SIMULATORE DI CHIAMATE A PROCEDURA

Autori:
Gabriele Bertini – gabriele.bertini3@stud.unifi.it
Lorenzo Pratesi Mariti – lorenzo.pratesi@stud.unifi.it

Data di consegna:
29 maggio 2016

Introduzione:

Il programma alloca 156 byte in memoria in cui memorizzarci la stringa presa in input dal file tramite la funzione *letturaFile*. Successivamente legge il terzo carattere della stringa perché è il primo per cui differiscono tutte e 4 le operazioni aritmetiche attraverso la procedura *readChar* e in base al carattere trovato salta ad una delle 4 etichette le quali richiamano le procedure *somma*, *sottrazione*, *prodotto* e *divisione.*
Ciascuna funzione avanza il puntatore al carattere successivo alla parentesi aperta e richiama la procedura *check* che verifica se il carattere è un numero intero.
Se il carattere è maggiore del valore ascii 58 allora è una lettera e richiama ricorsivamente la procedura *readChar*, altrimenti salta all'etichetta *esciCheck* nella quale viene chiamata la funzione *prendiIntero* che restituisce il numero intero letto.

Descrizione delle procedure:

- Main

Parametri in ingresso: nessuno.
Descrizione:

- richiama la procedura *letturaFile*;
- carica l'indirizzo del buffer in cui è contenuta la stringa presa in input da file;
- richiama la procedura *readChar*;
- stampa il risultato.

Valori di ritorno: nessuno.

- ReadChar

Parametri in ingresso: puntatore al buffer.
Descrizione:

- avanza il puntatore di 2 posizioni e controlla il carattere a cui punta;
- salta ad una di quattro etichette secondo il carattere a cui sta puntando il puntatore;
- richiama la funzione dentro l'etichetta alla quale è saltato.

Valori di ritorno: nessuno.

- Somma

Parametri in ingresso: puntatore al buffer.
Descrizione:

- alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp) e il valore di ritorno della funzione *check* (sp+4);
- avanza il puntatore di 4 posizioni;
- richiama la funzione *check*;

- avanza il puntatore di 1 posizione;
- richiama la funzione *check*;
- riprende il valore di ritorno della prima chiamata a *check*;
- somma i due valori;
- riprende l'indirizzo di ritorno del chiamante;
- dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: risultato dell'operazione *somma*.

- **Sottrazione**

Parametri in ingresso: puntatore al buffer.
Descrizione:

- alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp) e il valore di ritorno della funzione *check* (sp+4);
- avanza il puntatore di 10 posizioni;
- richiama la funzione *check*;
- avanza il puntatore di 1 posizione;
- richiama la funzione *check*;
- riprende il valore di ritorno della prima chiamata a *check*;
- sottrae i due valori;
- riprende l'indirizzo di ritorno del chiamante;
- dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: risultato dell'operazione *sottrazione*.

- **Prodotto**

Parametri in ingresso: puntatore al buffer.
Descrizione:

- alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp) e il valore di ritorno della funzione *check* (sp+4);
- avanza il puntatore di 7 posizioni;
- richiama la funzione *check*;
- avanza il puntatore di 1 posizione;
- richiama la funzione *check*;
- riprende il valore di ritorno della prima chiamata a *check*;
- moltiplica i due valori;
- riprende l'indirizzo di ritorno del chiamante;
- dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: risultato dell'operazione *prodotto*.

- **Divisione**

Parametri in ingresso: puntatore al buffer.
Descrizione:

- alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp) e il valore di ritorno della funzione *check* (sp+4);
- avanza il puntatore di 8 posizioni;
- richiama la funzione *check*;
- avanza il puntatore di 1 posizione;
- richiama la funzione *check*;

- controlla se il valore ritornato dalla seconda chiamata a *check* è 0 ed in tal caso salta all'etichetta *divZero* che stampa un messaggio di errore e termina il programma;
- riprende il valore di ritorno della prima chiamata a *check*;
- divide i due valori;
- riprende l'indirizzo di ritorno del chiamante;
- dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: risultato dell'operazione *divisione*.

- Check

Parametri in ingresso: puntatore al buffer.
Descrizione:

- legge il carattere a cui punta il puntatore e controlla se il suo valore ascii è minore di 58;
- se lo è allora vai all'etichetta *esciCheck* che richiama la procedura *prendiIntero*;
- altrimenti richiama ricorsivamente la funzione *check*.

Valori di ritorno: indirizzo del puntatore al buffer, numero intero.

- PrendiIntero

Parametri in ingresso: puntatore al buffer.
Descrizione:

- carica il numero intero 10 per i numeri a più cifre;
- legge il carattere a cui punta il puntatore e controlla se è una virgola (va all'etichetta *esci* e torna al chiamante) o una parentesi chiusa (va all'etichetta *fineStringa* che a sua volta controlla se il carattere successivo è un byte 0 e cioè la fine della stringa);
- altrimenti sottrae 48 dal valore ascii del carattere per ottenere il numero intero;
- moltiplica per 10 e somma l'intero;
- avanza il puntatore di 1 posizione;
- ricomincia il ciclo.

Valori di ritorno: indirizzo del puntatore al buffer, numero intero.

PROCEDURE DI STAMPA

-StampaTraccia

Parametri in ingresso: puntatore al buffer.
Descrizione:

- conta le parentesi aperte scorrendo la stringa
- conta le parentesi chiuse scorrendo la stringa
- trova il punto di ritorno traccia, salva tale indirizzo/valore in un registro temporaneo
- mette il carattere “fine stringa” all’indirizzo puntato precedentemente
- stampa la sotto-stringa
- rimette apposto il carattere estratto precedentemente
- torna al chiamante

Valori di ritorno: nessuno

-StampaTracciaRitorno

Parametri in ingresso: puntatore al buffer, id funzione, valore da stampare.
Descrizione:

- sceglie il nome della funzione da stampare in base all’id in igresso
- stampa il nome della funzione
- stampa la stringa con il valore calcolato dalle funzioni precedenti
- torna al chiamante

Valori di ritorno: nessuno

Simulazione:

La simulazione è stata eseguita sulla seguente stringa:
somma(7,somma(sottrazione(0,5),prodotto(divisione(7,2),3)))

```
PC        = 40002c
EPC       = 0
Cause     = 0
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 10
R3  [v1] = 0
R4  [a0] = 4
R5  [a1] = 7ffff658
R6  [a2] = 7ffff66c
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 10008000
R29 [sp] = 7ffff650
R30 [s8] = 0
R31 [ra] = 400018
```

L'immagine si riferisce all'inizio del programma e come possiamo vedere nel registro *ra* abbiamo caricato l'indirizzo di ritorno del main, mentre il registro *sp* (stack pointer, che inizialmente era 7ffff654) è stato decrementato dei 4 byte allocati in memoria.

```
User Stack [7ffff650]..[80000000]
[7ffff650]    00400018  00000004  7ffff7be  7ffff7b8
```

Successivamente il contenuto di *ra* viene caricato nello stack.

Al passo successivo salviamo nel registro *a0* l'indirizzo del puntatore al buffer in cui abbiamo caricato la stringa presa in input da file e mostriamo come, attraverso la funzione *letturaFile,* la stringa è stata caricata nello User Data Segment a partire dall'indirizzo 10010034:

```
R0  [r0] = 0
R1  [at] = 10010000
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 10010034
```

```
User data segment [10000000]..[10040000]
[10000000]..[1000ffff]  00000000
[10010000]    20656854  656c6966  73617720  746f6e20    T h e   f i l e   w a s   n o t
[10010010]    756f6620  203a646e  61696863  6574616d      f o u n d :   c h i a m a t e
[10010020]    7478742e  6c694600  6f632065  6e65746e    . t x t . F i l e   c o n t e n
[10010030]    203a7374  6d6d6f73  2c372861  6d6d6f73    t s :   s o m m a ( 7 , s o m m
[10010040]    6f732861  61727474  6e6f697a  2c302865    a ( s o t t r a z i o n e ( 0 ,
[10010050]    702c2935  6f646f72  286f7474  69766964    5 ) , p r o d o t t o ( d i v i
[10010060]    6e6f6973  2c372865  332c2932  00292929    s i o n e ( 7 , 2 ) , 3 ) ) ) .
[10010070]..[100100cf]  00000000
[100100d0]    6552000a  746c7573  4500203a  726f7272    . . R e s u l t :   . E r r o r
[100100e0]    64203a65  73697669  656e6f69  72657020    e :   d i v i s i o n e   p e r
[100100f0]    72657a20  0000006f  00000000  00000000      z e r o . . . . . .
[10010100]..[1003ffff]  00000000
```

Adesso andiamo a vedere come lavora il programma durante le chiamate alle funzioni *somma*, *sottrazione*, *prodotto* e *divisione*:

- <u>1° chiamata alla funzione *somma*</u>

Possiamo notare come l'indirizzo del puntatore al buffer sia avanzato di due byte e punti alla lettera "m".
Notiamo inoltre come l'indirizzo di ritorno del chiamante (la procedura *readChar*) sia stato salvato nello stack all'indirizzo del registro *sp.*

```
User Stack [7ffff644]..[80000000]
[7ffff644]    004000a8  00000000  0040003c
[7ffff650]    00400018  00000004  7ffff7be  7ffff7b8
```

```
R4  [a0] = 10010036
R5  [a1] = 10010034
R6  [a2] = 9c
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 6d
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 9
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 10008000
R29 [sp] = 7ffff644
R30 [s8] = 0
R31 [ra] = 4000a8
```

- 2° chiamata alla funzione *somma*

In queste immagini osserviamo come nella chiamata precedente, dopo la procedura *check*, abbiamo memorizzato in memoria il valore 7 all'indirizzo 7ffff648.

```
R0  [r0] = 0
R1  [at] = 6d
R2  [v0] = 1001003b
R3  [v1] = 7
R4  [a0] = 1001003e
R5  [a1] = 10010034
R6  [a2] = 9c
R7  [a3] = 0
R8  [t0] = 73
R9  [t1] = 6d
R10 [t2] = 7
R11 [t3] = a
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 9
```

```
User Stack [7ffff634]..[80000000]
[7ffff634]    004000a8  00000000  00400224
[7ffff640]    00400108  004000a8  00000007  0040003c
[7ffff650]    00400018  00000004  7ffff7be  7ffff7b8
```

- 1° chiamata alla funzione *sottrazione*

Il valore 0 viene salvato nello stack all'indirizzo 7ffff628 per poi essere ripreso prima di effettuare la sottrazione fra i due numeri

```
User Stack [7ffff620]..[80000000]
[7ffff620]    00400148  004000b8  00000000  00400224
[7ffff630]    004000f4  004000a8  00000000  00400224
[7ffff640]    00400108  004000a8  00000007  0040003c
[7ffff650]    00400018  00000004  7ffff7be  7ffff7b8
```

Sottraendo il valore 0 (contenuto nel registro *s1*) con il valore 5 (contenuto nel registro *t6*) possiamo notare che il risultato è fffffffb (contenuto nel registro *s0*) in esadecimale, cioè -5 in esadecimale.

```
R14 [t6] = 5
R15 [t7] = 0
R16 [s0] = fffffffb
R17 [s1] = 0
```

- Ritorno alla 2° chiamata alla funzione *somma*

Memorizzo il valore di ritorno della procedura *sottrazione* nello stack all'indirizzo 7ffff638.

```
User Stack [7ffff634]..[80000000]
[7ffff634]    004000a8  fffffffb  00400224
[7ffff640]    00400108  004000a8  00000007  0040003c
[7ffff650]    00400018  00000004  7ffff7be  7ffff7b8
[7ffff660]    7ffff7a3  7ffff775  00000000  7fffffed
```

- 1° chiamata alla funzione *prodotto*

Osserviamo come l'indirizzo di ritorno del chiamante (la procedura *readChar*) sia stato salvato nello stack all'indirizzo del registro *sp*.

```
R29 [sp] = 7ffff624
R30 [s8] = 0
R31 [ra] = 4000c8
```

```
User Stack [7ffff624]..[80000000]
[7ffff624]    004000c8  00000000  00400224
[7ffff630]    00400108  004000a8  fffffffb  00400224
[7ffff640]    00400108  004000a8  00000007  0040003c
[7ffff650]    00400018  00000004  7ffff7be  7ffff7b8
```

- 1° chiamata alla funzione *divisione*

Calcoliamo nel registro *s1* la divisione degli interi 7 e 2 ed essendo una divisione fra interi il risultato sarà 3 perché troncato.

```
R14 [t6] = 2
R15 [t7] = 0
R16 [s0] = 3
R17 [s1] = 7
```

- Ritorno alla 1° chiamata alla funzione *prodotto*

Memorizzo il valore di ritorno della procedura *divisione* nello stack e dopo aver preso l'intero successivo calcolo il risultato della funzione *prodotto*.

```
User Stack [7ffff620]..[80000000]
[7ffff620]    00400188  004000c8  00000003  00400224
[7ffff630]    00400108  004000a8  fffffffb  00400224
```

```
R14 [t6] = 3
R15 [t7] = 0
R16 [s0] = 9
R17 [s1] = 3
```

- Ritorno alla 2° chiamata alla funzione *somma*

Riprendo il valore di ritorno della funzione *sottrazione* dallo stack e lo sommo con il valore di ritorno della funzione *prodotto.*

```
R14 [t6] = 9
R15 [t7] = 0
R16 [s0] = 4
R17 [s1] = fffffffb
```

- Ritorno alla 1° chiamata alla funzione *somma*

Riprendo il valore7 salvato in precedenza nello stack all'indirizzo di memoria 7ffff648 e lo sommo con il valore di ritorno della funzione *somma* per ottenere il valore b in esadecimale, cioè 11 in decimale.

```
R14 [t6] = 4
R15 [t7] = 0
R16 [s0] = b
R17 [s1] = 7
```

Carico nel registro *ra* l'indirizzo di ritorno della procedura *readChar* che si trova in memoria all'indirizzo 7ffff644.

```
User Stack [7ffff620]..[80000000]
[7ffff620]    00400188  004000c8  00000003  00400224
[7ffff630]    00400108  004000a8  fffffffb  00400224
[7ffff640]    00400108  004000a8  00000007  0040003c
[7ffff650]    00400018  00000004  7ffff7be  7ffff7b8
[7ffff660]    7ffff7a3  7ffff775  00000000  7fffffed
```

- Ritorno alla funzione Main

Carico nel registro *ra* l'indirizzo di ritorno della procedura *main* che si trova all'indirizzo di memoria 7ffff654.
Restituisco il risultato della funzione *somma* e stampo il risultato.
Successivamente il programma termina la sua esecuzione.

```
PC        = 400018
EPC       = 0
Cause     = 0
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 9
```


```
User Stack [7ffff620]..[80000000]
[7ffff620]    00400188  004000c8  00000003  00400224
[7ffff630]    00400108  004000a8  fffffffb  00400224
[7ffff640]    00400108  004000a8  00000007  0040003c
[7ffff650]    00400018  00000004  7ffff7be  7ffff7b8
[7ffff660]    7ffff7a3  7ffff775  00000000  7fffffed
```


```
R0  [r0] = 0
R1  [at] = 10010000
R2  [v0] = 1
R3  [v1] = b
R4  [a0] = b
R5  [a1] = 10010034
R6  [a2] = 9c
R7  [a3] = 0
R8  [t0] = 1001006f
R9  [t1] = 29
R10 [t2] = 3
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 4
R15 [t7] = 0
R16 [s0] = b
R17 [s1] = 7
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 10008000
R29 [sp] = 7ffff654
R30 [s8] = 0
R31 [ra] = 400018
```

Codice:

```
#SIMULATORE DI CHIAMATE A PROCEDURA

# Autori:
# Gabriele Bertini – gabriele.bertini3@stud.unifi.it
# Lorenzo Pratesi Mariti – lorenzo.pratesi@stud.unifi.it
#
# Data di consegna:
# 29 maggio 2016

# Il file "chiamate.txt" deve trovarsi nella stessa cartella in cui si trova l'eseguibile QtSpim
.data
fnf:    .ascii   "The file was not found: "
file:   .asciiz  "chiamate.txt"
cont:   .ascii   "File contents: "
buffer: .space   156
strSomma: .asciiz "somma"
strSottr: .asciiz "sottrazione"
strProd: .asciiz "prodotto"
strDiv: .asciiz "divisione"
strReturn: .asciiz "-return("
aCapo:  .asciiz  "\n"
result: .asciiz  "Result: "
strClose:  .asciiz ")\n"
freccIaIn: .asciiz "-->"
frecciaOut: .asciiz "<--"
error:  .asciiz  "Errore: divisione per zero"


.text
.globl  main

main:
      # PUSH nello Stack
      addi $sp, $sp, -4 # alloco memoria
      sw   $ra, 0($sp)  # salvo ra nello stack

      jal  letturaFile  # richiamo la procedura letturaFile

      la   $a0, aCapo
      li   $v0, 4
      syscall

      la   $a0, buffer  # carico in $a0 l'indirizzo del buffer
      jal  readChar     # richiamo la procedura readChar che opera sulla stringa in input

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

```
    la   $a0, result  # carico in $a0 la stringa result per stamparla
    li   $v0, 4
    syscall

    move $a0, $v1     # carico il risultato delle operazioni in $a0 per stamparlo
    li   $v0, 1
    syscall

    # POP nello Stack:
    lw   $ra, 0($sp)  # riprendo ra
    addi $sp, $sp, 4  # dealloco
    jr   $ra        # torno al chiamante (exeption handler)

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
readChar:
    addi $sp, $sp, -4            # alloco memoria
    sw   $ra, 0($sp)             # salvo ra nello stack

    move $a1, $a0
    jal stampaTraccia
    addi $a0, 2                 # avanzo il puntatore di 2 byte
    lb   $t1, 0($a0)             # carico il carattere a cui punta il puntatore in $t1

    beq  $t1, 'm', chiamaSomma      # controllo se la lettera e' una "m", quindi richiamo la
procedura chiamaSomma
    beq  $t1, 't', chiamaSottrazione # controllo se la lettera e' una "t", quindi richiamo la
procedura chiamaSottrazione
    beq  $t1, 'o', chiamaProdotto    # controllo se la lettera e' una "o", quindi richiamo la
procedura chiamaProdotto
    beq  $t1, 'v', chiamaDivisione   # controllo se la lettera e' una "v", quindi richiamo la
procedura chiamaDivisione

    chiamaSomma:
       jal  somma               # richiamo la procedura somma
       li $t1, 'm'
       move $a1, $t1
       jal stampaTracciaRitorno
       lw   $ra, 0($sp)          # riprendo ra
       addi $sp, $sp, 4          # dealloco
       jr   $ra                # torno al chiamante (exeption handler)

    chiamaSottrazione:
       jal  sottrazione          # richiamo la procedura sottrazione
       li $t1, 't'
       move $a1, $t1
       jal stampaTracciaRitorno
       lw   $ra, 0($sp)          # riprendo ra
       addi $sp, $sp, 4          # dealloco
       jr   $ra                # torno al chiamante (exeption handler)

    chiamaProdotto:
       jal  prodotto            # richiamo la procedura prodotto
```

```
        li $t1, 'o'
        move $a1, $t1
        jal stampaTracciaRitorno

        lw   $ra, 0($sp)          # riprendo ra
        addi $sp, $sp, 4          # dealloco
        jr   $ra                  # torno al chiamante (exeption handler)

    chiamaDivisione:
        jal  divisione            # richiamo la procedura divisione
        li $t1, 'v'
        move $a1, $t1
        jal stampaTracciaRitorno

        lw   $ra, 0($sp)          # riprendo ra
        addi $sp, $sp, 4          # dealloco
        jr   $ra                  # torno al chiamante (exeption handler)

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
somma:
    addi $sp, $sp, -8  # alloco memoria
    sw   $ra, 0($sp)   # salvo ra nello stack

    addi $a0, 4       # avanzo il puntatore di 4 byte
    jal  check        # richiamo la funzione check
    move $t0, $v0     # copio il valore di ritorno (indirizzo del puntatore)
    sw   $v1, 4($sp)  # salvo nello stack il valore di ritorno (numero)
    addi $t0, 1       # avanzo il puntatore di 1 byte
    move $a0, $t0     # copio l'indirizzo del puntatore per passarlo alla funzione check
    jal  check        # richiamo la funzione check
    move $t6, $v1     # carico il contenuto a cui punta il puntatore in $t6
    lw   $s1, 4($sp)  # riprendo il numero che avevo salvato nello stack
    add  $s0, $s1, $t6 # sommo il contenuto dei due registri e lo carico in $s0
    move $v1, $s0     # carico il risultato in $v1 come valore di ritorno per il chiamante

    lw   $ra, 0($sp)  # riprendo ra
    addi $sp, $sp, 8  # dealloco
    jr   $ra          # torno al chiamante (exeption handler)
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
sottrazione:
    addi $sp, $sp, -8  # alloco memoria
    sw   $ra, 0($sp)   # salvo ra nello stack

    addi $a0, 10      # avanzo il puntatore di 10 byte
    jal  check        # richiamo la funzione check
    move $t0, $v0     # copio il valore di ritorno (indirizzo del puntatore)
    sw   $v1, 4($sp)  # salvo nello stack il valore di ritorno (numero)
    addi $t0, 1       # avanzo il puntatore di 1 byte
```

```
    move $a0, $t0     # copio l'indirizzo del puntatore per passarlo alla funzione check
    jal  check        # richiamo la funzione check
    move $t6, $v1     # carico il contenuto a cui punta il puntatore in $t6
    lw   $s1, 4($sp)  # riprendo il numero che avevo salvato nello stack
    sub  $s0, $s1, $t6 # sottraggo il contenuto dei due registri e lo carico in $s0
    move $v1, $s0     # carico il risultato in $v1 come valore di ritorno per il chiamante

    lw   $ra, 0($sp)  # riprendo ra
    addi $sp, $sp, 8  # dealloco
    jr   $ra          # torno al chiamante (exeption handler)
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
prodotto:
    addi $sp, $sp, -8  # alloco memoria
    sw   $ra, 0($sp)   # salvo ra nello stack

    addi $a0, 7       # avanzo il puntatore di 7 byte
    jal  check        # richiamo la funzione check
    move $t0, $v0     # copio il valore di ritorno (indirizzo del puntatore)
    sw   $v1, 4($sp)  # salvo nello stack il valore di ritorno (numero)
    addi $t0, 1       # avanzo il puntatore di 1 byte
    move $a0, $t0     # copio l'indirizzo del puntatore per passarlo alla funzione check
    jal  check        # richiamo la funzione check
    move $t6, $v1     # carico il contenuto a cui punta il puntatore in $t6
    lw   $s1, 4($sp)  # riprendo il numero che avevo salvato nello stack
    mul  $s0, $s1, $t6 # moltiplico il contenuto dei due registri e lo carico in $s0
    move $v1, $s0     # carico il risultato in $v1 come valore di ritorno per il chiamante

    lw   $ra, 0($sp)  # riprendo ra
    addi $sp, $sp, 8  # dealloco
    jr   $ra          # torno al chiamante (exeption handler)
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
divisione:
    addi $sp, $sp, -8       # alloco memoria
    sw   $ra, 0($sp)        # salvo ra nello stack

    addi $a0, 8             # avanzo il puntatore di 8 byte
    jal  check              # richiamo la funzione check
    move $t0, $v0           # copio il valore di ritorno (indirizzo del puntatore)
    sw   $v1, 4($sp)        # salvo nello stack il valore di ritorno (numero)
    addi $t0, 1             # avanzo il puntatore di 1 byte
    move $a0, $t0           # copio l'indirizzo del puntatore per passarlo alla funzione check
    jal  check              # richiamo la funzione check
    move $t6, $v1           # carico il contenuto a cui punta il puntatore in $t6
    beq  $t6, $zero, divZero # gestisco il caso in cui il denominatore e' 0
    lw   $s1, 4($sp)        # riprendo il numero che avevo salvato nello stack
    div  $s0, $s1, $t6      # divido il contenuto dei due registri e lo carico in $s0
    move $v1, $s0           # carico il risultato in $v1 come valore di ritorno per il chiamante
```

```
    lw   $ra, 0($sp)       # riprendo ra
    addi $sp, $sp, 8       # dealloco
    jr   $ra             # torno al chiamante (exeption handler)

  divZero:
      la $a2, error      # carico in $a0 la stringa error per stamparla
      li $v0, 4
      syscall
      li $v0, 10        # esco dal programma
      syscall
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
check:
    addi $sp, $sp, -4          # alloca memoria
    sw   $ra, 0($sp)           # salva ra nello stack

    lb   $t0, 0($a0)           # leggo il carattere della stringa in input a cui punta il
puntatore
    li   $t4, 1                #
    beq  $t0, '-', numeroNegativo # controllo se il numero e' negativo

    slt  $t1, $t0, 58          # controllo se il valore ascii del carattere e' minore di 58
    bne  $t1, $zero, esciCheck    # vado all'etichetta esciCheck se lo e'
    jal  readChar              # il carattere e' una lettera quindi richiamo la funzione
readChar
    lw   $ra, 0($sp)           # riprendo ra
    addi $sp, $sp, 4           # dealloco
    jr   $ra                 # torno al chiamante (exeption handler)

    esciCheck:
      jal  prendiIntero       # il carattere e' un intero quindi richiamo la funzione
prendiIntero
      mul  $v1, $v1, $t4      # moltiplico l'intero per il contenuto di $t4 per ottenere un
valore positivo/negativo
      lw   $ra, 0($sp)        # riprendo ra
      addi $sp, $sp, 4        # dealloco
      jr   $ra              # torno al chiamante (exeption handler)

    numeroNegativo:
      li   $t4, -1           # carico -1 per ottenere l'intero negativo
      addi $a0, 1            # avanzo il puntatore di 1 byte
      j    esciCheck         # vai all'etichetta esciCheck
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
prendiIntero:
    li   $t1, 0              # azzero il registro $t1
    li   $t2, 0              # azzero il registro $t2
    li   $t3, 10              # carico l'intero 10 in $t3
    move $t0, $a0               # copio il contenuto di $a0 (indirizzo del puntatore) in $t0
```

```
    ciclo:
        lb   $t1, 0($t0)         # memorizzo il contenuto di $t0 in $t1
        beq  $t1, ',', esci       # chiudo il ciclo e vado all'etichetta esci se il carattere e' una
virgola
        beq  $t1, ')', fineStringa # chiudo il ciclo e vado all'etichetta fineStringa se il
carattere e' una parentesi chiusa
        addi $t1, -48           # trasformo in intero sottraendo 48 dal valore ascii
        mul  $t2, $t2, $t3        # moltiplico per 10 il contenuto di $t2
        add  $t2, $t2, $t1        # sommo decine con unita'
        addi $t0, 1             # sposto il puntatore di una posizione
        j    ciclo              # continuo il ciclo tornando all'etichetta ciclo
    esci:
        move $v0, $t0           # copio l'indirizzo del puntatore per restituirlo al chiamante
        move $v1, $t2           # copio il valore per restituirlo al chiamante
        jr   $ra                # chiudo la funzione e torno al chiamante

    fineStringa:
        addi $t0, 1             # sposto il puntatore di una posizione
        lb   $t3, 0($t0)         # memorizzo il contenuto a cui punta $t0 in $t3
        beqz $t3, esci           # vado all'etichetta esci se il carattere e' un byte zero (fine
stringa)
        j    ciclo              # continuo il ciclo tornando all'etichetta ciclo perche' ancora non
ho trovato la fine della stringa
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
letturaFile:
  # Open File
  open:
    li   $v0, 13       # Open File Syscall
    la   $a0, file     # Load File Name
    li   $a1, 0        # Read-only Flag
    li   $a2, 0        # (ignored)
    syscall
    move $t6, $v0      # Save File Descriptor
    blt  $v0, 0, err   # Goto Error

  # Read Data
  read:
    li   $v0, 14       # Read File Syscall
    move $a0, $t6      # Load File Descriptor
    la   $a1, buffer   # Load Buffer Address
    li   $a2, 156      # Buffer Size
    syscall

  # Print Data
  print:
    li   $v0, 4        # Print String Syscall
    la   $a0, cont     # Load Contents String
    syscall

  # Close File
```

```
  close:
    li   $v0, 16       # Close File Syscall
    move $a0, $t6      # Load File Descriptor
    syscall
    j    done          # Goto End

  # Error
  err:
    li   $v0, 4        # Print String Syscall
    la   $a0, fnf      # Load Error String
    syscall
    li   $v0, 10
    syscall

  # Done
  done:
    jr   $ra
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
stampaTraccia:
    move $t8, $a0                  # salvo a0 per non perdelo
    move $t7, $a1                  # t7 = indirizzo attuale puntatore buffer
    li $t1, 0                    # t1 = contatore parentesi aperte
    li $t2, 0                    # t2 = contatore parentesi chiuse

    contaParentesiAperte:
      addi $t7, 5                  # salta di 5 posizioni
      lb $t3, 0($t7)                # t3 -> char estratto
      li $t4,'('                  # t4 -> Carattere di confronto
      beq $t3, $t4, contatoreAperte      # se il carattere estratto è una parentesi vado
ad incrementare il contatore

      li $t4,'a'
      beq $t3, $t4, saltaSottrazione

      li $t4,'t'
      beq $t3, $t4, saltaProdotto

      li $t4,'i'
      beq $t3, $t4, saltaDivisione

      contatoreAperte:
        addi $t1, 1              # incremento il contatore delle parentesi aperte
        addi $t7, 1              # vado al carattere successivo
        j contaParentesiChiuse        # vado a contare le parentesi chiuse

      saltaSottrazione:
        addi $t1, 1              # incremento il contatore delle parentesi aperte
        addi $t7, 7              # vado al carattere successivo
        j contaParentesiChiuse        # vado a contare le parentesi chiuse
```

```
saltaProdotto:
   addi $t1, 1              # incremento il contatore delle parentesi aperte
   addi $t7, 4              # vado al carattere successivo
   j contaParentesiChiuse       # vado a contare le parentesi chiuse

saltaDivisione:
   addi $t1, 1              # incremento il contatore delle parentesi aperte
   addi $t7, 5              # vado al carattere successivo
   j contaParentesiChiuse       # vado a contare le parentesi chiuse


contaParentesiChiuse:
   lb $t3, 0($t7)             # t3 -> char estratto
   li $t4, ')'              # t4 -> confronto

      beq $t3, $t4, contatoreChiuse     # se ho trovato una parentesi chiusa vado a
contarla
      li $t4, 'a'               # confronto
      slt $t5, $t3, $t4           # se t3 < t4 il char è un numero (o una virgola), altrimenti
è una lettera
      bnez $t5, jumpNum           # se t5 = 1 allora t3 non è una lettera
      j contaParentesiAperte        # se è una lettera è l'inizio di una funzione, quindi
troverò prima una parentesi aperta

      contatoreChiuse:
         addi $t2, 1            # countChiuse++
         addi $t7, 1            # scorro il ptr
         beq $t1, $t2, printSubstring # se il numero di parentesi aperte è uguale al
numero di quelle chiuse allora ho individuato la sottostringa
         j contaParentesiChiuse     # altrimenti continuo la conta

      jumpNum:
         addi $t7, 1
         j contaParentesiChiuse



      printSubstring:
         li $t1, 0
         lb $t1, 0($t7)          # carico i char da salvare in t1

         sb $zero, 0($t7)         # metto in fine stringa
         li $v0, 4
         la $a0, freccia\n         # stampo la stringa "-->"
         syscall

         li $v0, 4
         move $a0, $t8           # posizione stringa
         syscall              # stampo

         sb $t1, 0($t7)          # ripristino ciò che avevo estratto
```

```
            li $v0, 4
            la $a0, aCapo            # vado a capo
            syscall
            move $a0, $t8            # riprendo il puntatore al buffer originale

    jr $ra                      # torno al chiamante (exeption handler)
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
stampaTracciaRitorno:

    move $t2, $a0               # salvo il puntatore al buffer per non perderlo
    move $t3, $v0               # salvo v0 per non perderlo
    move $t8, $a1               # metto in t8 l'id della funzione
    move $t9, $s0               # in t9 ho il valore di ritorno

    li $v0, 4
    la $a0, frecciaOut          # stampo la stringa "<--"
    syscall

    li $t1, 'm'
    beq $t8, $t1, caricaSomma      # somma

    li $t1, 't'
    beq $t8, $t1, caricaSottrazione # sottrazione

    li $t1, 'o'
    beq $t8, $t1, caricaProdotto    # prodotto

    li $t1, 'v'
    beq $t8, $t1, caricaDivisione   # divisione


    caricaSomma:
      la $a0, strSomma          # carico la posizione della stringa
      j stampaRitorno

    caricaSottrazione:
      la $a0, strSottr          # carico la posizione della stringa
      j stampaRitorno

    caricaProdotto:
      la $a0, strProd           # carico la posizione della stringa
      j stampaRitorno

    caricaDivisione:
      la $a0, strDiv            # carico la posizione della stringa
      j stampaRitorno


    stampaRitorno:
      li $v0, 4                 # codice per stampare una stringa
```

```
        syscall             # stampo

        la $a0, strReturn       # carico la posizione dell'altro pezzo di stringa da stampare
        li $v0, 4           # codice per stampare una stringa
        syscall             # stampo

        move $a0, $t9           # carico il return value
        li $v0,1            # codice per stampare un intero
        syscall             # stampo

        la $a0, strClose        # carico la posizione dell'altro pezzo di stringa da stampare
        li $v0, 4           # codice per stampare una stringa
        syscall             # stampo

    move $a0, $t2
    move $v0, $t3

    jr $ra                  # torno al chiamante (exeption handler)
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

SCHEDULER DI PROCESSI

Autori:
Gabriele Bertini – gabriele.bertini3@stud.unifi.it
Lorenzo Pratesi Mariti – lorenzo.pratesi@stud.unifi.it

Data di consegna:
29 maggio 2016

Introduzione:

La funzionalità principale di questo programma è quella di simulare uno sheduler di processi, tramite un menù a scelta, l'utente decide quale operazione desidera eseguire. L'utente ha la possibilità di scegliere tra 6 operazioni fondamentali che permettono di manipolare i task inseriti.
I task verranno memorizzati in una lista alla quale è possibile accedere tramite l'indirizzo della testa, ed ogni task verrà collegato in posizione adeguata tramite un puntatore.

Il programma termina quando viene scelto l'apposito comando dal menù, chiamando una funzione che tramite la chiamata di sistema n.10 "exit" chiude tutti i processi e torna al simulatore QtSpim.

Implementazione dei task:
I "task" memorizzati all'interno della lista hanno questa struttura:

- ID → 4 byte
- Priorità → 4 byte
- Nome → 16 byte
- Numero Esecuzioni → 4 byte
- Ptr_Successivo → 4 byte

Per problemi di allineamento è stato scelto di preservare 4 byte per il campo ID, Priorità e Numero Esecuzioni. Inoltre allocando 4 byte per ogni campo si ha una migliore visibilità del record in memoria dinamica.
Il Nome è di 16 byte: 8 byte per il buffer , 1 byte poiché viene considerato anche il carattere di fine stringa ad infine per problemi di allineamento della memoria abbiamo aggiunto altri 7 byte.
Ovviamente sono stati preservati 4 byte di memoria per il Ptr_Successivo in quanto questo elemento è una word di 4 byte e quindi l'accesso a tale oggetto è garantito senza disallineamenti della memoria.
Memoria totale occupata: 28 byte per task.

Descrizione delle procedure:

- Main

Parametri in ingresso: nessuno
Descrizione:

- inizializza il contatore id, la testa e la coda della lista a zero
- imposta la politica di sheduling a "Scheduling per Priorità"
- stampa il menù
- legge il carattere da input e controlla che sia correttamente ammissibile

- tramite l'utilizzo della jump address table scelgo l'operazione da effettuare e la/le procedure da chiamare
- etichette per lo switch:
  1. inserimento nuovo task
  2. esecuzione task in testa,
  3. esecuzione task a scelta
  4. eliminazione task a scelta
  5. modifica priorità task a scelta
  6. modifica politica scheduling
  7. esci
- una volta terminata la chiamata a procedura adeguata torna a stampare il menu fintanto che non si esce dal programma

Valori di ritorno: nessuno

DESCRIZIONE DELLE CASE ACTIONS CON RELATIVE CHIAMATE A PROCEDURA

1. *Inserimento nuovo task:*
   - prende da input i valori di Priorità, nome e numero esecuzioni (per convenzione si suppone che nel campo priorità sia inserito un numero, altrimenti la priorità sarà 0).
   - controlla che si sia inserito i valori corretti: 0-9 per la priorità e 0-99 per il numero di esecuzioni
   - chiama la procedura *creazione*
   - controlla quale politica scheduling è attualmente attuata
   - chiama la procedura *inserzionePriorita* oppure *inserzioneNumEsecuzioni* in base alla politica scelta
   - aumenta l'id di 1 intero
   - chiama la procedura *stampaTask*
   - torna al menu

- creazione

Parametri in ingresso: nessuno
Descrizione:
- crea una nuova allocazione di memoria tramite la chiamata di sistema sbrk
- salva in memoria l'ID all'indirizzo 0(ptr nuovo task)
- salva in memoria la Priorità all'indirizzo 4(ptr nuovo task)
- salva in memoria ogni singolo carattere del campo nome (tranne '\n') all'indirizzo 8(ptr nuovo task)
- salva in memoria il Numero Esecuzioni all'indirizzo 20(ptr nuovo task)
- salva in memoria il ptr al task successivo inizialmente zero, all'indirizzo 24(ptr nuovo task)
- torna al chiamante

Valori di ritorno: nessuno

- inserzionePriorita

Parametri in ingresso: indirizzo al task creato, indirizzo della testa
Descrizione:
- controlla se la lista è vuota (indirizzo testa == null), in tal caso inserisce l'unico task creato, assegna sia la testa sia la coda a tale task e torna al chiamante; altrimenti, salta all'inserimento ordinato

- inizializzo un puntatore temporaneo alla testa, utilizzato per scorrere la lista
- inizio il ciclo di confronti per priorità. Se le priorità dei task confrontati sono uguali, gestisco l'inserimento per ID, sennò inserisco il task nella posizione adeguata
- aggiorno testa/coda nel caso che il nuovo task vada agli estremi della lista
- torno al chiamante

Valori di ritorno: nessuno

- inserzioneNumEsecuzioni

Parametri in ingresso: indirizzo al task creato, indirizzo della testa
Descrizione:

- controlla se la lista è vuota (indirizzo testa == null), in tal caso inserisce l'unico task creato, assegna sia la testa sia la coda a tale task e torna al chiamante; altrimenti, salta all'inserimento ordinato
- inizializzo un puntatore temporaneo alla testa, utilizzato per scorrere la lista
- inizio il ciclo di confronti per numero di esecuzioni. Se i numeri d'esecuzioni dei due task confrontati sono uguali, gestisco l'inserimento per ID, sennò inserisco il task nella posizione adeguata
- aggiorno testa e coda nel caso che il nuovo task vada agli estremi della lista
- torno al chiamante

Valori di ritorno: nessuno

2. *Esecuzione del task in testa*
   - chiamata alla procedura *eseguiTaskInTestaAllaCoda*
   - torna al menù

- eseguiTaskInTestaAllaCoda

Parametri in ingresso: nessuno
Descrizione:

- Push nello stack, alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp)
- scorre la lista fino a che non arriva al task in coda, nel caso la lista sia vuota chiama la procedura *stampaMessaggioListaVuota*, nel caso la ricerca dell'id specifico fallisca stampa il messaggio adeguato
- arrivato in coda esegue il task (decrementa numero esecuzioni di 1)
- controlla se numero esecuzioni è 0, in tal caso elimina il task facendo puntare il ptr_successivo del task precedente a zero
- controlla in che politica siamo; se siamo in scheduling per numero di esecuzioni, stacca il task e chiama la procedura *inserzioneNumEsecuzioni* per inserire il task aggiornato in maniera ordinata; altrimenti mantiene l'ordinamento per politica
- chiama la procedura *stampaTask*
- riprende l'indirizzo di ritorno del chiamante, dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: nessuno

3. *Esecuzione task specifico*
   - prende l'intero da input (ID)
   - chiamata alla procedura *eseguiTaskConID*
   - torna al menù

- eseguiTaskConID

Parametri in ingresso: ID da eseguire
Descrizione:

- Push nello stack, alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp)
- scorre la lista fino a che non arriva al task in coda, nel caso la lista sia vuota chiama la procedura *stampaMessaggioListaVuota*, nel caso la ricerca dell'id specifico fallisca stampa il messaggio adeguato
- arrivato al task specificato lo esegue (decrementa numero esecuzioni di 1)
- controlla se numero esecuzioni è 0, in tal caso elimina il task facendo puntare il ptr_successivo del task precedente a zero
- se il task specificato si trova in testa/coda, aggiorna la nuova testa oppure la nuova coda
- controlla in che politica siamo; se siamo in scheduling per numero di esecuzioni, stacca il task e chiama la procedura *inserzioneNumEsecuzioni* per inserire il task aggiornato in maniera ordinata; altrimenti mantiene l'ordinamento per politica
- chiama la procedura *stampaTask*
- riprende l'indirizzo di ritorno del chiamante, dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: nessuno

4. *Elimina task specifico*
   - prende l'intero da input (ID)
   - chiamata alla procedura *eliminaTaskConID*
   - chiamata alla procedura *stampaTask*
   - torna al menù

- eliminaTaskConID

Parametri in ingresso: ID da eseguire
Descrizione:

- Push nello stack, alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp)
- scorre la lista fino a che non arriva al task specificato, nel caso la lista sia vuota chiama la procedura *stampaMessaggioListaVuota*, nel caso la ricerca dell'id specifico fallisca stampa il messaggio adeguato
- arrivato al task specificato lo elimina, tiene conto con relativo aggiornamento se il task che vogliamo eliminare è la testa/coda
- riprende l'indirizzo di ritorno del chiamante, dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: nessuno

5. *Modifica priorità task specifico*
   - prende l'intero da input (PRIORITÀ)
   - chiamata alla procedura *modificaPrioritaTask*
   - chiamata alla procedura *stampaTask*
   - torna al menù

- modificaPrioritaTask

Parametri in ingresso: ID da modificare
Descrizione:
- Push nello stack, alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp)
- scorre la lista fino a che non arriva al task specificato, nel caso la lista sia vuota chiama la procedura *stampaMessaggioListaVuota*, nel caso la ricerca dell'id specifico fallisca stampa il messaggio adeguato
- arrivato al task specificato controlla in che politica siamo se siamo in scheduling per numero di esecuzioni non necessita di reinserire ordinatamente il task
- se il task che vogliamo modificare si trova in testa/coda, aggiorna adeguatamente la testa/coda
- stacca il task aggiornato dalla lista
- chiama la procedura *inserzionePriorita*
- riprende l'indirizzo di ritorno del chiamante, dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: nessuno

6. *Cambia la politica di scheduling*
   - controlla la politica attuale
   - inverte la politica
   - chiamata alla procedura *ordinamentoPerPolitica*
   - chiamata alla procedura *stampaTask*
   - torna al menù

- ordinamentoPerPolitica

Parametri in ingresso: ID da modificare
Descrizione:
- Push nello stack, alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp)
- scorre la lista, conta quanti task sono presenti nella lista, nel caso la lista sia vuota chiama la procedura *stampaMessaggioListaVuota*
- controlla la politica attuale
- fa un ciclo di estrazioni in testa
- aggiorna la nuova testa
- inserisce il task estratto in maniera ordinata nella nuova lista tramite le procedure *inserzionePriorita*, *inserzioneNumEsecuzioni* in base alla politica attuale
- riprende l'indirizzo di ritorno del chiamante, dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: nessuno

7. *Esci dal programma*
   - stampa messaggio d'uscita
   - esce dal programma (syscall n. 10)

DESCRIZIONE DELLE PROCEDURE DI STAMPA

- stampaTask

Parametri in ingresso: indirizzo della testa
Descrizione:
- Push nello stack, alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp)
- controlla se la lista è vuota (indirizzo testa == null), in tal caso torna al chiamante; altrimenti stampa la stringa "interlinea" e i "campi"
- inizializzo un puntatore temporaneo alla testa, utilizzato per scorrere la lista
- inizio del ciclo di stampa, chiama la procedura *stampaSingolo* fintanto che il puntatore temporaneo non arriva alla fine della lista
- riprende l'indirizzo di ritorno del chiamante, dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: nessuno

- stampaSingolo

Parametri in ingresso: indirizzo del task puntato
Descrizione:
- Push nello stack, alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp)
- stampa il task secondo questo schema:
- chiama le procedure in quest'ordine *stampaBarra*, *stampaSpazioID*, *stampaSpazio*, *stampaBarra*, *stampaSpazioPriorita*, *stampaSpazioPriorita*, *stampaBarra*, *stampaSpazioNome*, *stampaBarra*, *stampaSpazioEsecPre*, *stampaSpazioEsecPost*, *stampaBarra*, *vaiACapo*, *stampaInterlinea*
- per ogni campo stampa il valore nella posizione corretta durante le varie chiamate a procedura del punto precedente
- riprende l'indirizzo di ritorno del chiamante, dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: nessuno

- stampaSpazioID

Parametri in ingresso: puntatore al campo ID
Descrizione:
- Push nello stack, alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp)
- controlla le cifre del numero
- stampa gli spazi a sinistra del numero con la procedura *stampaSpazio* per una visualizzazione corretta
- riprende l'indirizzo di ritorno del chiamante, dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: nessuno

- **stampaSpazioPriorità**

Parametri in ingresso: nessuno
Descrizione:
- stampa pa stringa *stampaSpaziP* che contiene 6 spazi
- torna al chiamante

Valori di ritorno: nessuno

- **stampaSpazioNome**

Parametri in ingresso: puntatore al campo Nome
Descrizione:
- Push nello stack, alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp)
- conta quanti caratteri sono stati memorizzati
- sottrae 11 (numero di spazi nel caso il nome sia di lunghezza 0) al numero di caratteri memorizzati
- chiama la procedura *stampaSpazio* per il numero di volte calcolato al punto precedente
- riprende l'indirizzo di ritorno del chiamante, dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: nessuno

- **stampaSpazioEsecPre**

Parametri in ingresso: puntatore al campo Numero Esecuzioni
Descrizione:
- Push nello stack, alloca 8 byte per salvare l'indirizzo di ritorno del chiamante (sp)
- controlla le cifre del numero
- se il numero ha una sola cifra stampa la stringa di 7 spazi s*pazioEsecPre*
- se il numero ha 2 cifre stampa la stringa di 6 spazi spazioEsecPost
- riprende l'indirizzo di ritorno del chiamante, dealloca 8 byte utilizzati in precedenza.

Valori di ritorno: nessuno

- **stampaSpazioEsecPost**

Parametri in ingresso: nessuno
Descrizione:
- stampa la stringa di 6 spazi spazioEsecPost
- torna al chiamante

Valori di ritorno: nessuno

Le altre procedure di stampa quali: *stampaBarra*, *stampaSpazio*, *vaiACapo* e *stampaInterlinea* si limitano a:
- stampare la stringa allocata in *.data e tornare al chiamante*
  - *stampaBarra* → "|"
  - *stampaSpazio* → " "
  - *vaiACapo* → "\n"
  - *stampaInterlinea* → "+----+----------+----------+----------------+"

Simulazione:

La simulazione mostra l'evoluzione della *memoria dinamica* attraverso la creazione di 3 task e l'applicazione di tutte le operazioni effettuabili dal menù.

- Mostriamo l'evoluzione della struttura *Coda* durante l'inserimento dei 3 task, osservando come variano i puntatori a testa (contenuto nel registro t8) e coda (contenuto nel registro t9) .

```
+----+----------+----------+------------------+
| ID | PRIORITA' | NOME TASK | ESECUZ. RIMANENTI|
+----+----------+----------+------------------+
|  0 |     6     |first     |       54         |
+----+----------+----------+------------------+
```

```
R24 [t8] = 10040000
R25 [t9] = 10040000
```

```
[10010400]..[10040003]  00000000
[10040004]    00000006  73726966  00000074              . . . . f i r s t . . .
[10040010]    00000000  00000036  00000000                . . . . 6 . . . . . . . .
```

Inizialmente la testa e la coda coincidono perché abbiamo un solo task (ovvero un solo record).

Inseriamo il secondo task secondo la politica di default (cioè per priorità) e notiamo come il nuovo task si posiziona in testa aggiornando il puntatore alla testa che punta al campo ID del task *second*.

```
+----+----------+----------+------------------+
| ID | PRIORITA' | NOME TASK | ESECUZ. RIMANENTI|
+----+----------+----------+------------------+
|  1 |     8     |second    |       87         |
+----+----------+----------+------------------+
|  0 |     6     |first     |       54         |
+----+----------+----------+------------------+
```

```
R24 [t8] = 1004001c
R25 [t9] = 10040000
```

```
[10040004]    00000006  73726966  00000074              . . . . f i r s t . . .
[10040010]    00000000  00000036  00000000  00000001    . . . . 6 . . . . . . . . . . .
[10040020]    00000008  6f636573  0000646e  00000000    . . . . s e c o n d . . . . . .
[10040030]    00000057  10040000                        W . . . . . . .
```

Inserendo il terzo task in posizione intermedia non si ha una modifica dei puntatori.
Come vediamo, all'indirizzo 10040034 (campo Ptr_Successivo del task *second*) è memorizzato l'indirizzo al record successivo, ovvero il task *third*, che a sua volta punta al task *first*, indirizzo 10040000.

```
+----+----------+----------+------------------+
| ID | PRIORITA' | NOME TASK | ESECUZ. RIMANENTI|
+----+----------+----------+------------------+
|  1 |     8     |second    |       87         |
+----+----------+----------+------------------+
|  2 |     7     |third     |       91         |
+----+----------+----------+------------------+
|  0 |     6     |first     |       54         |
+----+----------+----------+------------------+
```

```
[10010400]..[10040003]  00000000
[10040004]    00000006  73726966  00000074              . . . . f i r s t . . .
[10040010]    00000000  00000036  00000000  00000001    . . . . 6 . . . . . . . . . . .
[10040020]    00000008  6f636573  0000646e  00000000    . . . . s e c o n d . . . . . .
[10040030]    00000057  10040038  00000002  00000007    W . . . 8 . . . . . . . . . . .
[10040040]    72696874  00000064  00000000  0000005b    t h i r d . . . . . . . [ . . .
[10040050]    10040000                                    . . . .
```

Successivamente mostriamo l'esecuzione del task in testa alla coda che, come specificato nelle istruzioni del progetto, è l'ultimo della lista stampata, cioè la nostra coda.
Notiamo come il task *first* ha decrementato il numero di esecuzioni rimanenti, visibile all'indirizzo di memoria 10040014.

ID	PRIORITA'	NOME TASK	ESECUZ. RIMANENTI
1	8	second	87
2	7	third	91
0	6	first	53

```
[10010400]..[10040003]  00000000
[10040004]    00000006  73726966  00000074              . . . . f i r s t . . .
[10040010]    00000000  00000035  00000000  00000001    . . . . 5 . . . . . . . . . . .
[10040020]    00000008  6f636573  0000646e  00000000    . . . . s e c o n d . . . . . .
[10040030]    00000057  10040038  00000002  00000007    W . . . 8 . . . . . . . . . . .
[10040040]    72696874  00000064  00000000  0000005b    t h i r d . . . . . . . [ . . .
[10040050]    10040000                                    . . . .
```

Eseguiamo il task con ID 2 (*third*) che ha decrementato il numero di esecuzioni rimanenti, visibile all'indirizzo di memoria 1004004c.

ID	PRIORITA'	NOME TASK	ESECUZ. RIMANENTI
1	8	second	87
2	7	third	90
0	6	first	53

```
[10010400]..[10040003]  00000000
[10040004]    00000006  73726966  00000074              . . . . f i r s t . . .
[10040010]    00000000  00000035  00000000  00000001    . . . . 5 . . . . . . . . . . .
[10040020]    00000008  6f636573  0000646e  00000000    . . . . s e c o n d . . . . . .
[10040030]    00000057  10040038  00000002  00000007    W . . . 8 . . . . . . . . . . .
[10040040]    72696874  00000064  00000000  0000005a    t h i r d . . . . . . . Z . . .
[10040050]    10040000                                    . . . .
```

Eseguendo l'opzione di modifica priorità del task con ID 2 (*third*) possiamo osservare come il programma riordini i task secondo il campo priorità, modificando il puntatore alla coda.

R24 [t8] = 1004001c
R25 [t9] = 10040038

ID	PRIORITA'	NOME TASK	ESECUZ. RIMANENTI
1	8	second	87
0	6	first	53
2	3	third	90

```
[10010400]..[10040003]  00000000
[10040004]    00000006  73726966  00000074              . . . . f i r s t . . .
[10040010]    00000000  00000035  10040038  00000001    . . . . 5 . . . 8 . . . . . . .
[10040020]    00000008  6f636573  0000646e  00000000    . . . . s e c o n d . . . . . .
[10040030]    00000057  10040000  00000002  00000003    W . . . . . . . . . . . . . . .
[10040040]    72696874  00000064  00000000  0000005a    t h i r d . . . . . . . Z . . .
[10040050]    00000000                                    . . . .
```

Mostriamo il cambio di politica scheduling, che riordina secondo il numero di esecuzioni rimanenti con conseguente modifica dei puntatori. Si noti come il task *first* adesso punti al task *second* che a sua volta punta al task *third*.

```
Politica per numero di esecuzioni
+----+-----------+-----------+------------------+
| ID | PRIORITA' | NOME TASK | ESECUZ. RIMANENTI|
+----+-----------+-----------+------------------+
|  0 |     6     |first      |        53        |
+----+-----------+-----------+------------------+
|  1 |     8     |second     |        87        |
+----+-----------+-----------+------------------+
|  2 |     3     |third      |        90        |
+----+-----------+-----------+------------------+
```

```
R24 [t8] = 10040000
R25 [t9] = 10040038

[10010400]..[10040003]  00000000
[10040004]    00000006  73726966  00000074              . . . . f i r s t . . .
[10040010]    00000000  00000035  1004001c  00000001    . . . . 5 . . . . . . . . . .
[10040020]    00000008  6f636573  0000646e  00000000    . . . . s e c o n d . . . . . .
[10040030]    00000057  10040038  00000002  00000003    W . . . 8 . . . . . . . . . .
[10040040]    72696874  00000064  00000000  0000005a    t h i r d . . . . . . . Z . . .
[10040050]    00000000                                    . . . .
```

Per mostrare come si comportano due task con stesso numero di esecuzioni rimanenti, abbiamo eseguito 3 volte il task in testa (*third*) così che eguagliasse il numero di esecuzioni rimanenti del task precedente (*second*). Notiamo l'aggiornamento dei puntatori al record successivo di ogni task e come il task con ID 2 si è posizionato come secondo nella lista stampata.

```
+----+-----------+-----------+------------------+
| ID | PRIORITA' | NOME TASK | ESECUZ. RIMANENTI|
+----+-----------+-----------+------------------+
|  0 |     6     |first      |        53        |
+----+-----------+-----------+------------------+
|  2 |     3     |third      |        87        |
+----+-----------+-----------+------------------+
|  1 |     8     |second     |        87        |
+----+-----------+-----------+------------------+
```

```
R24 [t8] = 10040000
R25 [t9] = 1004001c

[10010400]..[10040003]  00000000
[10040004]    00000006  73726966  00000074              . . . . f i r s t . . .
[10040010]    00000000  00000035  10040038  00000001    . . . . 5 . . . 8 . . . . . .
[10040020]    00000008  6f636573  0000646e  00000000    . . . . s e c o n d . . . . . .
[10040030]    00000057  00000000  00000002  00000003    W . . . . . . . . . . . . . . .
[10040040]    72696874  00000064  00000000  00000057    t h i r d . . . . . . . W . . .
[10040050]    1004001c                                    . . . .
```

Infine si mostra l'eliminazione del task con ID 1 e si noti come il puntatore al record successivo del task eliminato sia uguale a 0, mentre la coda sia stata aggiornata con il puntatore al task con ID 2.

```
+----+-----------+-----------+------------------+
| ID | PRIORITA' | NOME TASK | ESECUZ. RIMANENTI|
+----+-----------+-----------+------------------+
|  0 |     6     |first      |        53        |
+----+-----------+-----------+------------------+
|  2 |     3     |third      |        87        |
+----+-----------+-----------+------------------+
```

```
R24 [t8] = 10040000
R25 [t9] = 10040038

[10010400]..[10040003]  00000000
[10040004]    00000006  73726966  00000074              . . . . f i r s t . . .
[10040010]    00000000  00000035  10040038  00000001    . . . . 5 . . . 8 . . . . . .
[10040020]    00000008  6f636573  0000646e  00000000    . . . . s e c o n d . . . . . .
[10040030]    00000057  00000000  00000002  00000003    W . . . . . . . . . . . . . . .
[10040040]    72696874  00000064  00000000  00000057    t h i r d . . . . . . . W . . .
[10040050]    00000000                                    . . . .
```

Come ultima scelta del menù si ha l'uscita dal programma che concluderà il tutto con la stampa del messaggio "USCITA" e una syscall 10.

```
7. Esci

USCITA
```

Codice:

```
#SCHEDULER DI PROCESSI

# Autori:
# Gabriele Bertini – gabriele.bertini3@stud.unifi.it
# Lorenzo Pratesi Mariti – lorenzo.pratesi@stud.unifi.it
#
# Data di consegna:
# 29 maggio 2016

.data
#STRINGHE PER IL MENU CON VARIE OPZIONI INTERNE:
   intestMenu:            .asciiz "\n================== Menu ==================\n"
   opzione1:              .asciiz "1. Inserire nuovo Task\n"
         inserisciPriorita:          .asciiz "Inserisci un numero da 0 a 9 per Priorita' Task: "
         inserisciNome:              .asciiz "Inserisci al massimo 8 caratteri per Nome Task: "
         inserisciNumEsecuzioni:     .asciiz "Inserisci un numero da 1 a 99 per Numero di
Esecuzioni Rimanenti: "
   opzione2:              .asciiz "2. Eseguire Task in testa\n"
   opzione3:              .asciiz "3. Eseguire Task con ID #\n"
      inserisciIDDaEseguire:       .asciiz "Inserisci ID da eseguire: "
   opzione4:              .asciiz "4. Eliminare Task con ID #\n"
      inserisciIDDaEliminare:      .asciiz "Inserisci ID da eliminare : "
   opzione5:              .asciiz "5. Modificare priorita' Task\n"
      inserisciIDDaModificare:     .asciiz "Inserisci ID da modificare: "
      inserisciNuovaPriorita:      .asciiz "Inserisci priorita: "
   opzione6:              .asciiz "6. Cambiare politica scheduling\n"
      politicaPerPriorita:         .asciiz "\nPolitica per priorità"
      politicaPerNumEsec:          .asciiz "\nPolitica per numero di esecuzioni"
   opzione7:              .asciiz "7. Esci\n\n"

#STRINGHE PER CONTROLLO ERRORI INSERIMENTO
   inserimento:           .asciiz "Inserisci un numero da 1 a 7\n\n"
   erroreMenu:            .asciiz "Il numero inserito non era compreso tra 1 e 7 \n\n"
   errorePriorita:        .asciiz "Il numero inserito non era compreso tra 0 e 9 \n\n"
   erroreEsecuzioniRimanenti: .asciiz "Il numero inserito non era compreso tra 1 e 99
\n\n"
   erroreIDNonPresente:     .asciiz "\nID non presente nella lista\n"
   codaVuota:             .asciiz "\nCODA VUOTA\n"
   numeroNonPresenteInLista:  .asciiz "\nIl numero ID inserito non è presente nella
lista.\n"

#STRINGHE VARIE PER LA STAMPA TASK
   fine:                .asciiz "USCITA \n\n"
   aCapo:                .asciiz "\n"
   campi:                .asciiz "\n| ID | PRIORITA' | NOME TASK | ESECUZ. RIMANENTI|\n"
   interLinea:           .asciiz "+----+----------+----------+-----------------+"
   barra:               .asciiz "|"
   spazio:               .asciiz " "
   spazioP:              .asciiz "    "
   spazioEsecPost:         .asciiz "       "
```

```
   spazioEsecPre:          .asciiz "      "

#MEMORIZZAZIONE DATI
   name: .space 9  # buffer di appoggio per leggere un nomeTask
   policy: .space 3    # buffer di appoggio per mantenere salvata la politica attuale
   jump_table: .word 7 # jump table array a 7 word che verra' instanziata dal main con gli indirizzi delle label che chiameranno le corrispondenti procedure

.text
.globl  main

main:
        li   $s4, 0          # registro per l'id dei task, viene inizializzato a zero ed incrementato ad ogni nuovo inserimento
        move $t8, $zero       # t8 (= Testa della lista) = 0
        move $t9, $zero       # t9 (= Coda della lista) = 0
        li   $t3, 'a'         # registro d'appoggio per inizializzare la politica: a = per priorità, b = per numero di esecuzioni
        sb   $t3, policy($zero) # buffer che contiene la politica attuale (a, b)


      # prepara la jump_table con gli indirizzi delle case actions
        la   $t1, jump_table
        la   $t0, inserimentoTask
        sw   $t0, 0($t1)
        la   $t0, esecuzioneTaskInTesta
        sw   $t0, 4($t1)
        la   $t0, esecuzioneTaskSpecifico
        sw   $t0, 8($t1)
        la   $t0, eliminazioneTaskSpecifico
        sw   $t0, 12($t1)
        la   $t0, modificaPrioritaTaskSpecifico
        sw   $t0, 16($t1)
        la   $t0, cambioPoliticaScheduling
        sw   $t0, 20($t1)
        la   $t0, esci
        sw   $t0, 24($t1)

stampaMenu: # ETICHETTA: STAMPA IL MENU
        li   $v0, 4         # $v0 = codice della print_string
        la   $a0, intestMenu # $a0 =  indirizzo della stringa
        syscall             # stampa la stringa
        li   $v0, 4         # $v0 = codice della print_string
      la   $a0, opzione1   # $a0 =  indirizzo della stringa
        syscall             # stampa la stringa
        li   $v0, 4         # $v0 = codice della print_string
      la   $a0, opzione2   # $a0 =  indirizzo della stringa
        syscall             # stampa la stringa
        li   $v0, 4         # $v0 = codice della print_string
      la   $a0, opzione3   # $a0 =  indirizzo della stringa
        syscall             # stampa la stringa
        li   $v0, 4         # $v0 = codice della print_string
```

```
    la   $a0, opzione4   # $a0 =  indirizzo della stringa
         syscall         # stampa la stringa
         li   $v0, 4     # $v0 = codice della print_string
    la   $a0, opzione5   # $a0 =  indirizzo della stringa
         syscall         # stampa la stringa
         li   $v0, 4     # $v0 = codice della print_string
    la   $a0, opzione6   # $a0 =  indirizzo della stringa
         syscall         # stampa la stringa
         li   $v0, 4     # $v0 = codice della print_string
    la   $a0, opzione7   # $a0 =  indirizzo della stringa
         syscall         # stampa la stringa

choice: # ETICHETTA: ATTENDE SCELTA DELL'UTENTE E SALTA ALLA LABEL CALCOLATA
# scelta della procedura o dell'uscita
         li   $v0, 4             # $v0 = codice della print_string
    la   $a0, inserimento      # $a0 =  indirizzo della stringa
         syscall                 # stampa la stringa

    # legge la scelta
    li   $v0, 5
         syscall
         move $t2, $v0           # $t2 = scelta 1, ..., 7

         li   $v0, 4             # $v0 = codice della print_string
    la   $a0, aCapo          # $a0 =  indirizzo della stringa
         syscall                 # stampa la stringa

         blez $t2, choice_err       # rimanda a choice_err se il numero inserito è minore di 7
         li   $t0, 7             # inizializza t0 a 7
         sle  $t0, $t2, $t0
         beq  $t0, $zero, choice_err # errore se scelta > 7

branch_case:
         addi $t2, -1      # tolgo 1 da scelta perche' prima azione nella jump table (in
posizione 0) corrisponde alla prima scelta del case
         add  $t0, $t2, $t2
         add  $t0, $t0, $t0 # ho calcolato (scelta-1) * 4
         add  $t0, $t0, $t1 # sommo all'indirizzo della prima case action l'offset calcolato
sopra
         lw   $t0, 0($t0)   # $t0 = indirizzo a cui devo saltare
         jr   $t0         # salto all'indirizzo calcolato

         choice_err:        # etichetta per errori inserimento scelta menu
    li   $v0, 4
    la   $a0, erroreMenu # carico l'indirizzo della stringa
    syscall           # stampa la stringa errore
    j    choice        # ritorna alla richiesta di inserimento di un numero tra 1 e 7


#Case 1: INSERIRE NUOVO TASK
inserimentoTask:
  inserimentoPriorita:
```

```
        la   $a0, inserisciPriorita # stampa stringa inserisci priorita
        li   $v0, 4
        syscall
        li   $v0, 5             # prende intero da input
        syscall
        move $t2, $v0           # lo salva in t2
        blt  $t2, $zero, erroreP    # errore se scelta < 0
        li   $t0, 9
        sle  $t0, $t2, $t0
        beq  $t0, $zero, erroreP    # errore se scelta > 9

    inserimentoNome:
        la   $a0, inserisciNome # stampa stringa inserisci nome
        li   $v0, 4
        syscall
        li   $v0, 8          # prende stringa da input
        la   $a0, name       # alloca all'indirizzo name
        li   $a1, 9          # 8 byte = 8 caratteri
        syscall
        move $t3, $a0 #salva la stringa in t3

    inserimentoNumeroEsecuzioni:
        la   $a0, inserisciNumEsecuzioni    #stampa stringa inserisci numero esecuzioni
        li   $v0, 4
        syscall
        li   $v0, 5                 # prende intero da input
        syscall
        move $t4, $v0               # lo salva in t4
        blez $t4, erroreE           # salta se t4 è minore o uguale a 0
        li   $t0, 99                # inizializzo li a 99 numero massimo di esecuzioni
        sle  $t0, $t4, $t0
        beq  $t0, $zero, erroreE        # errore se scelta > 99

      jal  creazione                # chiamata a procedura creazione
      move $a1, $v0                 # salvo in a1 l'indirizzo del nuovo task
      move $a2, $t8                 # salvo in a2 la testa della lista

      lb   $t6, policy($zero)       # carico in t6 la politica attuale, vado a prenderla nel
buffer policy di appoggio
      beq  $t6, 'a', inserisciModPriorita # salta a inserisciModPriorita se siamo nella
condizione "a" (per Priorità)
      beq  $t6, 'b', inserisciModNumEsec  # salta a inserisciModNumEsec se siamo nella
condizione "b" (per Numero di Esecuzioni)

      inserisciModPriorita:
        jal inserzionePriorita      # chiamata a procedura inserzionePriorita
        j   fineInserzione          # salto alla fine dell'etichetta inserimento

      inserisciModNumEsec:
        jal inserzioneNumEsecuzioni    # chiamata a procedura inserzioneNumEsecuzioni
        j   fineInserzione          # salto alla fine dell'etichetta inserimento
```

```
   fineInserzione:
        addi $s4, $s4, 1              # aumenta numero ID task
        move $a1, $t8                 # salvo la nuova testa (eventualmente modificata
dall'inserimento del task) in a1 per poi passarla alla procedura di stampa

        jal  stampaTask               # chiamata a procedura stampaTask
        j    stampaMenu               # esce, torna al menu

      erroreP:
        li $v0, 4
        la $a0, errorePriorita
        syscall                  # stampa la stringa errore
        j  inserimentoPriorita       # ritorna alla richiesta di inserimento di un numero tra 0
e 9

      erroreE:
        li $v0, 4
        la $a0, erroreEsecuzioniRimanenti
        syscall                  # stampa la stringa errore
        j  inserimentoNumeroEsecuzioni # ritorna alla richiesta di inserimento di un
numero tra 1 e 99


#Case 2: ESEGUO IL PRIMO TASK
esecuzioneTaskInTesta:
        la   $a0, opzione2           # carico l'indirizzo della stringa
        li   $v0, 4
        syscall                  # stampo la tringa "Eseguire Task in testa"

      jal  eseguiTaskInTestaAllaCoda # chiamata a procedura
        j    stampaMenu              # esce, torna al menu


#Case 3: ESEGUIRE TASK a SCELTA
esecuzioneTaskSpecifico:
        la   $a0, opzione3           # carico l'indirizzo della stringa
        li   $v0, 4
        syscall                  # stampo la tringa "Eseguire Task con ID"
        la   $a0, inserisciIDDaEseguire # carico l'indirizzo della stringa
        li   $v0, 4
        syscall                  # stampo la tringa "Inserisci ID da eseguire: "
        li   $v0, 5              # prende intero da input
        syscall

        jal  eseguiTaskConID         # chiamata a procedura eseguiTaskConID
        j    stampaMenu              # esce, torna al menu


#Case 4: ELIMINA TASK a SCELTA
eliminazioneTaskSpecifico:
        la   $a0, opzione4           # carico l'indirizzo della stringa
        li   $v0, 4
```

```
        syscall                 # stampo la tringa "Eliminare Task con ID #"
        la   $a0, inserisciIDDaEliminare # carico l'indirizzo della stringa
        li   $v0, 4
        syscall                 # stampo la tringa "Inserisci ID da eliminare: "
        li   $v0, 5             # prende intero da input
        syscall

        jal  eliminaTaskConID         # chiamata a procedura eliminaTaskConID
      move $a1, $t8             # salvo la testa per passarla alla procedura di stampa
        jal  stampaTask             # chiamata alla prcedura stampaTask
        j    stampaMenu             # esce, torna al menu


#Case 5: MODIFICA PRIORITA' TASK a SCELTA
modificaPrioritaTaskSpecifico:
        la   $a0, opzione5            # carico l'indirizzo della stringa
        li   $v0, 4
        syscall                 # stampo la tringa "Modificare priorita' Task"
        la   $a0, inserisciIDDaModificare # carico l'indirizzo della stringa
        li   $v0, 4
        syscall                 # stampo la tringa "Inserisci ID da modificare: "
        li   $v0, 5             # prende intero da input
        syscall

      move $t2, $v0             # salvo in t2 l'id da modificare

        la   $a0, inserisciPriorita     # carico l'indirizzo della stringa
        li   $v0, 4
        syscall                 # stampo la tringa "Inserisci priorità: "
        li   $v0, 5             # prende intero da input
        syscall

      jal  modificaPrioritaTask       # chiamata a procedura modificaPrioritaTask
      move $a1, $t8             # salvo la testa per passarla alla procedura di stampa
      jal  stampaTask           # chiamata alla prcedura stampaTask
        j stampaMenu                # esce, torna al menu


#Case 6: CAMBIA LA POLITICA
cambioPoliticaScheduling:
        la   $a0, opzione6            # carico l'indirizzo della stringa
        li   $v0, 4
        syscall                 # stampo la tringa "Cambiare politica scheduling"

        lb   $t3, policy($zero)       # carico in t3 la politica attuale, vado a prenderla nel
buffer policy di appoggio
        beq  $t3, 'a', cambiaB        # salta a cambiaB se t3 = a
        li   $t3, 'a'           # altrimenti significa che t3 = b, devo cambiare in a
        sb   $t3, policy($zero)       # carico nel buffer la nuova politica cambiata
        la   $a0, politicaPerPriorita   # carico l'indirizzo della stringa
        li   $v0, 4
        syscall                 # stampo la tringa "Politica per priorità"
```

```
        j   cambiaPolitica           # salto all' etichetta cambiaPolitica nella quale verra
cahimata la procedura adeguata

        cambiaB:
        li  $t3, 'b'            # se sono a questa etichetta significa che t3 = a, devo cambiare
in b
        sb  $t3, policy($zero)     # carico nel buffer la nuova politica cambiata
        la  $a0, politicaPerNumEsec # carico l'indirizzo della stringa
        li  $v0, 4
        syscall                 # stampo la tringa "Politica per numero di esecuzioni"

      cambiaPolitica:
        jal  ordinamentoPerPolitica  # chiamata a procedura ordinamentoPerPolitica
        move $a1, $t8            # salvo la testa per passarla alla procedura di stampa
        jal  stampaTask          # chiamata alla prcedura stampaTask
        j   stampaMenu           # esce, torna al menu


#Case 7: ESCE DAL PROGRAMMA
esci:
        la  $a0, opzione7 # carico l'indirizzo della stringa
        li  $v0, 4
        syscall         #stampo la tringa "Esci"

        j   exit        # esce
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
creazione:                  #PROCEDURA: crea un record e lo riempie con i dati inseriti
dall'utente
        li  $v0, 9              # chiamata sbrk: restituisce un blocco di 8 byte, puntato da v0: il
nuovo record
        li  $a0, 28             # byte allocati
        syscall

      move $t7, $v0             # salvo il ptr al nuovo record -> v0= ptr record
        addi $t7, 8
        sw  $s4, 0($v0)           # campo id
        sw  $t2, 4($v0)           # campo priorita
      move $a0, $t3             # a0 = indirizzo della stringa inserita
        mettiCarattere:
        lb  $t3, 0($a0)       # carico int t3 il carattere digitato
        beq  $t3, '\n', fineNome # se t3 = \n salta
        sb  $t3, 0($t7)       # metto in memoria il carattere
        addi $t7, 1           # avanzo alla locazione successiva
        addi $a0, 1           # avanzo al carattere successivo
        j   mettiCarattere
    fineNome:
        sb  $zero, 0($t7)         # metto in memoria carattere fine stringa
        sw  $t4, 20($v0)          # campo numEsec
        sw  $zero, 24($v0)        # campo next

      jr  $ra               # torno al chiamante (exeption handler)
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
inserzionePriorita: #PROCEDURA: inserisce task in lista in modo ordinato (per priorità)
      move $t2, $a1                 # salvo in t2 il ptr al task da inserire
      move $t8, $a2                 # salvo in t8 la testa (eventualmente modificata)
      move $t6, $t8                 # t6 verra utilizzato come testa per scorrere
         bne  $t8, $zero, link_list         # se t8!=nil (coda non vuota) vai a link_list
         move $t8, $t2                 # coda vuota, inserisco l'unico elemento, testa = t2
         move $t9, $t2                 # coda = t2
         j   esciInserzione            # esco dall'inserzione

      link_list:
         lw   $t3, 4($t6)            # t3 = priorità task puntato
         lw   $t4, 4($t2)            # t4 = priorità nuovo task
         slt  $t5, $t4, $t3          # t5 = nuovo campo priorità < attuale campo priorità
puntato

         beq  $t3, $t4, controllaID     # se le priorità sono uguali controllo l'id di ciascun
task
         beqz $t5, attaccaAllaLista     # salta se t5 == false ( 0 )
         move $t7, $t6               # t7 = t6 salvo puntatore attuale
         lw   $t3, 24($t6)           # t3 = puntatore elemento successivo
         bnez $t3, vaiAlSuccessivo      # salta se t3 != nil
         sw   $t2, 24($t9)           # il campo elemento successivo dell'ultimo del record
prende v0
         move $t9, $t2               # Coda = v0
         sw   $zero, 24($t2)           # rimetto a zero prt next
         j   esciInserzione          # fine inserizione

      vaiAlSuccessivo:
         lw   $t6, 24($t6)           # vado al sucessivo
         j   link_list              # torno a link list

      controllaID:
         lw   $t3, 0($t6)            # t3 = id task puntato
         lw   $t4, 0($t2)            # t4 = id nuovo task
         slt  $t5, $t4, $t3          # t5 = nuovo campo id < attuale campo id puntato
         beqz $t5, attaccaAllaLista     # salta se t5 == false ( 0 )
         move $t7, $t6               # t7 = t6 salvo puntatore attuale
         lw   $t3, 24($t6)           # t3 = puntatore elemento successivo
         bnez $t3, vaiAlSuccessivo      # salta se t3 != nil
         sw   $t2, 24($t9)           # il campo elemento successivo dell'ultimo del record
prende v0
         move $t9, $t2               # Coda = v0
         sw   $zero, 24($t2)           # rimetto a zero prt next
         j   esciInserzione          # fine inserizione

      attaccaAllaLista:
         bne  $t6, $t8, inserisciInLista # se il task nuovo non ha priorità massima, salta a
inserisciInLista
         sw   $t8, 24($t2)           # sennò puntatore next = punt nex del task in testa
         move $t8, $t2               # il task inserito è la nuova testa
```

```
        j   esciInserzione          # fine inserizione

    inserisciInLista:
        sw   $t6, 24($t2)          # puntatore next del nuovo task = task successivo
        sw   $t2, 24($t7)          # puntatore next del task precendente = punt next al
nuovo task inserito

  esciInserzione:
    jr   $ra                    # torno al chiamante (exeption handler)
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
inserzioneNumEsecuzioni:                #PROCEDURA: inserisce task in lista in modo
ordinato (per numero di esecuzioni)
    move $t2, $a1                  # salvo in t2 il ptr al task da inserire
    move $t8, $a2                  # salvo in t8 la testa (eventualmente modificata)
    move $t6, $t8                  # t6 verra utilizzato come testa per scorrere
        bne  $t8, $zero, link_list1         # se t8!=nil (coda non vuota) vai a link_list1
        move $t8, $t2                  # coda vuota, inserisco l'unico elemento, testa = t2
        move $t9, $t2                  # coda = t2
        j   esciInserzione1             # esco dall'inserzione

    link_list1:
        lw   $t3, 20($t6)          # t3 = numEsecuzioni task puntato
        lw   $t4, 20($t2)          # t4 = numEsecuzioni nuovo task
        slt  $t5, $t3, $t4         # t5 = nuovo campo numEsecuzioni > attuale campo
numEsecuzioni puntato

        beq  $t3, $t4, controllaID1     # se il numero di esecuzione di esntrambi i task sono
uguali controllo l'id di ciascun task
        beqz $t5, attaccaAllaLista1     # salta se t5 == false ( 0 )
        move $t7, $t6              # t7 = t6 salvo puntatore attuale
        lw   $t3, 24($t6)          # t3 = puntatore elemento successivo
        bnez $t3, vaiAlSuccessivo1      # salta se t3 != nil
        sw   $t2, 24($t9)          # il campo elemento successivo dell'ultimo del record
prende v0
        move $t9, $t2              # Coda = v0
        sw   $zero, 24($t2)          # rimetto a zero prt next
        j   esciInserzione1         # esco dall'inserzione

    controllaID1:
        lw   $t3, 0($t6)           # t3 = id task puntato
        lw   $t4, 0($t2)           # t4 = id nuovo task
        slt  $t5, $t4, $t3         # t5 = nuovo campo id < attuale campo id puntato
        beqz $t5, attaccaAllaLista1     # salta se t5 == false ( 0 )
        move $t7, $t6              # t7 = t6 salvo puntatore attuale
        lw   $t3, 24($t6)          # t3 = puntatore elemento successivo
        bnez $t3, vaiAlSuccessivo1      # salta se t3 != nil
        sw   $t2, 24($t9)          # il campo elemento successivo dell'ultimo del record
prende v0
        move $t9, $t2              # Coda = v0
        sw   $zero, 24($t2)          # rimetto a zero prt next
```

```
        j   esciInserzione1          # esco dall'inserzione

    vaiAlSuccessivo1:
        lw  $t6, 24($t6)            # vado al sucessivo
        j   link_list1              # torno a link list

    attaccaAllaLista1:
        bne  $t6, $t8, inserisciInLista1 # se il task nuovo non ha numEsecuzioni massimo,
salta a inserisciInLista
        sw   $t8, 24($t2)           # sennò puntatore next = punt nex del task in testa
        move $t8, $t2               # il task inserito è la nuova testa
        j   esciInserzione1         # esco dall'inserzione

    inserisciInLista1:
        sw   $t6, 24($t2)           # puntatore next del nuovo task = task successivo
        sw   $t2, 24($t7)           # puntatore nest del task precendente = punt next al
nuovo task inserito

  esciInserzione1:
    jr  $ra                     # torno al chiamante (exeption handler)
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
eseguiTaskInTestaAllaCoda:                  #PROCEDURA: esegue il task in coda.
    # PUSH nello Stack
    addi $sp, $sp, -4                   # alloca memoria
    sw   $ra, 0($sp)                    # salva ra nello stack

        move $t6, $t8                       # puntatore scorrimento
        move $t7, $t6                       # salvo in t7 il puntatore alla coda
    scorri1:
        lw   $t3, 24($t6)               # salvo in t3 il next attuale
        beqz $t3, decrementaNumEsec         # se next == 0 salta
        move $t7, $t6                   # aggiorno t7
        lw   $t6, 24($t6)               # vado al sucessivo
        j   scorri1                     # ciclo fino a che non sono arrivato in coda alla lista

    decrementaNumEsec:
        lw   $t3, 20($t9)               # metto in t3 il valore di num esecuzioni
        addi $t3, -1                    # faccio eseguire = decremento il numero
        beqz $t3, eliminaUltimoTask         # se t3 è arrivato a 0 sato ad elimina task
        sw   $t3, 20($t9)               # metto in memoria il nuovo num esec

        lb   $t4, policy($zero)             # carico in t4 la policy
        beq  $t4, 'a', trascuraOrdinamento  # se sono in politica Per priorità non
necessito di ordinare i task per num esecuzioni
        bne  $t6, $t8, eseguiUltimo         # altrimenti eseguo l'ultimo task
        li   $t8, 0

    eseguiUltimo:
        sw   $zero, 24($t7)             # metto a zero il next_ptr del task precedente a
quello eseguito (stacco dalla lista il task eseguito)
```

```
        move $t9, $t7                   # coda = t7
        move $a1, $t6                   # salvo in a1 il puntatore del task eseguito
        move $a2, $t8                   # salvo in a2 la testa
        jal  inserzioneNumEsecuzioni          # chiamo la procedura
inserzioneNumEsecuzioni
        move $a1, $t8                   # salvo la nuova testa (eventualmente modificata) per
passarla alla procedura stampa task

    trascuraOrdinamento:
        jal  stampaTask                 # stampo la lista
        j    esciEliminaTask            # esco

    eliminaUltimoTask:
        beq  $t8, $t9, stampaMessaggioCodaVuota # se il task che voglio eliminare è
l'unico della lista -> lista vuota
        sw   $zero, 24($t7)              # elimino il collegamento a quel task
        move $t9, $t7                   # salvo in t9 la nuova coda della lista
        jal  stampaTask                 # stampo la lista
        j    esciEliminaTask            # esco

    stampaMessaggioCodaVuota:
        jal  stampaMessaggioListaVuota        # se la lista è vuota chiamo la procedura
stampaMessaggioListaVuota

  esciEliminaTask:
    # POP nello Stack:
    lw   $ra, 0($sp)                  # riprendo ra
    addi $sp, $sp, 4                  # dealloco
    jr   $ra                        # torno al chiamante (exeption handler)
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
eseguiTaskConID:                        #PROCEDURA: esegue il task assegnato
    # PUSH nello Stack
    addi $sp, $sp, -4                 # alloca memoria
    sw   $ra, 0($sp)                  # salva ra nello stack

    move $t6, $t8                     # salvo in t6 la testa, t6 verra utilizzato come testa
per scorrere
    move $t7, $t6                     # salvo in t7 la testa, t7 mi serve come puntatore al
task precendente, puntato da t6
    scorri2:
        lw   $t3, 0($t6)                # carico l'id del task puntato
        beq  $t3, $v0, decrementaNumEsecID      # se t3 è uguale e v0 (task inserito
dall'utente) salta a decrementaNumEsecID
        move $t7, $t6                   # altrimenti salvo in t7 ciò che punta t6
        lw   $t6, 24($t6)               # avanzo t6, t6 punta al task successivo
        beqz $t6, numeroIDNonPresente         # se t6 è uguale a zero (significa che è
arrivato in fondo alla lista) salta alla stampa di errore id
        j    scorri2                  # ritorna a scorri2

    decrementaNumEsecID:
```

```
        lw   $t3, 20($t6)                  # metto in t3 il valore di num esecuzioni
        addi $t3, -1                       # faccio eseguire = decremento il numero
        beqz $t3, eliminaTask                  # se t3 è arrivato a 0 sato ad elimina task
        sw   $t3, 20($t6)                  # metto in memoria il nuovo num esec
        lb   $t4, policy($zero)              # carico in t4 la policy
        beq  $t4, 'a', trascuraOrdinamento2      # se sono in politica Per priorità non
necessito di ordinare i task per num esecuzioni
        beq  $t6, $t8, spostaLaTesta           # se voglio eseguire la testa, salto all'etichetta
indicata
        beq  $t6, $t9, spostaLaCoda            # se voglio eseguire la coda, salvo all'etichetta
indicata
        lw   $t3, 24($t6)                  # salvo in t3 il next_ptr del task puntato
        sw   $t3, 24($t7)                  # stacco il task puntato (faccio puntare t7 a ciò che
puntava t6)
        j    modificaPrioritaTaskINS           # salto a modificaPrioritaTaskINS

        spostaLaTesta:
           lw   $t8, 24($t8)               # stacco la testa (basta avanzarla)
           j    modificaPrioritaTaskINS

        spostaLaCoda:
           sw   $zero, 24($t7)              # stacco la coda (basta far puntare a zero il task
precedente)
           move $t9, $t7                   # salvo la nuova coda
           j    modificaPrioritaTaskINS

        modificaPrioritaTaskINS:
           sw   $zero, 24($t6)              # metto a zero il next_ptr del task che ho eseguito
           move $a1, $t6                   # salvo in a1 il task interessato
           move $a2, $t8                   # salvo in a2 la testa
           jal  inserzioneNumEsecuzioni       # chiamata alla procedura
inserzioneNumEsecuzioni
           move $a1, $t8                   # salvo la nuova testa (eventualmente modificata)
per passarla alla procedura stampa task

     trascuraOrdinamento2:
        jal  stampaTask                    # stampo la lista
        j    esciEliminaTaskID               # esco

     eliminaTask:
        beq  $t8, $t9, stampaMessaggioCodaVuotaID # se il task che voglio eliminare è
l'unico della lista -> lista vuota
        beq  $t6, $t8, eliminaLaTesta          # se t6 è la testa salto a eliminaLaTesta
        lw   $t3, 24($t6)                  # altrimenti, salvo in t3 il next_ptr del task puntato
        sw   $t3, 24($t7)                  # stacco il task puntato (faccio puntare t7 a ciò che
puntava t6)
        jal  stampaTask                    # stampo la lista
        j    esciEliminaTaskID               # esco

     eliminaLaTesta:
        lw   $t3, 24($t6)                  # salvo in t3 il next_ptr del task puntato
```

```
        move $t8, $t3                   # salvo la nuova testa che è il successivo next_ptr di
t6
        jal  stampaTask                 # stampo la lista
        j   esciEliminaTaskID

    numeroIDNonPresente:
        li   $v0, 4
        la   $a0, numeroNonPresenteInLista      # carico l'indirizzo della stringa
        syscall                         # stampo la stringa "Numero ID non presente"
        j   esciEliminaTaskID               # esco

    stampaMessaggioCodaVuotaID:
        jal  stampaMessaggioListaVuota          # se la lista è vuota chiamo la procedura
stampaMessaggioListaVuota

  esciEliminaTaskID:
    # POP nello Stack:
    lw   $ra, 0($sp)                    # riprendo ra
    addi $sp, $sp, 4                    # dealloco
    jr   $ra                            # torno al chiamante (exeption handler)
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
eliminaTaskConID:                       #PROCEDURA: elimina task da id dato in input
    # PUSH nello Stack
    addi $sp, $sp, -4                   # alloca memoria
    sw   $ra, 0($sp)                    # salva ra nello stack

    move $t6, $t8                       # salvo in t6 la testa, t6 verra utilizzato come testa
per scorrere
    move $t7, $t6                       # salvo in t7 la testa, t7 mi serve come puntatore al
task precendente, puntato da t6
    scorri3:
        lw   $t3, 0($t6)                # carico l'id del task puntato
        beq  $t3, $v0, eliminaTaskId            # se t3 è uguale e v0 (task inserito dall'utente)
salta a eliminaTaskConID
        move $t7, $t6                   # altrimenti salvo in t7 ciò che punta t6
        lw   $t6, 24($t6)               # avanzo t6, t6 punta al task successivo
        beqz $t6, numeroIDNonPresente2          # se t6 è uguale a zero (significa che è
arrivato in fondo alla lista) salta alla stampa di errore id
        j   scorri3                     # torno a scorri3

    eliminaTaskId:
        beq  $t8, $t9, stampaMessaggioCodaVuotaID2 # se il task che voglio eliminare è
l'unico della lista -> lista vuota
        beq  $t6, $t8, eliminaLaTestaID         # se il task che voglio eliminare è la testa
salto ad eliminaLaTestaID
        beq  $t6, $t9, eliminaLACodaID          # se il task che voglio eliminare è la coda
salto  ad eliminaLACodaID
        lw   $t3, 24($t6)               # altrimenti, salvo in t3 il next_ptr del task puntato
        sw   $t3, 24($t7)               # stacco il task puntato (faccio puntare t7 a ciò che
puntava t6)
```

```
        j   esciEliminaTaskID2              # esco

    eliminaLaTestaID:
        lw   $t3, 24($t6)                 # salvo in t3 il next_ptr del task puntato
        move $t8, $t3                     # salvo la nuova testa che è il successivo next_ptr di
t6
        j   esciEliminaTaskID2            # esco

    eliminaLACodaID:
        sw   $zero, 24($t7)               # stacco t6, metto a zero il next_ptr di t7
        move $t9, $t7                     # t7 = la nuova coda
        j   esciEliminaTaskID2            # esco

    numeroIDNonPresente2:
        li   $v0, 4
        la   $a0, numeroNonPresenteInLista      # carico l'indirizzo della stringa
        syscall                           # stampo la stringa "Numero ID non presente"
        j   esciEliminaTaskID2            # esco


    stampaMessaggioCodaVuotaID2:
        jal  stampaMessaggioListaVuota          # se la lista è vuota chiamo la procedura
stampaMessaggioListaVuota

  esciEliminaTaskID2:
    # POP nello Stack:
    lw $ra, 0($sp)                        # riprendo ra
    addi $sp, $sp, 4                      # dealloco
    jr $ra                                # torno al chiamante (exeption handler)
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
modificaPrioritaTask:                     #PROCEDURA: modifica la priorità del task indicato
da input
    # PUSH nello Stack
    addi $sp, $sp, -4                     # alloca memoria
    sw   $ra, 0($sp)                      # salva ra nello stack

    move $a3, $t8                         # salva in a3 la testa della lista
    move $t5, $a3                         # salva in t5 la testa della lista
    loop1:
        lw   $t3, 0($a3)                  # prendo id
        beq  $t3, $t2, controlloPolicy    # se id preso è uguale a quella inserita da input
salto.
        move $t5, $a3                     # altrimenti, salvo in t5 ciò che punta a3
        lw   $a3, 24($a3)                 # altrimenti scorro
        beqz $a3, numeroIDNonPresente3    # se a3 è uguale a zero (significa che è
arrivato in fondo alla lista) salta alla stampa di errore id
        j   loop1                         # ritorno a loop1

    controlloPolicy:
        lb   $t3, policy($zero)           # carico in t3 la policy
```

```
        beq  $t3, 'b', esciModificaPriorita # se siamo in modalità numero esecuzioni, cambia la priorità senza ordinare

        beq  $a3, $t8, testaAA          # salta se voglio modificare la testa
        beq  $a3, $t9, codaAA           # salta se voglio modificare la coda
        lw   $t3, 24($a3)              # carico in t3 il puntatore dell'id che voglio modificare
        sw   $t3, 24($t5)              # lo stacco dalla lista, t5 = puntatore precedente
        sw   $v0, 4($a3)               # metto la priorità desiderata nel task staccato
        move $a1, $a3                   # salvo il task staccato in a1
        move $a2, $t8                   # salvo la testa in a2
        j    chiamaINS                 # vado ad inserire
        testaAA:
           move $a1, $a3,               # salvo il task che staccherò in a1
           sw   $v0, 4($t8)            # metto la priorità desiderata nel task staccato
           lw   $t8, 24($t8)           # avanzo la testa
           move $a2, $t8                # salvo la testa in a2
           j    chiamaINS              # vado ad inserire
        codaAA:
           sw   $v0, 4($a3)            # metto la priorità desiderata nel task staccato
           move $a1, $a3                # salvo il task che staccherò in a1
           sw   $zero, 24($t5)           # stacco la coda mettensdo a zero il ptr precedente
           move $a2, $t8                # salvo la testa in a2
           move $t9, $t5                # salvo la nuova coda
           j    chiamaINS              # vado ad inserire

        chiamaINS:
           jal  inserzionePriorita      # chiamata a procedura inserzione per priorità
           j    esciFunzione            # esco

     numeroIDNonPresente3:
        jal  stampaMessaggioIDNonPresente
        j    esciFunzione

   esciModificaPriorita:
     sw   $v0, 4($a3)

   esciFunzione:
     # POP nello Stack:
     lw   $ra, 0($sp)                  # riprendo ra
     addi $sp, $sp, 4                   # dealloco
     jr   $ra                        # torno al chiamante (exeption handler)
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ordinamentoPerPolitica:                 #PROCEDURA: faccio un ciclo di estrazioni in testa dalla vecchia lista e reinserisco i record 1 a 1 con i metodi di inserzione ordinati
     # PUSH nello Stack
     addi $sp, $sp, -4                 # alloca memoria
     sw   $ra, 0($sp)                 # salva ra nello stack

     li   $s7, 0                     # registro per contare quanti task ho inserito
     move $s3, $t8                     # se = testa
```

```
    beqz $s3, stampaMessaggioCodaVuotaOP # se la lista è vuota chiamo la procedura stampaMessaggioListaVuota
  numTask:
    lw   $s3, 24($s3)              # avanza il ptr
    beqz $s3, selezione             # salta se sono arrivato alla fine
    addi $s7, 1                  # incremento il contatore
    j    numTask

  selezione:
    lb   $t3, policy($zero)          # carico in t3 la politica attuale, vado a prenderla nel buffer policy di appoggio
    beq  $t3, 'a', inserisciPerPriorita  # salta a inserisciModPriorita se siamo nella condizione "a" (per Priorità)
    beq  $t3, 'b', inserisciPerNumEsec   # salta a inserisciModNumEsec se siamo nella condizione "b" (per Numero di Esecuzioni)

  inserisciPerPriorita:
    li   $s3, 0                  # inizializzo a 0 il contatore dei cicli
    li   $s6, 0                  # inizializzo a 0 la nuova testa
    move $s2, $t8                # salvo la vecchia testa in t2
  cicloPriorita:
    move $s5, $s2                # s5 = testa
    lw   $s2, 24($s2)            # avanzo t8 = estraggo la testa
    sw   $zero, 24($s5)           # metto a 0 il next_ptr di s5
    move $a1, $s5                # salvo in a1 il task da inserire
    move $a2, $s6                # salvo in a2 la nuova testa
    jal  inserzionePriorita         # chiamata alla procedura inserzionePriorita
    move $s6, $t8                # salvo in s6 la nuova testa
    beq  $s3, $s7, fineCiclo         # salta se ho raggiunto il nmero dei task
    addi $s3, 1                  # incremento il contatotore dei cicli
    j    cicloPriorita            # ciclo finche non ho estratto tutti i task dalla vecchia lista

  inserisciPerNumEsec:
    li   $s3, 0                  # inizializzo a 0 il contatore dei cicli
    li   $s6, 0                  # inizializzo a 0 la nuova testa
    move $s2, $t8                # salvo la vecchia testa in t2
  cicloNumEsec:
    move $s5, $s2                # s5 = testa
    lw   $s2, 24($s2)            # avanzo t8 = estraggo la testa
    sw   $zero, 24($s5)           # metto a 0 il next_ptr di s5
    move $a1, $s5                # salvo in a1 il task da inserire
    move $a2, $s6                # salvo in a2 la nuova testa
    jal  inserzioneNumEsecuzioni       # chiamata alla procedura inserzioneNumEsecuzioni
    move $s6, $t8                # salvo in s6 la nuova testa
    beq  $s3, $s7, fineCiclo         # salta se ho raggiunto il nmero dei task
    addi $s3, 1                  # incremento il contatotore dei cicli
    j    cicloNumEsec             # ciclo finche non ho estratto tutti i task dalla vecchia lista


  stampaMessaggioCodaVuotaOP:
    jal  stampaMessaggioListaVuota
```

```
   fineCiclo:
      # POP nello Stack:
      lw   $ra, 0($sp)                # riprendo ra
      addi $sp, $sp, 4                 # dealloco
      jr   $ra                      # torno al chiamante (exeption handler)
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

exit: # stampa messaggio di uscita e esce
         li   $v0, 4
         la   $a0, fine
         syscall

         li   $v0, 10
      syscall

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
stampaTask:               #PROCEDURA: stampa la lista con tutti i task
         # PUSH nello Stack
      addi $sp, $sp, -4     # alloca memoria

      sw   $ra, 0($sp)      # salva ra nello stack
      beqz $t8, esciStampa  # se t8 == 0 non ci sono elementi in lista, salta alla stampa del
messaggio adeguato

      jal  vaiACapo        # chiamata alla procedura vaiACapo
         jal  stampaInterlinea # chiamata alla procedura stampaInterlinea

         li   $v0, 4
         la   $a0, campi      # stampa i campi
         syscall

         jal  stampaInterlinea # chiamata alla procedura stampaInterlinea
         jal  vaiACapo        # chiamata alla procedura vaiACapo

         move $t6, $a1        # t6 verra utilizzato come testa per scorrere
   initLoop:
      lw   $t7, 24($t6)     # t7 = valore del campo elemento-successivo dell'elemento
corrente (puntato da t6)
      beqz $t7, endLoop     # va a endLoop se si è raggiunto la fine
      jal  stampaSingolo    # chiama a procedura di stampa Singolo task
      j    initLoop        # cicla

   endLoop:
         jal  stampaSingolo    # chiama a procedura di stampa Singolo task (stampa L'ultimo
task)
   esciStampa:
      # POP nello Stack:
      lw   $ra, 0($sp)      # riprendo ra
```

```
    addi $sp, $sp, 4     # dealloco
    jr   $ra           # torno al chiamante (exeption handler)
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
stampaSingolo:               #PROCEDURA: stampa singolarmente il task
    # PUSH nello Stack
    addi $sp, $sp, -4       # alloca memoria
    sw   $ra, 0($sp)        # salva ra nello stack

    jal  stampaBarra        # stampa la barra
       li   $v0, 1           # altrimenti si stampa l'elemento corrente. Cioe':
       lw   $a0, 0($t6)        # a0 = valore del campo intero dell'elemento corrente
(puntato da t6)
       jal  stampaSpazioID      # chiamata alla procedura stampaSpazioID
       li   $v0, 1           # altrimenti si stampa l'elemento corrente. Cioe':
       lw   $a0, 0($t6)        # a0 = valore del campo intero dell'elemento corrente
(puntato da t6)
       syscall              # stampa valore intero dell'elemento corrente
    jal  stampaSpazio       # stampa uno spazio

       jal  stampaBarra        # stampa la barra
       jal  stampaSpazioPriorita # chiamata a procedura stampaSpazioPriorita
       li   $v0, 1
       lw   $a0, 4($t6)        # stampa la priorità del task
       syscall
       jal  stampaSpazioPriorita # chiamata a procedura stampaSpazioPriorita

       jal  stampaBarra        # stampa la barra
       li   $v0, 4
       addi $a0, $t6, 8        # aumenta il puntatore al campo successivo
       syscall
       jal  stampaSpazioNome    # chiamata alla procedura stampa spazio nome

       jal  stampaBarra        # stampa la barra
       jal  stampaSpazioEsecPre  # chiamata alla procedura stampaSpazioEsecPre
       li   $v0, 1
       lw   $a0, 20($t6)       # stampa il campo numero esecuzioni
       syscall
       jal  stampaSpazioEsecPost # chimata alla procedura stampaSpazioEsecPre

       lw   $t6, 24($t6)       # t0 = valore del campo elemento-successivo dell'elemento
corrente (puntato da t0)
       jal  stampaBarra        # stampa la barra
       jal  vaiACapo
       jal  stampaInterlinea
       jal  vaiACapo

    # POP nello Stack:
    lw   $ra, 0($sp)        # riprendo ra
    addi $sp, $sp, 4        # dealloco
    jr   $ra             # torno al chiamante (exeption handler)
```

```
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#VARIE PROCEDURE DI STAMPA SPAZZI PER UNA CORRETTA VISONE DELLA TABELLA
DEI TASK
vaiACapo:
        li   $v0, 4
        la   $a0, aCapo     # stampa "\n"
        syscall
        jr   $ra          # torna al chiamante

stampaInterlinea:
        li   $v0, 4
        la   $a0, interLinea # stampa interlina es: +------+------+-----
        syscall
        jr   $ra          # torna al chiamante

stampaBarra:
        li   $v0, 4
        la   $a0, barra     # stampa barra |
        syscall
        jr   $ra          # torna al chiamante

stampaSpazio:
      li   $v0, 4
      la   $a0, spazio    # stampa uno spazio " "
      syscall
      jr   $ra          # torna al chiamante

stampaSpazioPriorita:
      li   $v0, 4
      la   $a0, spazioP    # stampa 6 spazi
      syscall
      jr   $ra          # torna al chiamante
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#VARIE POCEDURE DI STAMPA ERRORI
stampaMessaggioListaVuota:
      li   $t8, 0              # azzero la testa
      li   $t9, 0              # azzero la coda
      la   $a0, codaVuota         # stampo il messaggio CODA VUOTA
      li   $v0, 4
      syscall
      jr   $ra                # torna al chiamante

stampaMessaggioIDNonPresente:
      li   $v0, 4
      la   $a0, erroreIDNonPresente # stampa messaggio "ID non presente"
      syscall
      jr   $ra                # torna al chiamante
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
stampaSpazioID:          #PROCEDURA: stampa i corretti spazi del campo id
      # PUSH nello Stack
      addi $sp, $sp, -4    # alloca memoria
      sw   $ra, 0($sp)     # salva ra nello stack

      slt  $t7, $a0, 10    # guarda se il numero è < 10
      beqz $t7, dueCifreID # salta se non lo è
      jal  stampaSpazio
         jal  stampaSpazio
         j    uscita

   dueCifreID:
      slt  $t7, $a0, 100   # guarda se il numero è < 100
      beqz $t7, uscita     # salta se non lo è
      jal  stampaSpazio

   uscita:
      # POP nello Stack:
      lw   $ra, 0($sp)     # riprendo ra
      addi $sp, $sp, 4     # dealloco
      jr   $ra          # torno al chiamante (exeption handler)
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
stampaSpazioNome:        #PROCEDURA: stampa i corretti spazi del campo nome
      # PUSH nello Stack
      addi $sp, $sp, -4  # alloca memoria
      sw   $ra, 0($sp)   # salva ra nello stack

      li   $t2, 11       # inizializzo il contatore di spazi a 11
   loop:
      lb   $t7, 0($a0)   # carica il carattere del nome task
      beqz $t7, fineLoop # se il carattere è uguale a zero salta
      addi $t2, -1       # decrementa il contatore
      addi $a0, 1        # va al prossimo carattere
      j    loop
   fineLoop:
      beqz $t2, ex       # salta quando il contatore ragginge zero
      jal  stampaSpazio
      addi $t2, -1       # decrementa il contatore
      j    fineLoop

   ex:
      # POP nello Stack:
      lw   $ra, 0($sp)   # riprendo ra
      addi $sp, $sp, 4   # dealloco
      jr   $ra         # torno al chiamante (exeption handler)
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
stampaSpazioEsecPre:          #PROCEDURA: stampa i corretti spazi prima del num esecuzione
      # PUSH nello Stack
      addi $sp, $sp, -4      # alloca memoria
      sw   $ra, 0($sp)       # salva ra nello stack

      li   $v0, 1
         lw   $a0, 20($t6)      # carica in a0 il campo num esecuzioni puntato
      slt  $t7, $a0, 10
      beqz $t7, dueCifreEsec  # salta se il num di esecuzini è > 10
      li   $v0, 4
      la   $a0, spazioEsecPre # stampo 7 spazi
      syscall
      jal  stampaSpazio
      lw   $ra, 0($sp)
      addi $sp, $sp, 4
      jr   $ra

   dueCifreEsec:
      li   $v0, 4
      la   $a0, spazioEsecPre # il numeo è a due cifre
      syscall

      # POP nello Stack:
      lw   $ra, 0($sp)       # riprendo ra
      addi $sp, $sp, 4       # dealloco
      jr   $ra             # torno al chiamante (exeption handler)

stampaSpazioEsecPost:         #PROCEDURA: stampa i corretti spazi dopo del num esecuzione
      li   $v0, 4
      la   $a0, spazioEsecPost
      syscall
      jr   $ra
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```