				
[7ffff950]	0000000000	0000000002	0000000000	0004194420
[7ffff960]	0000000002	0004194328	0000000003	2147482202
[7ffff970]	2147482196	2147482143	0000000000	2147483617

In giallo sono evidenziati il numero inserito da tastiera dall'utente ed il return address per l'Exception Handler, in rosso invece sono evidenziati il valori di \$t0(b), \$t1(n) e \$t2(k) da preservare prima della chiamata della procedura F.

User Stack [7ffff94c]..[80000000]				
[7ffff94c]	0004194644			
[7ffff950]	0000000001	0000000002	0000000001	0004194420
[7ffff960]	0000000002	0004194328	0000000003	2147482202

Qui in particolare vengono evidenziati il return address per tornare alla procedura G e in questo caso in rosso è evidenziato il $k = 1$ (\$t2) che viene passato alla F(k).

User Stack [7ffff944]..[80000000]				
[7ffff944]	0000000000	0000000001	0004194644	
[7ffff950]	0000000001	0000000002	0000000001	0004194420
[7ffff960]	0000000002	0004194328	0000000003	2147482202

In giallo sono evidenziati i valori da preservare di \$a0(n-1) e \$a1(n) da passare ad F(n-1).

User Stack [7ffff940]..[80000000]				
[7ffff940]	0004194908	0000000000	0000000001	0004194644
[7ffff950]	0000000001	0000000002	0000000001	0004194420
[7ffff960]	0000000002	0004194328	0000000003	2147482202

In giallo è evidenziato il return address di F(n) memorizzato da F(n-1).

User Stack [7ffff950]..[80000000]				
[7ffff950]	0000000004	0000000002	0000000002	0004194420
[7ffff960]	0000000002	0004194328	0000000003	2147482202
[7ffff970]	2147482196	2147482143	0000000000	2147483617

In giallo sono evidenziati di \$t0(b), \$t1(n) e \$t2(k) che devono essere preservati da G prima di passare ad F (in questo caso per l'ultima volta perché $k = n = 2$).

User Stack [7ffff944]..[80000000]				
[7ffff944]	0000000001	0000000002	0004194644	
[7ffff950]	0000000004	0000000002	0000000002	0004194420
[7ffff960]	0000000002	0004194328	0000000003	2147482202

In giallo sono evidenziati \$a0 ($n - 1$) e \$a1 (n) e il return address di G. Tutto prima della chiamata ricorsiva ad F($n - 1$).

User Stack [7ffff940]..[80000000]				
[7ffff940]	0004194908	0000000001	0000000002	0004194644
[7ffff950]	0000000004	0000000002	0000000002	0004194420
[7ffff960]	0000000002	0004194328	0000000003	2147482202

In giallo è evidenziato il return address di F(2) memorizzato da F(1).

```
User Stack [7ffff938]..[80000000]
[7ffff938] 0000000000 0000000001
[7ffff940] 0004194908 0000000001 0000000002 0004194644
[7ffff950] 0000000004 0000000002 0000000002 0004194420
[7ffff960] 0000000002 0004194328 0000000003 2147482202
```

Sono evidenziati in giallo \$a0(n-1) e \$a1(n) prima di chiamare F(0).

```
User Stack [7ffff934]..[80000000]
[7ffff934] 0004194908 0000000000 0000000001
[7ffff940] 0004194908 0000000001 0000000002 0004194644
[7ffff950] 0000000004 0000000002 0000000002 0004194420
[7ffff960] 0000000002 0004194328 0000000003 2147482202
```

In giallo evidenziato il return address di F(1). Poi l'esecuzione del programma termina senza altri cambiamenti rilevanti nell'User Stack.

Codice:

Esercizio 2: Procedure annidate e ricorsive.

.data

input: .asciiz "Inserire il valore del numero naturale n --> "

ris: .asciiz "\n\nRisultato: G("

ParentesiA: .asciiz "("

parentesiC: .asciiz ")"

soloF: .asciiz "F"

soloG: .asciiz "G"

trattino: .asciiz "-"

freccia: .asciiz "->"

ritorno: .asciiz "return"

traccia: .asciiz "\nTraccia: "

uguale: .asciiz "="

.text

.globl main

main:

```
addi $sp, $sp, -4          # Salvataggio dell'indirizzo di ritorno al main.
sw $ra, 0($sp)
```

inserimento:

li \$t1, 1 # Per il controllo.

li \$v0, 4 # Stampa della stringa che richiede l'inserimento di n.

la \$a0, input

syscall

li \$v0, 5 # Inserimento di n.

syscall

slt \$t0, \$v0, 0 # Controllo che il numero inserito sia naturale, ovvero ≥ 0 .

beq \$t0, \$t1, inserimento

move \$t7, \$v0

li \$v0, 4 # Stampo la scritta traccia.

la \$a0, traccia

syscall

move \$t6, \$t7 # Salvo n per la stampa.

move \$a0, \$t7 # Passo come parametro n.

procedura. addi \$sp, \$sp, -4 # Salvo \$t6 nello stack prima di chiamare G perche' verra' utilizzato al ritorno dalla

sw \$t6, 0(\$sp)

jal G # Chiamata della procedura G.

lw \$t6, 0(\$sp) # Dealloca dallo stack \$t6 (ripristino dei registri caller-saved).

addi \$sp, \$sp, 4

move \$t5, \$v0 # Per non perdere il valore.

li \$v0, 4	# Stampa della stringa.
la \$a0, ris	
syscall	

li \$v0, 1	# Stampa dell'n inserito da tastiera.
move \$a0, \$t6	
syscall	

li \$v0, 4	# Stampa parentesi chiusa.
la \$a0, parentesiC	
syscall	

li \$v0, 4	# Stampa dell'uguale.
la \$a0, uguale	
syscall	

move \$a0, \$t5	# Metto il valore restituito dalla G in \$a0.
li \$v0, 1	# Stampa del risultato.
syscall	

lw \$ra, 0(\$sp)	# Ripristino dello stack frame.
addi \$sp, \$sp, 4	
jr \$ra	

G:

addi \$sp, \$sp, -4	# Il chiamato si salva il return address.
sw \$ra, 0(\$sp)	

li \$t0, 0	# b = 0.
move \$t1, \$a0	# \$t1 = n.
li \$t2, 0	# Indice del ciclo for.

li \$v0, 4	# Stampa di G.
la \$a0, soloG	

syscall

li \$v0, 4 # Stampa della parentesi aperta.

la \$a0, ParentesiA

syscall

li \$v0, 1 # Stampa di n.

move \$a0, \$t1

syscall

li \$v0, 4 # Stampa parentesi chiusa.

la \$a0, parentesiC

syscall

li \$v0, 4 # Stampa della freccia.

la \$a0, freccia

syscall

cicloFor:

bgt \$t2, \$t1, restituzione

move \$a0, \$t2

Passo k alla procedura

F.

addi \$sp, \$sp, -12

Salvo nello stack i registri non

preservati che userò al ritorno della chiamata a procedura.

sw \$t0, 0(\$sp)

sw \$t1, 4(\$sp)

sw \$t2, 8(\$sp)

jal F

Chiamata

della procedura F.

lw \$t0, 0(\$sp)

Ripristino dei registri

caller - saved.

lw \$t1, 4(\$sp)

lw \$t2, 8(\$sp)

addi \$sp, \$sp, 12

mul \$t0, \$t0, \$t0

b^2

add \$t0, \$t0, \$v0

b^2 + u (valore restituito da F).

addi \$t2, \$t2, 1

Incremento dell'indice del ciclo for

ovvero k.

j cicloFor

restituzione:

move \$v0, \$t0

return b.

li \$v0, 4

Stampa di G.

la \$a0, soloG

syscall

li \$v0, 4

Stampa del trattino.

la \$a0, trattino

syscall

li \$v0, 4

Stampa del ritorno.

la \$a0, ritorno

syscall

li \$v0, 4

Stampa di parentesi aperta.

la \$a0, ParentesiA

syscall

move \$a0, \$t0

Passo il valore da restituire ad \$a0.

li \$v0, 1

Stampa del valore finale.

syscall

li \$v0, 4

Stampa parentesi chiusa.

la \$a0, parentesiC

syscall

move \$v0, \$t0

Metto il risultato in \$v0 per poi tornare al chiamante.

lw \$ra, 0(\$sp)

addi \$sp, \$sp, 4

jr \$ra

F:

addi \$sp, \$sp, -4

Il chiamato si salva il return address.

sw \$ra, 0(\$sp)

move \$t8, \$a0

Mi salvo k per stamparlo.

li \$v0, 4

Stampa di F.

la \$a0, soloF

syscall

li \$v0, 4

Stampa della parentesi aperta.

la \$a0, ParentesiA

syscall

li \$v0, 1

Stampa di k.

move \$a0, \$t8

syscall

li \$v0, 4

Stampa parentesi chiusa.

la \$a0, parentesiC

syscall

li \$v0, 4

Stampa della freccia.

la \$a0, freccia

syscall

```
beq $a0, $zero, restZero
```

```
move $a1, $a0           # Salvo n per sommarlo dopo...
```

```
addi $a0, $a0, -1      # n - 1
```

chiamata a procedura.

```
addi $sp, $sp, -8           # Salvo nello stack i registri non preservati che intendo usare al ritorno della
```

```
sw $a0, 0($sp)
```

sw \$a1, 4(\$sp)

ricorsiva).

```

jal F                                # La procedura F richiama se stessa ( funzione

```

```
lw $a0, 0($sp)      # Ripristino dei registri caller - saved.
```

lw \$a1, 4(\$sp)

```
addi $sp, $sp, 8
```

- \$t3, 2

```
mul $v0, $v0, $t3      # 2 * F ( n - 1 )
```

```
add $v0, $v0, $a1      # 2 * F ( n - 1 ) + n
```

```
move $t9, $v0
```

li \$v0, 4 # Stampa di F.

la \$a0, soloF

syscall

li \$v0, 4 # Stampa del trattino.

la \$a0, trattino

syscall

```
li $v0, 4          # Stampa della scritta return.
```

la \$a0, ritorno

syscall

li \$v0, 4

Stampa della parentesi aperta.

la \$a0, ParentesiA

syscall

move \$a0, \$t9

li \$v0, 1

Stampa del valore restituito.

syscall

li \$v0, 4

Stampa della parentesi chiusa.

la \$a0, parentesiC

syscall

li \$v0, 4

Stampa della freccia.

la \$a0, freccia

syscall

move \$v0, \$t9

chiamante.

lw \$ra, 0(\$sp)

Il chiamato ripristina i registri caller - saved e torna la

addi \$sp, \$sp, 4

jr \$ra

restZero:

Caso base...

li \$v0, 1

In questo caso devo restituire 1.

move \$t9, \$v0

li \$v0, 4

Stampa di F.

la \$a0, soloF

syscall

li \$v0, 4 # Stampa del trattino.

la \$a0, trattino

syscall

li \$v0, 4 # Stampa della scritta return.

la \$a0, ritorno

syscall

li \$v0, 4 # Stampa della parentesi aperta.

la \$a0, ParentesiA

syscall

move \$a0, \$t9 # Rimetto il valore da stampare in \$a0.

li \$v0, 1 # Stampa del valore restituito.

syscall

li \$v0, 4 # Stampa della parentesi chiusa.

la \$a0, parentesiC

syscall

li \$v0, 4 # Stampa della freccia.

la \$a0, freccia

syscall

li \$v0, 1 # Valore che la procedura deve restituire.

chiamante. lw \$ra, 0(\$sp # Il chiamato ripristina i registri caller - saved e torna al

addi \$sp, \$sp, 4

jr \$ra

Esercizio 3

Utilizzando QtSpim, scrivere e provare un programma che visualizzi all'utente un menù di scelta con le seguenti quattro opzioni:

a) **Inserimento di matrici.** Il programma richiede di inserire da tastiera un numero naturale $n > 0$, e richiede quindi l'inserimento di due matrici quadrate, chiamate A e B, di dimensione $n \times n$, contenenti numeri interi. Quindi si ritorna al menù di scelta.

Le matrici A e B dovranno essere allocate dinamicamente in memoria utilizzando la system call 'sbrk' del MIPS, e per loro memorizzazione si consiglia l'utilizzo di una struttura dati a lista.

Ogni volta che si seleziona l'opzione a) del menu, i nuovi valori inseriti di A e B dovranno essere salvati nelle stesse locazioni di memoria in cui erano stati salvati i vecchi valori (per limitare l'utilizzo della memoria), quindi i nuovi valori sovrascriveranno quelli vecchi. Si potrà allocare (con la 'sbrk') uno spazio aggiuntivo di memoria solo se le due nuove matrici dovessero richiedere più spazio di memoria rispetto a quello già allocato in precedenza.

Esempio di interfaccia per l'inserimento delle due matrici:

Dimensione matrici: 3x3

Matrice A:

Riga1: -2 44 5

Riga2: 1 1 1

Riga3: 3 0 1

Matrice B:

Riga1: 0 0 10

Riga2: -1 1 -1

Riga3: 1 0 0

b) **Somma di matrici.** Il programma effettua la somma fra le due matrici A e B, e visualizza su console il risultato $A+B$. Quindi si ritorna al menù di scelta. Ad esempio, se A e B sono state inserite come riportato nell'esempio al punto a), il programma dovrà visualizzare su console:

Risultato di A+B:

Riga1: -2 44 15

Riga2: 0 2 0

Riga3: 4 0 1

c) **Sottrazione di matrici.** Il programma effettua la sottrazione fra le due matrici A e B, e visualizza su console il risultato $A-B$. Quindi si ritorna al menù di scelta.

Ad esempio, se A e B sono state inserite come riportato nell'esempio al punto a), il programma dovrà visualizzare su console:

Risultato di A-B:

Riga1: -2 44 -5

Riga2: 2 0 2

Riga3: 2 0 1

d) **Prodotto di matrici.** Il programma effettua il prodotto fra le due matrici A e B, e visualizza su console il risultato $A*B$. Quindi si ritorna al menù di scelta. Ad esempio, se A e B sono state inserite come riportato nell'esempio al punto a), il programma dovrà visualizzare su console:

*Risultato di A*B:*

Riga1: -39 44 -64

Riga2: 0 1 9

Riga3: 1 0 30

e) Stampa un messaggio di uscita e esce dal programma.

Alle opzioni a), b), c), d) corrisponderanno le chiamate alle opportune procedure utilizzando l'istruzione jal (jump and link). Alla scelta e) corrisponderà l'uscita dal programma.

Mostrare e discutere nella relazione l'evoluzione della memoria dinamica (heap) in alcuni casi di interesse.

Descrizione algoritmi utilizzati

Per lo svolgimento di questo esercizio è stata utilizzata una lista di liste, ovvero per scorrere le colonne della matrice si utilizza una lista (detta secondaria) che è puntata dal corrispondente record della lista principale la quale serve a scorrere le righe. La lista principale è stata realizzata nel seguente modo: il record punta con i primi 4 byte all'elemento successivo della lista principale (in modo da scorrere le righe) e con i secondi 4 byte alla corrispondente lista secondaria. Per quanto riguarda la sovrascrizione della matrice al richiamo dell'operazione di inserimento controlliamo la matrice fino ad n (dimensione della matrice) se i secondi 4 byte dell' n record sono a 0, vuol dire che ho reinserito una matrice di uguale dimensione di quella inserita in precedenza, se invece tale elemento contiene un puntatore significa che la matrice che ho inserito questa volta ha una dimensione minore rispetto a quella in precedenza, perciò mi limito a stampare fino ad n (dimensione matrice corrente) senza mettere a 0 il puntatore (quindi conservandolo). Se invece la dimensione della matrice corrente è maggiore di quella inserita in precedenza semplicemente continuo ad aggiungere record fino ad n , sovrascrivendo tutta la matrice vecchia. Per quanto riguarda somma e sottrazione la matrice viene scorsa e iterativamente vengono prelevati dalla memoria con della load word gli elementi presenti nei record, vengono sommati/ sottratti e subito stampati a video. Per quanto riguarda la

moltiplicazione abbiamo utilizzato una procedura detta prendi Elemento che dati in input un indice i ed un indice j di restituire dalla matrice l'elemento che serve in quel determinato momento, più esternamente la procedura prodotto Matrici (chiamante di prendiElemento) esegue le moltiplicazioni (sommandole in un registro che contiene il risultato della sommatoria) aggiornando gli indici da passare a prendiElemento.

Scelte implementative:

Per rappresentare le matrici abbiamo utilizzato una lista di liste perché permette lo scorrimento verticale con la lista principale(fatta solo da record che contengono puntatori) e quello orizzontale con le liste secondarie che vengono puntate da ogni record della lista principale, quindi ci sembra adatta a risolvere le nostre problematiche. Per quanto riguarda la somma e sottrazione di matrici scorriamo una volta sola le matrici in modo da rendere l'algoritmo efficiente e veloce (appena un risultato è calcolato viene stampato a video senza dover tornare in quel punto della matrice altre volte). Per quanto riguarda la moltiplicazione abbiamo scelto di fare la procedura prendiElemento perché volevamo separare la parte di scansione delle matrici (dato che nella moltiplicazione lo scorrimento è più complesso) da quello di calcolo del risultato, questo per una maggiore leggibilità del codice ed una maggiore chiarezza implementativa. Non si è utilizzato nessun tipo di memoria statica (per costruire le matrici) perché non adatta a questo contesto dinamico, si è utilizzato quindi la syscall 9 che ha permesso l'allocazione di memoria dinamica. La jump_table (indirizzo contenuto nel registro preservato \$s5) implementata nel nostro esercizio in realtà fa sottintendere una memoria statica con struttura simile questa:

```
la $t0, jump_table
la $t1, a
sw $t1, 0($t0)           # Prima scelta.
la $t1, .b
sw $t1, 4($t0)           # Seconda scelta.
la $t1, c
sw $t1, 8($t0)           # Terza scelta.
la $t1, d
sw $t1, 12($t0)          # Quarta scelta.
la $t1, e
sw $t1, 16($t0)          # Quinta scelta.
```

Decidendo quindi di risparmiare codice.

Simulazioni

```
Inserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione di matrici, 3:
Moltiplicazione di matrici, 4: Uscita
0
Dimensione matrici:
2

Matrice A:

Inserire elemento --> 1
Inserire elemento --> 2
Inserire elemento --> 3
Inserire elemento --> 4

1 2
3 4
Matrice B:

Inserire elemento --> 5
Inserire elemento --> 6
Inserire elemento --> 7
Inserire elemento --> 8

5 6
7 8
Inserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione di matrici, 3:
Moltiplicazione di matrici, 4: Uscita
```

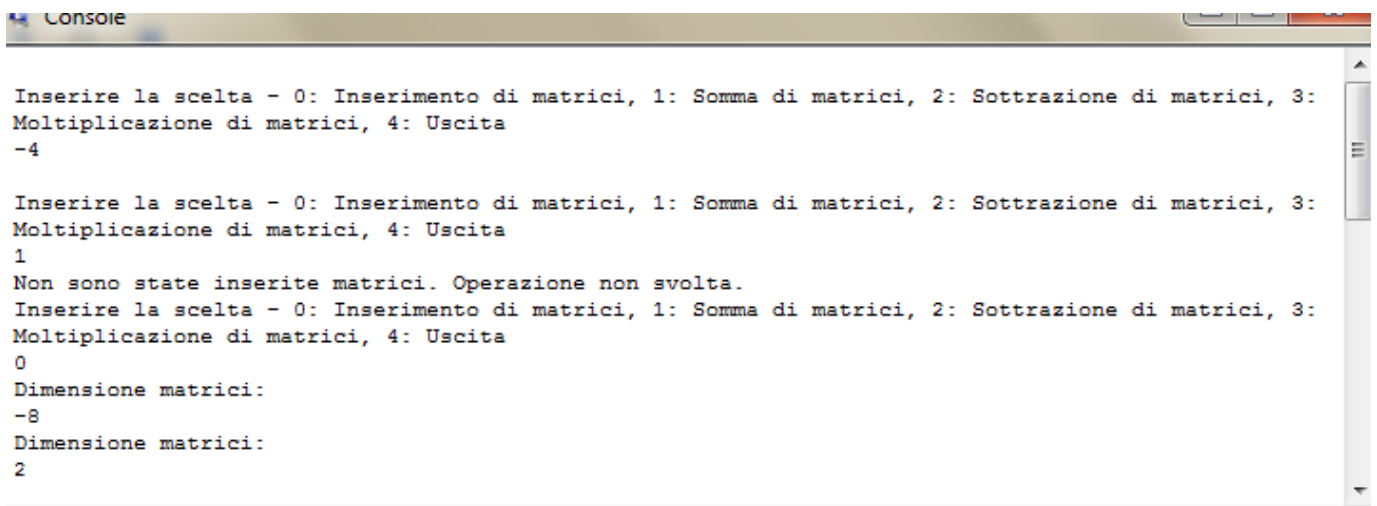
Simulazione nel caso di inserimento delle matrici.

```

Inserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione
di matrici, 3: Moltiplicazione di matrici, 4: Uscita
1
Risultato A + B:
6 8
10 12
Inserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione
di matrici, 3: Moltiplicazione di matrici, 4: Uscita
2
Risultato A - B:
-4 -4
-4 -4
Inserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione
di matrici, 3: Moltiplicazione di matrici, 4: Uscita
3
Risultato A * B:
19 22
43 50
Inserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione
di matrici, 3: Moltiplicazione di matrici, 4: Uscita
4
-- Arrivederci --

```

Simulazione di tutte le operazioni con gli input della simulazione riguardante l'inserimento delle matrici.



```

ConSOLE
Inserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione di matrici, 3:
Moltiplicazione di matrici, 4: Uscita
-4

Inserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione di matrici, 3:
Moltiplicazione di matrici, 4: Uscita
1
Non sono state inserite matrici. Operazione non svolta.
Inserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione di matrici, 3:
Moltiplicazione di matrici, 4: Uscita
0
Dimensione matrici:
-8
Dimensione matrici:
2

```

Simulazione con input sbagliati, infatti non è possibile fare le operazioni sulle matrici se non sono state inserite matrici, se inserisco valori fuori dal range dello switch case e se do valori negativi alla dimensione delle matrici.

```

Console

Inserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione di matrici, 3:
Moltiplicazione di matrici, 4: Uscita
1

Non sono state inserite matrici. Operazione non svolta.

Inserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione di matrici, 3:
Moltiplicazione di matrici, 4: Uscita
2

Non sono state inserite matrici. Operazione non svolta.

Inserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione di matrici, 3:
Moltiplicazione di matrici, 4: Uscita
3

Non sono state inserite matrici. Operazione non svolta.

Inserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione di matrici, 3:
Moltiplicazione di matrici, 4: Uscita

```

Simulazione del caso in cui l'utente selezioni operazioni sulle matrici prima di averle inserite.

Evoluzione dell'User Data Segment:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268697600	268697624	268697608	1	268697616	2	0	0	268697632
268697632	3	268697640	4	0	268697672	268697656	5	268697664
268697664	6	0	0	268697680	7	268697688	8	0
268697696	0	0	0	0	0	0	0	0
268697728	0	0	0	0	0	0	0	0
268697760	0	0	0	0	0	0	0	0
268697792	0	0	0	0	0	0	0	0
268697824	0	0	0	0	0	0	0	0

Adesso trattiamo il caso significativo in cui l'utente inserisca due matrici 2x2, più precisamente: $A = [1,2,3,4]$ e $B = [5,6,7,8]$. Nella figura sono evidenziati gli elementi $A[1,1]$ e $B[1,1]$ che saranno sovrascritti al richiamo successivo dell'opzione "a" del menù di scelta (che è appunto l'inserimento delle matrici).

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268697600	268697624	268697608	33	268697616	2	0	0	268697632
268697632	3	268697640	4	0	268697672	268697656	44	268697664
268697664	6	0	0	268697680	7	268697688	8	0
268697696	0	0	0	0	0	0	0	0
268697728	0	0	0	0	0	0	0	0
268697760	0	0	0	0	0	0	0	0
268697792	0	0	0	0	0	0	0	0
268697824	0	0	0	0	0	0	0	0

In quest' immagine si ha lo User Data Segment che si ottiene dopo aver richiamato una seconda volta l'opzione "a" del menù di scelta in cui l'utente inserisce due matrici 1x1 (A = [33] e B = [44]), abbiamo scelto questo caso perché innanzitutto si evidenzia bene come il 33 vada a sovrascrivere l'1 e come il 44 vada a sovrascrivere il 5. Osserviamo anche che i puntatori ai vecchi valori sono mantenuti, quindi si ha un reale riuso della memoria infatti assume un ruolo determinante la dimensione della matrice che viene utilizzata come limite a cui arrivare per l'acquisizione di nuove matrici ed anche per la stampa di esse. Nel caso in cui avessimo messo a nill i secondi 4 byte degli ultimi record delle matrici si sarebbe perso il puntatore agli elementi della matrice inserita precedentemente lasciando sparsi per la memoria tali valori.

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268697600	268697624	268697608	9	268697616	8	268697696	268697712	268697632
268697632	6	268697640	5	268697704	268697672	268697656	11	268697664
268697664	22	268697744	268697760	268697680	44	268697688	55	268697752
268697696	7	0	4	0	0	268697720	3	268697728
268697728	2	268697736	1	0	33	0	66	0
268697760	0	268697768	77	268697776	88	268697784	99	0
268697792	0	0	0	0	0	0	0	0
268697824	0	0	0	0	0	0	0	0

In questa immagine invece sono evidenziati in giallo i nuovi elementi della matrice A ed in rosso quelli della matrice B nel caso di matrici 3x3 che vanno a sovrascrivere quelle della prima immagine. A = [9,8,7,6,5,4,3,2,1] e B = [11, 22, 33, 44, 55, 66, 77, 88, 99]. Infatti nel caso della matrice A 9, 8, 6 e 5 vanno a sovrascrivere 1,2,3 e 4 mentre 7, 4, 3, 2 e 1 occupano nuove posizioni. Stesso ragionamento vale per la matrice B.

Codice:

Esercizio 3: Matrici.

.data

jump_table: .word a, .b, c, d, e

```
matriceA: .asciiiz "\nMatrice A:\n"

matriceB: .asciiiz "\nMatrice B:\n"

dimensioneM: .asciiiz "\nDimensione matrici:\n"

accapo: .asciiiz "\n"

choice: .asciiiz "\nInserire la scelta - 0: Inserimento di matrici, 1: Somma di matrici, 2: Sottrazione di matrici, 3: Moltiplicazione di matrici, 4: Uscita\n"

inserimentoValore: .asciiiz "\nInserire elemento --> "

spazio: .asciiiz " "

risSomma: .asciiiz "Risultato A + B: "

risSottrazione: .asciiiz "Risultato A - B: "

risProdotto: .asciiiz "Risultato A * B:\n "

msgUscita: .asciiiz "-- Arrivederci --"

nessunaMatrice: .asciiiz "\nNon sono state inserite matrici. Operazione non svolta.\n"
```

```
.text
```

```
.globl main
```

```
main:
```

```
    addi $sp, $sp, -24                                # Variabili globali messe in registri preservati: Verranno
deallocare solamente quando torneremo al chiamante, in questo caso QtSpim( ultimo caso del menu' di scelta).

    sw $s7, 0($sp)                                     # Testa di A.

    sw $s6, 4($sp)                                     # Testa di B.

    sw $s4, 8($sp)                                     # Bisogna preservare anche la dimensione delle matrici
fino il punto a non verrà richiamato per inserirne una nuova.

    sw $s0, 12($sp)                                    # Contatore per capire a che matrice siamo arrivati ad
inserire.

    sw $ra, 16($sp)

    sw $s5, 20($sp)                                    # Indirizzo della jump_table che dovrà essere
preservato per tutto il programma.

    la $s5, jump_table                                # Indirizzo iniziale della jump table.

scelta:

    move $t0, $s5                                     # Reinizializzo l'indirizzo della jump_table.
```

scelta2:

```
li $v0, 4                                # Richiesta della scelta.
```

```
la $a0, choice
```

```
syscall
```

```
li $v0, 5                                # Inserimento della scelta.
```

```
syscall
```

```
move $t2, $v0                            # Copio il contenuto di $v0 in $t2.
```

```
blt $t2, $zero, scelta2                  # Verifica se n < 0
```

```
li $t1, 4
```

```
bgt $t2, $t1, scelta2                    # Verifica se n > 4
```

branch_case:

```
add $t2, $t2, $t2
```

```
add $t2, $t2, $t2
```

```
add $t0, $t0, $t2                        # Incremento dell'indirizzo dell'offset calcolato sopra.
```

```
lw $t0, 0($t0)                           # Leggo l'indirizzo della jump table.
```

```
jr $t0                                    # Salto all'indirizzo in $t0.
```

a:

```
li $s0, 0                                # $s0 = 0 --> Sto facendo A, altrimenti B.
```

insDim:

```
li $v0, 4                                # Richiesta della dimensione delle matrici.
```

```
la $a0, dimensioneM
```

```
syscall
```

```
li $v0, 5                                # Inserimento della dimensione delle matrici.
```


syscall

li \$s4, 0

move \$s4, \$v0
che e' la nostra variabile globale per la dimensione delle matrici.

Quindi il valore inserito da tastiera lo sposto in \$s4

ble \$s4, \$zero, insDim

Controllo che la dimensione inserita sia un valore maggiore di 0.

move \$a0, \$s4

Passo come parametro alla procedura.

move \$a1, \$s7

Passo la testa della matrice A.

move \$a2, \$s6

Passo la testa della matrice B.

jal inserimentoMatrici

j scelta

Si ritorna al menu' di scelta.

inserimentoMatrici:

addi \$sp, \$sp, -4

sw \$s0, 0(\$sp)

move \$t3, \$a0

Salvo n.

bgtz \$s0, InsB

move \$t1, \$a1

\$t1 = testa di A.

li \$v0, 4

Stampa della scritta MatriceA

la \$a0, matriceA

syscall

j inserimentoListaPrinc

InsB:

move \$t1, \$a2

\$t1 = testa di B.

addi \$s0, \$s0, 1

li \$v0, 4

la \$a0, matriceB

syscall

inserimentoListaPrinc:

bne \$t1, \$zero, inizializzazionePuntatori

primoInserimento:

li \$v0, 9

Creazione del primo record.

li \$a0, 8

syscall

move \$t1, \$v0

Puntatore di inizio matrice.

move \$s6, \$t1

inizializzazionePuntatori:

move \$t4, \$t1

t4 scorre la lista principale.

li \$t8, 0

Contatore

cicloEsterno:

li \$t7, 0

Contatore riga.

move \$t6, \$t4

t6 scorre la lista secondaria.

cicloInterno:

beq \$t7, \$t3, fineCicloInterno

Se indice = n devo andare alla riga successiva.

successivo.	lw \$t5, 4(\$t6)	# Prendo l'indirizzo del record
sovrascrivere.	bne \$t5, \$zero, recordSuccessivo	# Se l'indirizzo del record successivo e' diverso da zero devo
quindi devo creare un nuovo record.	li \$v0, 9	# Se arrivo qui significa che e' 0,
	li \$a0, 8	
	syscall	
successivo.	sw \$v0, 4(\$t6)	# Faccio il puntatore al record
	recordSuccessivo:	
all'elemento successivo.	lw \$t6, 4(\$t6)	# Incremento del puntatore
valore da tastiera.	li \$v0, 4	# Viene richiesto l'inserimento del
	la \$a0, inserimentoValore	
	syscall	
tastiera.	li \$v0, 5	# Quindi inserisco il valore da
	syscall	
memoria.	sw \$v0, 0(\$t6)	# Scrivo l'elemento messo in
n(fine della riga).	addi \$t7, \$t7, 1	# Incremento l'indice che mi fa arrivare a
	j cicloInterno	
	fineCicloInterno:	
scorre la lista principale.	addi \$t8, \$t8, 1	# Indica a quale riga sono arrivato, cioè

beq \$t8, \$t3, print

all'elemento successivo della lista principale.

Scorro di riga, cioè vado

record successivo della lista principale lo sovrascrivo.

Stesso discorso di prima, se e' già stato scritto il

il puntatore e' a 0.

li \$v0, 9

Creo nuova memoria dinamica se

li \$a0, 8

syscall

record creato.

sw \$v0, 0(\$t4)

Scrivo il nuovo puntatore nel

rigaSuccessiva:

lw \$t4, 0(\$t4)

Vado alla riga successiva.

j cicloEsterno

print:

bgtz \$s0, diSotto

move \$s7, \$t1

Salvo la testa di A in \$s7.

diSotto:

principale.

move \$t4, \$t1

Uso \$t4 per scorrere la lista

secondaria.

move \$t6, \$t1

Uso \$t6 per scorrere la lista

li \$t7, 0

Contatori di riga e di colonna.

li \$t8, 0

printCicloEsterno:

li \$t7, 0

Contatore riga.

move \$t6, \$t4

t6 scorre la lista secondaria.

li \$v0, 4

Vado a capo

la \$a0, accapo

syscall

loopPrintInterno:

beq \$t7, \$t3, finePrintInterno

Se indice = n devo andare alla riga successiva.

lw \$t6, 4(\$t6)

Prendo l'indirizzo del

record in cui c'è l'elemento da stampare.

lw \$a0, 0(\$t6)

Prendo l'elemento da

stampare.

li \$v0, 1

Stampo l'elemento.

syscall

li \$v0, 4

Spazio.

la \$a0, spazio

syscall

addi \$t7, \$t7, 1

Incremento il contatore di riga.

j loopPrintInterno

finePrintInterno:

addi \$t8, \$t8, 1

Incremento il contatore della lista

principale.

bnez \$s0, diSotto2

beq \$t8, \$t3, InsB

diSotto2:

beq \$t8, \$t3, deallocazione

lw \$t4, 0(\$t4)

Vado alla riga

successiva.

j printCicloEsterno

deallocazione:

move \$v0, \$s7

Testa di A.

move \$v1, \$s6

Testa di B.

lw \$s0, 0(\$sp)

addi \$sp, \$sp, 4

jr \$ra

.b:

beqz \$s7, noMatrici

Controllo per stabilire se quando richiamo questa operazione ho inserito almeno una volta le

matrici.

move \$a2, \$s4

Passo alla procedura n.

move \$a0, \$s7

Passo alla procedura la testa di A.

move \$a1, \$s6

Passo alla procedura la testa di B.

jal sommaMatrici

Procedura che fa la somma tra le matrici A e B.

j scelta

Si ritorna al menu' di scelta.

noMatrici:

Caso in cui vengano richiamate le operazioni sulle matrici senza averle inserite.

li \$v0, 4

la \$a0, nessunaMatrice

syscall

j scelta

sommaMatrici:

addi \$sp, \$sp, -4

Il chiamato si salva i registri preservati che usa.

sw \$s3, 0(\$sp)

move \$t9, \$a2

\$t9 = n (dimensione della matrice).

move \$t3, \$a0

Testa di A come indirizzo iniziale della matrice A. \$t3 = per scorrere le

liste secondarie di A.

move \$t5, \$a0

\$t5 = per scorrere la lista principale di A.

move \$t4, \$a1

Testa di B come indirizzo iniziale della matrice B. \$t4 = per scorrere le

liste secondarie di B.

move \$t6, \$a1

\$t6 = per scorrere la lista principale di B.

li \$t7, 0

Contatore delle liste principali.

li \$t8, 0

Contatore delle liste secondarie.

li \$v0, 4

Risultato A + B.

la \$a0, risSomma

syscall

cicloEsternoSomma:

li \$t8, 0

Quando scorro nella lista principale, reinizializzo gli indici della lista

secondaria.

addi \$t7, \$t7, 1

Incremento il contatore delle due matrici, per la lista principale (scorrimento di riga).

move \$t3, \$t5

Metto il puntatore che scorre la lista principale, in quello che scorre la lista

secondaria.

move \$t4, \$t6

li \$v0, 4

Vado a capo

la \$a0, accapo

syscall

loopInternoSomma:

```
beq $t8, $t9, fineLoopInternoSomma
```

```
lw $t3, 4($t3)          # Prendo l'indirizzo dell'elemento dalla matrice A che andra' sommato.
```

```
lw $t1, 0($t3)          # Prendo l'elemento dalla matrice A.
```

```
lw $t4, 4($t4)          # Prendo l'indirizzo dell'elemento dalla matrice B che andra' sommato.
```

```
lw $t2, 0($t4)          # Prendo l'elemento dalla matrice B.
```

```
add $s3, $t1, $t2      # Sommo a[i,j] + b[i,j].
```

```
move $a0, $s3           # Prendo in argomento di stampa il risultato della somma.
```

```
li $v0, 1               # Stampo il risultato.
```

```
syscall
```

```
li $v0, 4               # Stampo uno spazio
```

```
la $a0, spazio
```

```
syscall
```

```
li $s3, 0               # Riazzero il registro che tiene memoria del risultato della somma.
```

```
addi $t8, $t8, 1        # Incremento gli indici della liste secondarie delle rispettive matrici.
```

```
j loopInternoSomma
```

```
fineLoopInternoSomma:
```

```
beq $t7, $t9, deallocazione2
```

```
lw $t5, 0($t5)          # Vado alla riga successiva nella  
matrice A (scorro la lista principale).
```

```
lw $t6, 0($t6)          # Vado alla riga successiva nella  
matrice B (scorro la lista principale).
```

```
j cicloEsternoSomma
```


deallocazione2:

lw \$s3, 0(\$sp)

Ripristino i registri caller - saved.

addi \$sp, \$sp, 4

jr \$ra

c:

beqz \$s7, noMatrici # Controllo per stabilire se quando richiamo questa operazione ho inserito almeno una volta le
matrici.

move \$a2, \$s4 # Passo alla procedura n.

move \$a0, \$s7 # Passo alla procedura la testa di A.

move \$a1, \$s6 # Passo alla procedura la testa di B.

jal differenzaMatrici # Procedura che fa la somma tra le matrici A e B.

j scelta # PC + 4 al ritorno dalla procedura differenzaMatrici, per tornare al menu' di
scelta.

differenzaMatrici:

addi \$sp, \$sp, -4 # Il chiamato si salva i registri preservati che usa.

sw \$s3, 0(\$sp)

move \$t9, \$a2 # \$t9 = n (dimensione della matrice).

move \$t3, \$a0 # Testa di A come indirizzo iniziale della matrice A. \$t3 = per scorrere le
liste secondarie di A.

move \$t5, \$a0 # \$t5 = per scorrere la lista principale di A.

move \$t4, \$a1 # Testa di B come indirizzo iniziale della matrice B. \$t4 = per scorrere le
liste secondarie di B.

move \$t6, \$a1 # \$t6 = per scorrere la lista principale di B.

li \$t7, 0 # Contatore delle liste principali.

li \$t8, 0 # Contatore delle liste secondarie.

li \$v0, 4 # Risultato A - B

la \$a0, risSottrazione

syscall

cicloEsternoSottrazione:

```
secondaria.    li $t8, 0                # Quando scorro nella lista principale, reinizializzo gli indici della lista

addi $t7, $t7, 1    # Incremento il contatore delle due matrici, per la lista principale ( scorrimento di riga ).

secondaria.    move $t3, $t5            # Metto il puntatore che scorre la lista principale, in quello che scorre la lista
               move $t4, $t6

               li $v0, 4                # Vado a capo
               la $a0, accapo
               syscall
```

loopInternoSottrazione:

```
               beq $t8, $t9, fineLoopInternoSottrazione

               lw $t3, 4($t3)            # Prendo l'indirizzo dell'elemento dalla matrice A che andra' sottratto.
               lw $t1, 0($t3)            # Prendo l'elemento dalla matrice A.

               lw $t4, 4($t4)            # Prendo l'indirizzo dell'elemento dalla matrice B che andra' sottratto.
               lw $t2, 0($t4)            # Prendo l'elemento dalla matrice B.

               sub $s3, $t1, $t2        # Faccio  $a[i,j] - b[i,j]$ .

               move $a0, $s3            # Prendo in argomento di stampa il risultato della differenza.

               li $v0, 1                # Stampo il risultato.
               syscall

               li $v0, 4                # Spazio.

               la $a0, spazio
```

syscall

li \$s3, 0 # Riassetto il registro che tiene memoria del risultato della somma.

addi \$t8, \$t8, 1 # Incremento gli indici della liste secondarie delle rispettive matrici.

j loopInternoSottrazione

fineLoopInternoSottrazione:

beq \$t7, \$t9, deallocazione3

lw \$t5, 0(\$t5) # Vado alla riga successiva nella
matrice A (scorro la lista principale).

lw \$t6, 0(\$t6) # Vado alla riga successiva nella
matrice B (scorro la lista principale).

j cicloEsternoSottrazione

deallocazione3:

lw \$s3, 0(\$sp) # Ripristino i registri caller - saved.

addi \$sp, \$sp, 4

jr \$ra

d:

beqz \$s7, noMatrici # Controllo per stabilire se quando richiamo questa operazione ho inserito almeno una volta le
matrici.

move \$a2, \$s4 # Passo alla procedura n.

move \$a0, \$s7 # Passo alla procedura la testa di A.

move \$a1, \$s6 # Passo alla procedura la testa di B.

jal prodottoMatrici # Chiamata della procedura prodottoMatrici.

j scelta # PC + 4 al ritorno dalla procedura prodottoMatrici, per tornare al menu' di scelta.

prodottoMatrici:

addi \$sp, \$sp, -20 # Chiamato che salva nello stack i registri \$s che vuole alterare e \$ra.

sw \$s0, 0(\$sp)

sw \$s2, 4(\$sp)

sw \$s3, 8(\$sp)

sw \$s4, 12(\$sp)

sw \$ra, 16(\$sp)

li \$s4, 0 # Riavzero \$s4 perche' in questa procedura ci si arriva sempre con la dimensione delle matrici. Riacquisirà il suo valore una volta usciti dalla procedura prodottoMatrici.

move \$t9, \$a2 # \$t9 = n (dimensione della matrice).

move \$t3, \$a0 # Testa di A come indirizzo iniziale della matrice A. \$t3 = per scorrere le liste secondarie di A.

move \$t4, \$a1 # Testa di B come indirizzo iniziale della matrice B. \$t4 = per scorrere le liste secondarie di B.

li \$t7, 1 # Contatore della lista principale di A.

li \$t8, 1 # Contatore delle liste secondarie di A.

li \$t1, 1 # Contatore della lista principale di B.

li \$t2, 1 # Contatore della lista secondaria di B.

li \$v0, 4 # Risultato A * B

la \$a0, risProdotto

syscall

cicloEsternoPrelievo:

move \$a0, \$t3 # Passo la testa di A.

move \$a1, \$t7 # Passo l'indice della lista principale.

move \$a2, \$t8 # Passo l'indice della lista secondaria.

addi \$sp, \$sp, -12 # Salvo nello stack i registri non preservati di cui voglio conservare il valore a ritorno dalla procedura.

sw \$t3, 0(\$sp)

sw \$t7, 4(\$sp)

sw \$t8, 8(\$sp)

jal prendiElemento # Chiamata della procedura prendiElemento.

procedura.

lw \$t3, 0(\$sp) # Ripristino i registri non preservati di cui voglio conservare il valore a ritorno dalla

lw \$t7, 4(\$sp)

lw \$t8, 8(\$sp)

addi \$sp, \$sp, 12

move \$s0, \$v0 # Passo in \$s0 il valore di A[i,j].

move \$a0, \$t4 # Passo la testa di B.

move \$a1, \$t1 # Passo l'indice della lista principale.

move \$a2, \$t2 # Passo l'indice della lista secondaria.

procedura.

addi \$sp, \$sp, -12 # Salvo nello stack i registri non preservati di cui voglio conservare il valore a ritorno dalla

sw \$t4, 0(\$sp)

sw \$t1, 4(\$sp)

sw \$t2, 8(\$sp)

jal prendiElemento # Sistemare i temporanei.

procedura.

lw \$t4, 0(\$sp) # Ripristino i registri non preservati di cui voglio conservare il valore a ritorno dalla

lw \$t1, 4(\$sp)

lw \$t2, 8(\$sp)

addi \$sp, \$sp, 12

move \$s2, \$v0 # Passo in \$s2 il valore di B[i,j].

mul \$s3, \$s2, \$s0 # Moltiplico i due elementi ottenuti con la procedura prendiElemento.

add \$s4, \$s3, \$s4 # \$s4 = sommatoria righe per colonne corrente.

```

multiplicazioni.    beq $t8, $t9, stampa    # Alla fine di ogni riga devo stampare la sommatoria accumulata con le

                    addi $t8, $t8, 1    # Incremento indice lista secondaria di A.
                    addi $t1, $t1, 1    # Incremento indice lista principale di B.

                    j cicloEsternoPrelievo

stampa:

                    li $v0, 4            # Spazio.
                    la $a0, spazio
                    syscall

                    move $a0, $s4

                    li $v0, 1
                    syscall

matrice.            bne $t2, $t9, ok    # Controllo per stampare i risultati in forma di

                    li $v0, 4            # Vado a capo.
                    la $a0, accapo
                    syscall

                    li $v0, 4            # Spazio.
                    la $a0, spazio
                    syscall

                    ok:

sommatoria.        li $s4, 0            # Reinizializzo il risultato della

                    beq $t7, $t9, controlloFinale    # Caso in cui sono arrivato a fine delle matrici.

```

j avanti

controlloFinale:

beq \$t2, \$t9, deallocazione4

avanti:

in A. beq \$t2, \$t9, incrementPrincA # Se sono arrivato alla fine delle colonne di B devo cambiare riga

B. addi \$t2, \$t2, 1 # Altrimenti vado alla colonna successiva di

secondaria di A all'inizio. li \$t8, 1 # Riporto l'indice della lista

principale di B all'inizio. li \$t1, 1 # Riporto l'indice della lista

j cicloEsternoPrelievo

incrementPrincA:

addi \$t7, \$t7, 1

li \$t8, 1

li \$t2, 1

B dovrà ripartire dalla prima posizione.

li \$t1, 1

j cicloEsternoPrelievo

prendiElemento:

addi \$sp, \$sp, -16

sw \$s1, 0(\$sp)

sw \$s5, 4(\$sp)

sw \$s6, 8(\$sp)

sw \$s7, 12(\$sp)

principale. move \$t5, \$a0 # \$t5 = puntatore della lista

 move \$s1, \$a1 # Salvo l'indice della lista
principale(limite alla quale bisogna arrivare nella lista principale).

 move \$t0, \$a2 # Salvo l'indice della lista
secondaria(limite alla quale bisogna arrivare nella lista secondaria).

(contatore). li \$s5, 1 # Indice della lista secondaria

(contatore). li \$s6, 1 # Indice della lista principale

cicloEsternoScansione:

 beq \$s6, \$s1, cicloInternoScansione

della lista principale. addi \$s6, \$s6, 1 # Incremento fino ad arrivare al punto giusto

 lw \$t5, 0(\$t5) # Scorro la lista principale.

j cicloEsternoScansione

cicloInternoScansione:

 lw \$t5, 4(\$t5)

cercata. lw \$s7, 0(\$t5) # Prendo l'elemento nella posizione

 beq \$s5, \$t0, restituzione # Quando si e' arrivati all'indice passato come
parametro alla procedura si può restituire l'elemento corrispondente.

 addi \$s5, \$s5, 1 # Incremento il contatore della lista
secondaria perche' devo arrivare all'elemento chiesto.

j cicloInternoScansione

restituzione:

```
move $v0, $s7
ripristino i registri caller - saved.

lw $s1, 0($sp)
lw $s5, 4($sp)
lw $s6, 8($sp)
lw $s7, 12($sp)
addi $sp, $sp, 16
jr $ra
```

Restituisco l'elemento cercato e

deallocazione4:

```
lw $s0, 0($sp)
ritorno al chiamante.

lw $s2, 4($sp)
lw $s3, 8($sp)
lw $s4, 12($sp)
lw $ra, 16($sp)
addi $sp, $sp, 20
jr $ra
```

Ripristino i registri caller - saved e

e:

```
li $v0, 4
la $a0, msgUscita
syscall
```

Messaggio d'uscita.

```
lw $s7, 0($sp)
QtSpim ), dopo aver ripristinato i registri
```

Exception Handler (ritorno a

```
lw $s6, 4($sp)
lw $s4, 8($sp)
lw $s0, 12($sp)
lw $ra, 16($sp)
lw $s5, 20($sp)
```

addi \$sp, \$sp, 24

jr \$ra