

Relazione sul Progetto dell'Esame di Architetture degli Elaboratori

Bindi Giovanni - 5530804 - *giovanni.bindi@stud.unifi.it*
Lippi Lorenzo - 6221250 - *lorenzo.lippi@stud.unifi.it*
Puliti Gabriele - 5300140 - *gabriele.puliti@stud.unifi.it*

7 gennaio 2017

Indice

1 Primo Esercizio	2
1.1 Descrizione generale	2
1.2 Strutture dati	2
1.3 Funzioni	3
1.3.1 Descrizione delle funzioni	3
1.4 Screenshots	5
1.4.1 Simulazioni	5
1.4.2 Gestione della Memoria	6
1.5 Codice	14
2 Secondo Esercizio	20
2.1 Schema logico	21
2.2 Strutture dati	22
2.3 Funzioni	22
2.3.1 Descrizione delle funzioni	23
2.4 Screenshots	24
2.4.1 Simulazioni	24
2.4.2 Gestione della Memoria	28
2.5 Codice	31

1 Primo Esercizio

Simulatore di chiamate a procedura:

Sia **S** una stringa che possa rappresentare una qualsiasi combinazione delle funzioni somma, sottrazione, prodotto e divisione (intera) fra due numeri interi. Esempi validi di stringa **S** sono:

- “divisione(5,2)”
- “prodotto(prodotto(3,4),2)”
- “somma(7,somma(sottrazione(0,5),prodotto(divisione(7,2),3)))”

Scrivere e provare un programma in assembly MIPS che prenda in input la stringa **S** letta da file (composta da al massimo 150 caratteri), e che:

- implementi le funzioni somma, sottrazione, prodotto e divisione (intera) fra due numeri interi, come procedure MIPS;
- simuli la sequenza delle chiamate alle funzioni nell'ordine in cui appaiono nella stringa in input (da sinistra a destra);
- visualizzi su console la traccia con la sequenza delle chiamate annidate (con argomento fra parentesi) ed i valori restituiti dalle varie chiamate annidate (valore restituito fra parentesi);

1.1 Descrizione generale

Il compito di questo programma è l'esecuzione di un'analisi lessicale per la comprensione e la gestione di combinazioni di operazioni aritmetiche binarie tra numeri interi (SOMMA, SOTTRAZIONE, PRODOTTO e DIVISIONE). Date le specifiche progettuali sovraelencate ci è sembrato più adeguato adottare una soluzione di tipo ricorsivo, che ci permettesse di elaborare i dati in ingresso a seconda delle diverse possibilità che possono occorrere : un numero, ad esempio, può essere espresso sia direttamente sia essere il risultato di una o più operazioni già effettuate. L'algoritmo, basandosi su queste considerazioni, itera la scannerizzazione della stringa **S** in ingresso fino a trovare due numeri interi, ne esegue l'operazione direttamente relativa ed utilizza il risultato ottenuto come operando in un'operazione successiva (o precedente), qualora ve ne sia una. Abbiamo quindi definito procedure legate all'elaborazione (esecuzione delle operazioni) ed alla lettura (riconoscimento di caratteri) ma cercando di mantenere una adeguata divisione dei compiti; ad esempio la funzione su cui si basa lo scheletro del parser, la procedura **scan**, riceve in input una stringa **S** ma delega le responsabilità per il calcolo dei risultati ad altre procedure. Abbiamo poi deciso di gestire la memoria necessaria nell'esecuzione del programma attraverso lo **stack**.

1.2 Strutture dati

Le strutture dati utilizzate per risolvere il primo esercizio vengono dichiarate ed inizializzate nel **.data** e sono definite nel modo seguente:

- **fNF** : stringa che contiene il messaggio di errore che viene stampato quando il file non viene trovato
- **file** : stringa contenente il percorso del file di testo nel quale è presente la stringa **S**
- **buffer** : locazione di memoria utilizzata per salvare la stringa **S**
- **newl** : stringa contenente i caratteri '\n' (*newline*)
- **rightarrow** : stringa che formatta l'inizio della traccia di output
- **retsum** : stringa che formatta sulla traccia il valore restituito da una somma
- **retsot** : stringa che formatta sulla traccia il valore restituito da una sottrazione
- **retpro** : stringa che formatta sulla traccia il valore restituito da un prodotto
- **retdiv** : stringa che formatta sulla traccia il valore restituito da una divisione

- **finres** : stringa che formatta sulla traccia il valore finale
- **par** : stringa contenente il carattere ')
- **divzero** : stringa che contiene il messaggio di errore legato alla divisione per zero
- **stack** : utilizzato per l'allocazione dell'indice di posizione sulla stringa ed il risultato dell'ultima operazione effettuata

1.3 Funzioni

- **main** : funzione di apertura del file e preparazione del file descriptor
- **read** : funzione che legge il contenuto del file e lo memorizza all'interno del **buffer**
- **start** : funzione che stampa il contenuto del file (la stringa **S**), prepara i parametri da passare alla funzione **scan**, stampa il risultato dell'espressione letta ed esce
- **scan** : funzione ricorsiva che riceve come argomento in input la stringa contenente l'operazione matematica da eseguire (letta dal file), la scansiona carattere per carattere, allocando nello stack l'indirizzo di ritorno e la posizione attuale, saltando poi alla procedura incaricata di svolgere quella determinata operazione
- **nega** : funzione che moltiplica per -1 il numero letto
- **ind** : funzione che incrementa l'indice di scansionamento della stringa
- **sum** : funzione che esegue la somma e ne restituisce il risultato
- **subt** : funzione che esegue la sottrazione e ne restituisce il risultato
- **prod** : funzione che esegue il prodotto e ne restituisce il risultato
- **divi** : funzione che esegue la divisione e ne restituisce il risultato, eseguendo i controlli sul denominatore
- **printop, iter** : funzioni che stampano la stringa contente l'operazione in esame
- **loop** : funzione che esegue la scansione vera e propria dei numeri che contengono anche piu' di una cifra
- **close** : funzione che chiude la connessione con il descrittore del file contente l'operazione
- **err** : funzione che stampa la stringa **fNF**
- **quit** : funzione che esegue la system call per l'uscita dal programma

1.3.1 Descrizione delle funzioni

La funzione **main** si occupa di eseguire la system call per aprire il file specificato dal percorso contenuto nella stringa **file** e restituire il descrittore del file relativo. La funzione **read** si occupa di eseguire la system call per leggere il contenuto del file specificato dal descrittore di file e lo salva nella locazione di memoria identificata da **buffer**. La funzione **start** memorizza nel registro \$a0 il puntatore al primo carattere della stringa **S**, stampa la stringa e passa alla funzione **scan** il puntatore stesso (eseguendo una JAL). Una volta che è stata eseguita la **scan** scrive il risultato dell'espressione ed esce dal programma. La funzione **scan** riceve come argomento in \$a0 l'indirizzo di inizio del buffer allocato in memoria, alloca 3 posizioni nello **stack** per memorizzare il puntatore alla posizione attuale, l'indirizzo di ritorno del chiamante e l'intero letto (quando riconosce un numero). Per quanto riguarda la scansione vera e propria la funzione si sposta intelligentemente nella stringa per capire quale operazione deve essere chiamata (ad esempio se nella stringa abbiamo una somma sappiamo che i primi 2 caratteri sono identici all'operazione sottrazione, la differenza è nel terzo carattere quindi controlleremo che il terzo carattere sia una m), appena l'operazione viene riconosciuta viene eseguita la procedura che esegue l'operazione. Se questa non ha successo verifica che il carattere sia uguale ad un meno e, nel caso, trasforma l'interno positivo in intero negativo. Se anche questo riconoscimento non ha successo vuol dire che ha incontrato l'interno cercato, trasforma il carattere da codice **ASCII** in costante numerica,

verifica che sia composto da una o più cifre (in questo caso entra all'interno di un loop che esegue i calcoli per la corretta rappresentazione posizionale in base 10) e la restituisce come risultato (in \$v0). Come ultima operazione dealloca le posizioni sullo *stack* allocate all'inizio e torna al chiamante.

La funzione **neg** viene richiamata quando la **scan** legge il carattere '-'. Quest'ultima quindi aumenta di 1 il valore di \$a0 e chiama **scan** per leggere il valore da negare. Il valore da negare viene restituito dallo **scan** nel registro \$v0 che viene poi moltiplicato per -1. Come ultima operazione salva il risultato nello *stack*. La funzione **ind** estrae dallo *stack* il puntatore alla posizione attuale, lo incrementa di uno e richiama la procedura **scan** per continuare la scansione.

La procedura **sum**, al fine di eseguire la somma, preleva dallo *stack* la posizione attuale sulla stringa e la incrementa di 6, puntando quindi al primo argomento dopo la parentesi e richiama **scan**, che a sua volta verifica se il primo argomento sia un numero oppure un'altra operazione. Quando **scan** restituisce un intero vuol dire che si è letto il primo argomento dell'operazione, il quale viene allocato momentaneamente nello *stack*, viene poi richiamata ancora una volta la procedura di scannerizzazione per leggere il secondo argomento dell'operazione. Una volta effettuato ciò preleva dallo *stack* il primo argomento della funzione, esegue la somma con il secondo operando appena letto, salva il risultato nel registro \$v0, dealloca lo *stack* e passa l'esecuzione al chiamante (jr \$ra). Le altre procedure si differenziano unicamente per l'operazione effettuata ad eccezione di **divi** che esegue anche un controllo sul dividendo, il quale deve essere ovviamente diverso da 0. Le procedure **printop** ed **iter** hanno il compito di stampare l'operazione fatta dal chiamante, **printop** riceve come parametro in ingresso il puntatore corrente nella stringa, alloca nello *stack* l'indirizzo di ritorno, scorre ed **iter** stampa finché non trova una parentesi chiusa. Quindi stampa l'operazione corrente con entrambi gli argomenti e **fine** ripassa il controllo al chiamante. La procedura **close** invoca la system call per chiudere la connessione con il file specificato dal descrittore passato in \$a0.

1.4 Screenshots

Semplici simulazioni che mostrano l'esecuzione del programma in corrispondenza di tre casi d'interesse, come richiesto nelle specifiche. Purtroppo non abbiamo ottenuto la corretta modalità di stampa ed abbiamo quindi aggiunto una stringa 'result' in cui indichiamo il risultato finale delle operazioni.

1.4.1 Simulazioni

- $S = \text{prodotto}(\text{prodotto}(3,4),2)$

```
-->prodotto(prodotto(3,4),2)
-->prodotto(3,4)
<->prodotto-return(12)
<->prodotto-return(24)
<->result(24)
```

- $S = \text{somma}(\text{somma}(1,2),\text{somma}(3,4))$

```
-->somma(somma(1,2),somma(3,4))
-->somma(1,2)
<->somma-return(3)
<->somma-return(7)
<->somma-return(10)
<->result(10)
```

- In quest'ultima simulazione mostriamo il comportamento del programma in corrispondenza di un input errato. $S = \text{somma}(a,\text{somma}(1,..))$

```
-->prodotto(a,somma(1,..))
-->a,somma(1,..)
<->somma-return(15)
<->prodotto-return(15)
<->result(15)
```

1.4.2 Gestione della Memoria

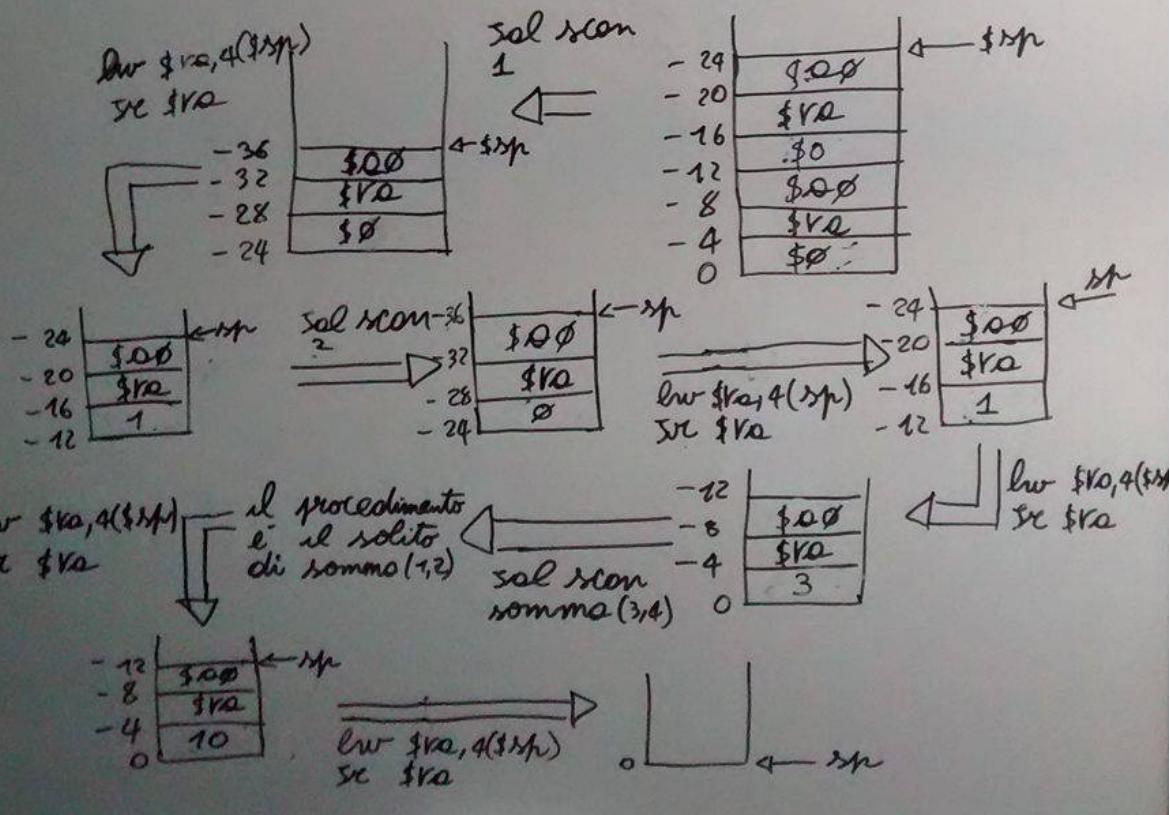
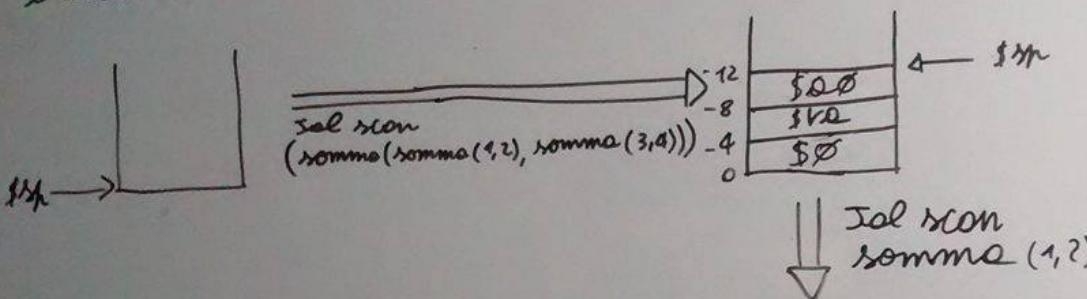
Mostriamo qui l'andamento della memoria nel caso della stringa $\mathbf{S} = \text{somma}(\text{somma}(1,2),\text{somma}(3,4))$. Per rendere più chiaro il contenuto dei seguenti screenshot abbiamo scritto a mano i passaggi interessati.

Io sono dedico 3 word nello stock per:

- indirizzo della stringa (\$0)
 - indirizzo di ritorno al chiamante (\$rs)
 - word dedicato al risultato (\$rs)

15m - 12 →	\$0.0
5m - 8 →	\$1.2
3m - 4 →	\$1.0
15m - 0 →	\$15.0

l'evoluzione dello stock per S è il seguente:



- Main chiama JAL scan sulla stringa somma(somma(1,2),somma(3,4))

```

        User Text Segment [00400000]..[00440000]
[00400000] 8fa40000 lw $4, 0($29)           ; 183: lw $a0 0($sp) # argv
[00400004] 27a50004 addiu $5, $29, 4       ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4       ; 185: addiu $a2 $a1 4 # envp
[00400010] 00041080 sll $2, $4, 2          ; 186: sll $v0 $a0 2
[00400014] 0c100009 addu $6, $6, $2         ; 187: addu $a2 $a2 $v0
[00400018] 00000000 jal 0x00400024 [main]   ; 188: jal main
[0040001c] 3402000a ori $2, $0, 10          ; 189: nop
[00400020] 00000000 syscall                 ; 190: syscall
[00400024] 3402000d ori $2, $0, 13          ; 191: li $v0 10
[00400028] 3c011001 lui $1, 4097 [file]      ; 192: syscall # syscall 10 (exit)
[0040002c] 34240010 ori $4, $1, 16 [file]    ; 193: li $v0, 13 # syscall per aprire il file
[00400030] 34050000 ori $5, $0, 0           ; 194: la $a0, file # carico il nome del file
[00400034] 34060000 ori $6, $0, 0           ; 195: jal main
[00400038] 00000000 syscall                 ; 196: li $a0, 0 # flag solo lettura
[0040003c] 00027021 addu $14, $0, $2        ; 197: addu $a0, $a0, 0 # (ignored)
[00400040] 28410000 slti $1, $2, 0          ; 198: addu $a1, $a1, 0 # flag solo lettura
[00400044] 14200138 bne $1, $0, 1248 [err-0x00400044]
[00400048] 34020000 ori $2, $0, 14          ; 199: li $v0, 14 # syscall per leggere dal file
[0040004c] 000e2021 addu $4, $0, $14         ; 200: move $t6, $v0 # salvo il descrittore del file
[00400050] 3c011001 lui $1, 4097 [buffer]    ; 201: move $a0, $t6 # carico il descrittore del file
[00400054] 34250057 ori $5, $1, 87 [buffer] ; 202: la $a1, buffer # carico l'indirizzo del buffer
[00400058] 34060400 ori $6, $0, 1024        ; 203: li $a2, 1024 # grandezza del buffer
[0040005c] 0000000c syscall                 ; 204: syscall
[00400060] 3c011001 lui $1, 4097 [rightarrow] ; 205: la $a0,rightarrow
[00400064] 342404b4 ori $4, $1, 1204 [rightarrow] ; 206: li $v0, 4
[00400068] 34020004 ori $2, $0, 4           ; 207: syscall
[00400070] 34020004 ori $2, $0, 4           ; 208: syscall
[00400074] 3c011001 lui $1, 4097 [buffer]    ; 209: li $v0, 4
[00400078] 34240057 ori $4, $1, 87 [buffer] ; 210: move $t6, $v0 # salvo il descrittore del file
[0040007c] 0000000c syscall                 ; 211: syscall
[00400080] 34020004 ori $2, $0, 4           ; 212: move $a0, $t6 # carico il descrittore del file
[00400084] 3c011001 lui $1, 4097 [newl]      ; 213: la $a0, newl
[00400088] 34240457 ori $4, $1, 1111 [newl]  ; 214: li $v0, 4
[0040008c] 0000000c syscall                 ; 215: syscall

```

- Viene aggiornato lo stack con le prime 3 word dedicate ai valori di:\$a0, \$ra, \$zero

```

        User Text Segment [00400000]..[00440000]
[00400000] 34060400 ori $0, $0, 1024        ; 20: li $a2, 1024 # grandezza del buffer
[00400004] 0000000c syscall                 ; 21: syscall
[00400008] 3c011001 lui $1, 4097 [rightarrow] ; 22: la $a0,rightarrow
[00400012] 342404b4 ori $4, $1, 1204 [rightarrow] ; 23: li $v0, 4
[00400016] 34020004 ori $2, $0, 4           ; 24: syscall
[00400020] 0000000c syscall                 ; 25: syscall
[00400024] 34020004 ori $2, $0, 4           ; 26: li $v0, 4
[00400028] 3c011001 lui $1, 4097 [buffer]    ; 27: la $a0,buffer
[00400032] 34240057 ori $4, $1, 87 [buffer] ; 28: li $v0, 4
[00400036] 0000000c syscall                 ; 29: syscall
[00400040] 34020004 ori $2, $0, 4           ; 30: li $v0, 4
[00400044] 3c011001 lui $1, 4097 [newl]      ; 31: la $a0,newl
[00400048] 34240457 ori $4, $1, 1111 [newl]  ; 32: li $v0, 4
[00400052] 0000000c syscall                 ; 33: syscall
[00400056] 3c011001 lui $1, 4097 [buffer]    ; 34: la $a0,buffer
[00400060] 34240057 ori $4, $1, 87 [buffer] ; 35: li $v0, 4
[00400064] 0000000c syscall                 ; 36: syscall
[00400068] 3c011001 lui $1, 4097 [rightarrow] ; 37: la $a0,rightarrow
[00400072] 342404b4 ori $4, $1, 1204 [rightarrow] ; 38: li $v0, 4
[00400076] 34020004 ori $2, $0, 4           ; 39: syscall
[00400080] 0000000c syscall                 ; 40: syscall
[00400084] 3c011001 lui $1, 4097 [buffer]    ; 41: la $a0,buffer
[00400088] 34240457 ori $4, $1, 1111 [newl]  ; 42: li $v0, 4
[00400092] 0000000c syscall                 ; 43: syscall
[00400096] 3c011001 lui $1, 4097 [buffer]    ; 44: la $a0,newl
[004000a0] 34240057 ori $4, $1, 87 [buffer] ; 45: li $v0, 4
[004000a4] 0000000c syscall                 ; 46: syscall
[004000a8] 3c011001 lui $1, 4097 [buffer]    ; 47: jal scan # salta all'indirizzo dell'etichetta
[004000ac] 0c100035 jal 0x00400044 [scan]    ; 48: jal scan # salta all'indirizzo dell'etichetta
[004000b0] 00082021 addu $4, $0, $8          ; 49: la $a0,fires
[004000b4] 34020000 nello scan               ; 50: move $a0,$t0 # stampa il risultato trovato
[004000b8] 0000000c syscall                 ; 51: syscall
[004000bc] 3c011001 lui $1, 4097 [par]       ; 52: move $a0,$t0 # stampa il risultato trovato
[004000c0] 342404b2 ori $4, $1, 1202 [par]   ; 53: li $v0, 1
[004000c4] 34020004 ori $2, $0, 4           ; 54: syscall
[004000c8] 0000000c syscall                 ; 55: la $a0,par
[004000cc] 0c100145 jal 0x00400514 [close]  ; 56: li $v0, 4
[004000d0] 0810014c j 0x00400530 [quit]     ; 57: syscall
[004000d4] 23bdffff addi $29, $29, -12        ; 58: jal close
[004000d8] afa40000 sw $4, 0($29)           ; 59: j quit # si salta all'etichetta che farà
[004000dc] afbf0004 sw $31, 4($29)           ; 60: move $a0,$t0 # stampa il risultato trovato
[004000e0] 0000000c syscall                 ; 61: move $a0,$t0 # stampa il risultato trovato
[004000e4] 0000000c syscall                 ; 62: sw $a0,0($sp) # salva l'indirizzo della stringa
[004000e8] 0000000c syscall                 ; 63: sw $ra,4($sp) # salva l'indirizzo di ritorno

```

- Il primo argomento della prima somma è somma(1,2), viene quindi chiamata la JAL scan. Viene modificato il valore di \$sp, in modo da allocare altre 3 word per i valori di \$a0, \$ra, \$zero

```

QtSpim
File Simulator Registers Text Segment Data Segment Window Help
Int Regs [16] Data Text
Int Regs [16] Text
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 10010000
R2 [v0] = 4
R3 [v1] = 0
R4 [a0] = 1001005d
R5 [a1] = 10010067
R6 [a2] = 400
R7 [a3] = 0
R8 [t0] = 29
R9 [t1] = 29
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 1001005d
R14 [t6] = c
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] = 7ffff7f4
R30 [s8] = 0
R31 [ra] = 4001d4

[004000d4] 23bdfff4 addi $29, $29, -12 ; 61: addi $sp,$sp,-12 # diminuisco il valore dello stack pointer di 12, facendo spazio per 12/4=3 word.
[004000d8] afa40000 sw $4, 0($29) ; 62: sw $a0,0($sp) # salva l'indirizzo della stringa del chiamante
[004000dc] afbf0004 sw $31, 4($29) ; 63: sw $ra,4($sp) # salva l'indirizzo di ritorno
[004000e0] afa00008 sw $0, 8($29) ; 64: sw $0,8($sp) # la terza parola allocata nello stack viene dedicata al risultato che sarà contenuto in $v0
[004000e4] 24840002 addiu $4, $4, 2 ; 67: addu $a0,$a0,2 # verifico se e' una somma o una sottrazione
[004000e8] 90880000 lbu $8, 0($4) ; 68: lbu $t0,($a0) # leggo un carattere della stringa (pseudoistruzione)
[004000ec] 34090074 ori $9, $0, 116 ; 69: li $t1,'t'
[004000f0] 1109005c beg $8, $9, 368 [subt-0x004000f0]
[004000f4] 3409006d ori $9, $0, 109 ; 71: li $t1,'m'
[004000f8] 1109002d beg $8, $9, 180 [sum-0x004000f8]
[004000fc] 2484fffe addiu $4, $4, -2 ; 76: addu $a0,$a0,-2 # si torna all'inizio della stringa
[00400100] 90880000 lbu $8, 0($4) ; 77: lbu $t0,($a0) # leggo un carattere della stringa (pseudoistruzione)
[00400104] 34090064 ori $9, $0, 100 ; 78: li $t1,'d'
[00400108] 11090086 beg $8, $9, 536 [divi-0x00400108]
[00400114] 34090029 ori $9, $0, 41 ; 83: li $t1,''
[00400118] 1109001f beg $8, $9, 124 [ind-0x00400118]
[0040011c] 3409002c ori $9, $0, 44 ; 85: li $t1,''
[00400120] 1109001d beg $8, $9, 116 [ind-0x00400120]
[00400124] 34090020 ori $9, $0, 32 ; 87: li $t1,' '
[00400128] 1109001b beg $8, $9, 108 [ind-0x00400128]
[0040012c] 3409002d ori $9, $0, 45 ; 90: li $t1,'-'
[00400130] 1109000e beg $8, $9, 56 [nega-0x00400130]
[00400134] 3102000f andi $2, $8, 15 ; 92: andi $v0,$t0,0x0F # and logico bit a bit
(immediato) per passare da ASCII ad intero
[00400138] 24840001 addiu $4, $4, 1 ; 95: addu $a0,$a0,1 # incremento di 1 la posizione sulla stringa
[0040013c] 90880000 lbu $8, 0($4) ; 96: lbu $t0,($a0) # leggo il carattere
[00400140] 29090030 slti $9, $8, 48 ; 98: slti $t1,$t0,48
[00400144] 15200007 bne $9, $9, 28 [deallocation-0x00400144]

```

- La chiamata di JAL scan questa volta è per il primo valore della somma(1,2), cioè 1. Prima della chiamata di JR \$ra, verrà modificato il valore di ritorno \$v0 con 1

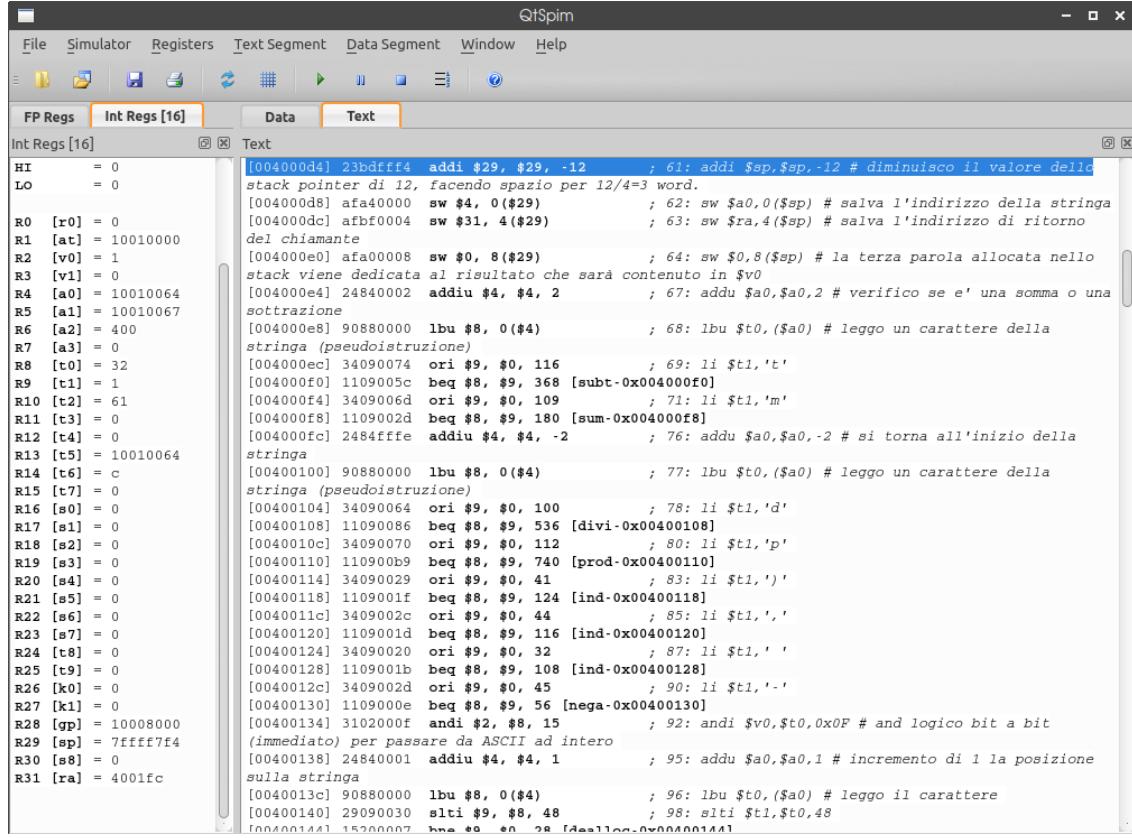
```

QtSpim
File Simulator Registers Text Segment Data Segment Window Help
Int Regs [16] Data Text
Int Regs [16] Text
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 10010000
R2 [v0] = 4
R3 [v1] = 0
R4 [a0] = 10010063
R5 [a1] = 10010067
R6 [a2] = 400
R7 [a3] = 0
R8 [t0] = 31
R9 [t1] = 61
R10 [t2] = 1
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 10010063
R14 [t6] = c
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] = 7ffff7e8
R30 [s8] = 0
R31 [ra] = 4001d4

[004000d4] 23bdfff4 addi $29, $29, -12 ; 61: addi $sp,$sp,-12 # diminuisco il valore dello stack pointer di 12, facendo spazio per 12/4=3 word.
[004000d8] afa40000 sw $4, 0($29) ; 62: sw $a0,0($sp) # salva l'indirizzo della stringa del chiamante
[004000dc] afbf0004 sw $31, 4($29) ; 63: sw $ra,4($sp) # salva l'indirizzo di ritorno
[004000e0] afa00008 sw $0, 8($29) ; 64: sw $0,8($sp) # la terza parola allocata nello stack viene dedicata al risultato che sarà contenuto in $v0
[004000e4] 24840002 addiu $4, $4, 2 ; 67: addu $a0,$a0,2 # verifico se e' una somma o una sottrazione
[004000e8] 90880000 lbu $8, 0($4) ; 68: lbu $t0,($a0) # leggo un carattere della stringa (pseudoistruzione)
[004000ec] 34090074 ori $9, $0, 116 ; 69: li $t1,'t'
[004000f0] 1109005c beg $8, $9, 368 [subt-0x004000f0]
[004000f4] 3409006d ori $9, $0, 109 ; 71: li $t1,'m'
[004000f8] 1109002d beg $8, $9, 180 [sum-0x004000f8]
[004000fc] 2484fffe addiu $4, $4, -2 ; 76: addu $a0,$a0,-2 # si torna all'inizio della stringa
[00400100] 90880000 lbu $8, 0($4) ; 77: lbu $t0,($a0) # leggo un carattere della stringa (pseudoistruzione)
[00400104] 34090064 ori $9, $0, 100 ; 78: li $t1,'d'
[00400108] 11090086 beg $8, $9, 536 [divi-0x00400108]
[00400114] 34090070 ori $9, $0, 112 ; 80: li $t1,'p'
[00400118] 110900b9 beg $8, $9, 740 [prod-0x00400110]
[0040011c] 34090029 ori $9, $0, 41 ; 83: li $t1,''
[00400120] 1109001f beg $8, $9, 124 [ind-0x00400118]
[00400124] 3409002c ori $9, $0, 44 ; 85: li $t1,''
[00400128] 1109001d beg $8, $9, 116 [ind-0x00400120]
[0040012c] 34090020 ori $9, $0, 32 ; 87: li $t1,' '
[0040012e] 1109001b beg $8, $9, 108 [ind-0x00400128]
[00400130] 1109000e beg $8, $9, 56 [nega-0x00400130]
[00400134] 3102000f andi $2, $8, 15 ; 92: andi $v0,$t0,0x0F # and logico bit a bit
(immediato) per passare da ASCII ad intero
[00400138] 24840001 addiu $4, $4, 1 ; 95: addu $a0,$a0,1 # incremento di 1 la posizione sulla stringa
[0040013c] 90880000 lbu $8, 0($4) ; 96: lbu $t0,($a0) # leggo il carattere
[00400140] 29090030 slti $9, $8, 48 ; 98: slti $t1,$t0,48
[00400144] 15200007 bne $9, $9, 28 [deallocation-0x00400144]

```

- Si nota che il valore dello stack è tornato uguale al valore registrato precedentemente della chiamata di JAL scan del valore 1



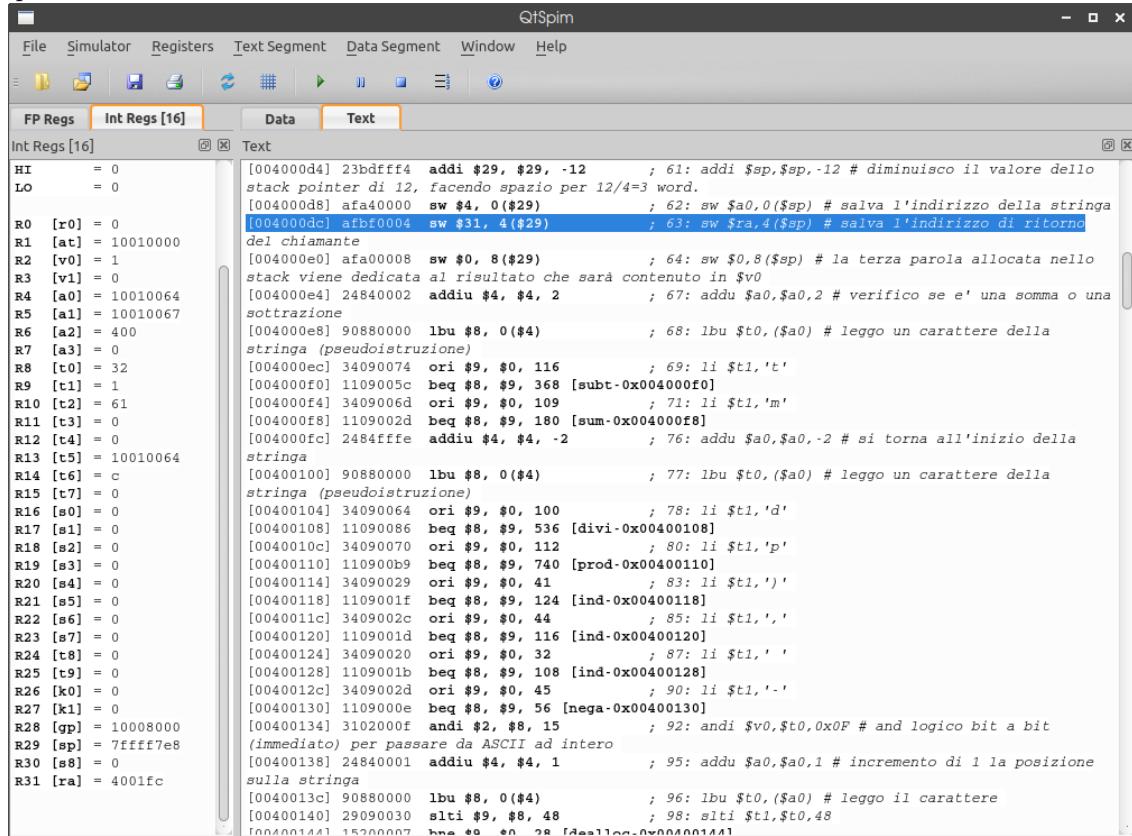
The screenshot shows the QTSpim interface with the assembly window open. The assembly code is as follows:

```

    .text
    addi $29, $29, -12      ; 61: addi $sp,$sp,-12 # diminuisco il valore della
    stack pointer di 12, facendo spazio per 12/4=3 word.
    sw $4, 0($29)           ; 62: sw $a0,0($sp) # salva l'indirizzo della stringa
    afa40000                 ; 63: sw $ra,4($sp) # salva l'indirizzo di ritorno
    del chiamante
    afa00008     sw $0, 8($29)   ; 64: sw $0,8($sp) # la terza parola allocata nello
    stack viene dedicata al risultato che sarà contenuto in $v0
    24840002 addiu $4, $4, 2   ; 67: addu $a0,$a0,2 # verifico se e' una somma o una
    sottrazione
    90880000 lbu $8, 0($4)    ; 68: lbu $t0,($a0) # leggo un carattere della
    stringa (pseudoistruzione)
    34090074 ori $9, $0, 116   ; 69: li $t1,'t'
    1109005c beq $8, $9, 368 [subt-0x004000f0]
    3409006d ori $9, $0, 109   ; 71: li $t1,'m'
    1109002d beq $8, $9, 180 [sum-0x004000f8]
    2484ffff addiu $4, $4, -2  ; 76: addu $a0,$a0,-2 # si torna all'inizio della
    stringa
    90880000 lbu $8, 0($4)    ; 77: lbu $t0,($a0) # leggo un carattere della
    stringa (pseudoistruzione)
    34090064 ori $9, $0, 100   ; 78: li $t1,'d'
    11090086 beq $8, $9, 536 [divi-0x00400108]
    34090070 ori $9, $0, 112   ; 80: li $t1,'p'
    110900b9 beq $8, $9, 740 [prod-0x00400110]
    34090029 ori $9, $0, 41    ; 83: li $t1,''
    1109001f beq $8, $9, 124 [ind-0x00400118]
    3409002c ori $9, $0, 44    ; 85: li $t1,''
    1109001d beq $8, $9, 116 [ind-0x00400120]
    34090020 ori $9, $0, 32    ; 87: li $t1,' '
    1109001b beq $8, $9, 108 [ind-0x00400128]
    3409002d ori $9, $0, 45    ; 90: li $t1,'-'
    110900e6 beq $8, $9, 56 [nega-0x00400130]
    3102000f andi $2, $8, 15   ; 92: andi $v0,$t0,0x0F # and logico bit a bit
    (immediato) per passare da ASCII ad intero
    24840001 addiu $4, $4, 1   ; 95: addu $a0,$a0,1 # incremento di 1 la posizione
    sulla stringa
    90880000 lbu $8, 0($4)    ; 96: lbu $t0,($a0) # leggo il carattere
    29090030 slti $9, $8, 48   ; 98: slti $t1,$t0,48
    15200007 hwe $0, $0, 28 [data1100...000000001111]

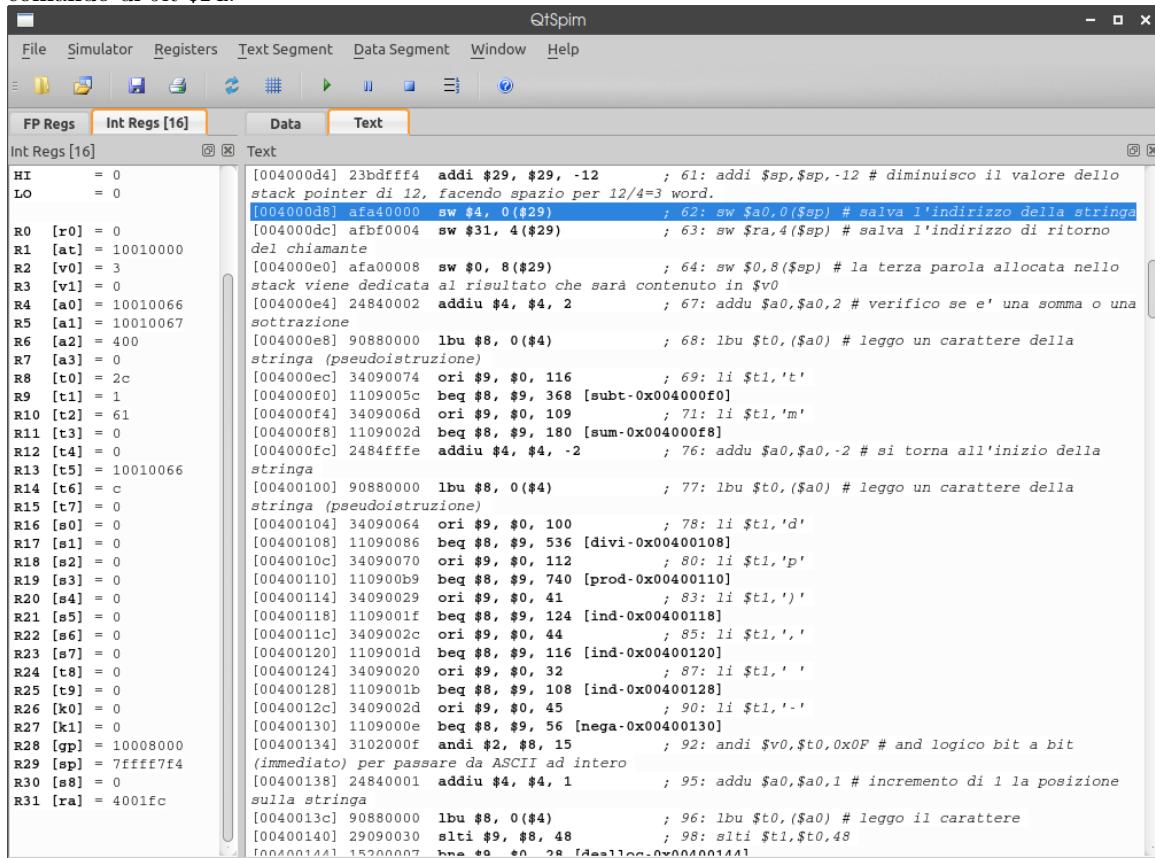
```

- Viene chiamato il JAL scan per il secondo valore di somma(1,2), cioè 2. Il valore di \$sp viene aggiornato, possiamo notare che il suo valore è uguale alla precedente chiamata di JAL scan di 1, questo è normale dato che viene allocata la stessa dimensione nello stack



The screenshot shows the QTSpim interface with the assembly window open. The assembly code is identical to the previous one, but the register values show the stack pointer (\$sp) has been updated to reflect the previous call's allocation.

A questo punto termina anche la `scan` di somma(1,2) che aggiornerà il valore di `$v0` con 3 prima del comando di `JR $ra`.



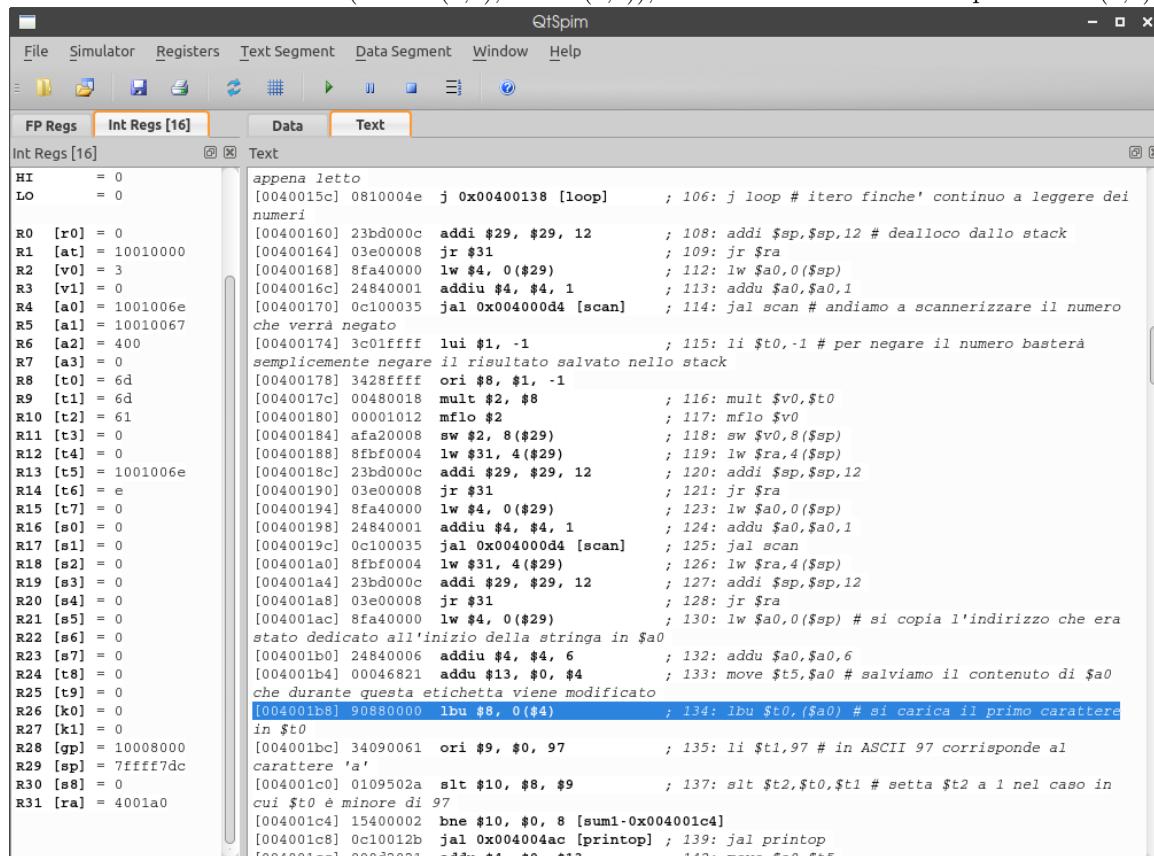
The screenshot shows the QtSpim debugger interface. The Registers window displays integer and floating-point registers (HI, LO, R0-R31). The assembly window shows the following code:

```

[004000d4] 23bdfff4 addi $29, $29, -12 ; 61: addi $sp,$sp,-12 # diminuisco il valore dello stack pointer di 12, facendo spazio per 12/4=3 word.
[004000d8] afa40000 sw $4, 0($29) ; 62: sw $a0,0($sp) # salva l'indirizzo della stringa dei chiamante
[004000dc] afbf0004 sw $31, 4($29) ; 63: sw $ra,4($sp) # salva l'indirizzo di ritorno
[004000e0] afa00008 sw $0, 8($29) ; 64: sw $0,8($sp) # la terza parola allocata nello stack viene dedicata al risultato che sarà contenuto in $v0
[004000e4] 24840002 addiu $4, $4, 2 ; 67: addu $a0,$a0,2 # verifico se e' una somma o una sottrazione
[004000e8] 90880000 lbu $8, 0($4) ; 68: lbu $t0,($a0) # leggo un carattere della stringa (pseudoistruzione)
[004000ec] 34090074 ori $9, $0, 116 ; 69: li $t1,'t'
[004000f0] 1109005c beq $8, $9, 368 [subt-0x004000f0]
[004000f4] 3409006d ori $9, $0, 109 ; 71: li $t1,'m'
[004000f8] 1109002d beq $8, $9, 180 [sum-0x004000f8]
[004000fc] 2484ffff addiu $4, $4, -2 ; 76: addu $a0,$a0,-2 # si torna all'inizio della stringa
[00400100] 90880000 lbu $8, 0($4) ; 77: lbu $t0,($a0) # leggo un carattere della stringa (pseudoistruzione)
[00400104] 34090064 ori $9, $0, 100 ; 78: li $t1,'d'
[00400108] 11090086 beq $8, $9, 536 [divi-0x00400108]
[0040010c] 34090070 ori $9, $0, 112 ; 80: li $t1,'p'
[00400110] 110900b9 beq $8, $9, 740 [prod-0x00400110]
[00400114] 34090029 ori $9, $0, 41 ; 83: li $t1,')'
[00400118] 1109001f beq $8, $9, 124 [ind-0x00400118]
[0040011c] 3409002c ori $9, $0, 44 ; 85: li $t1,''
[00400120] 1109001d beq $8, $9, 116 [ind-0x00400120]
[00400124] 34090020 ori $9, $0, 32 ; 87: li $t1,' '
[00400128] 1109001b beq $8, $9, 108 [ind-0x00400128]
[0040012c] 3409002d ori $9, $0, 45 ; 90: li $t1,'-'
[00400130] 1109000e beq $8, $9, 56 [neg-0x00400130]
[00400134] 3102000f andi $2, $8, 15 ; 92: andi $v0,$t0,0xF # and logico bit a bit (immediato) per passare da ASCII ad intero
[00400138] 24840001 addiu $4, $4, 1 ; 95: addu $a0,$a0,1 # incremento di 1 la posizione sulla stringa
[0040013c] 90880000 lbu $8, 0($4) ; 96: lbu $t0,($a0) # leggo il carattere
[00400140] 29090030 slti $9, $8, 48 ; 98: slti $t1,$t0,48
[00400144] 15200007 bne $9, $9, 28 [label-0x00400144]

```

- La terza word nello stack viene aggiornata con il valore della `scan` precedente, si può quindi passare al secondo valore di somma(somma(1,2),somma(3,4)), viene chiamato il JAL `scan` per somma(3,4)



The screenshot shows the QtSpim debugger interface. The Registers window displays integer and floating-point registers (HI, LO, R0-R31). The assembly window shows the following code:

```

[0040015c] 0810004e jal 0x00400138 [loop] ; 106: jal loop # itero finche' continuo a leggere dei numeri
[00400160] 23bd000c addi $29, $29, 12 ; 108: addi $sp,$sp,12 # dealloco dallo stack
[00400164] 03e00008 jr $31 ; 109: jr $ra
[00400168] 8fa40000 lw $4, 0($29) ; 110: lw $a0,0($sp)
[0040016c] 24840001 addiu $4, $4, 1 ; 113: addu $a0,$a0,1
[00400170] 0c100035 jal 0x004000d4 [scan] ; 114: jal scan # andiamo a scannerizzare il numero che verrà negato
[00400174] 3c01ffff lui $1, -1 ; 115: li $t0,-1 # per negare il numero basterà semplicemente negare il risultato salvato nello stack
[00400178] 3428ffff ori $8, $1, -1
[0040017c] 00480018 mult $2, $8 ; 116: mult $v0,$t0
[00400180] 00001012 mflo $2 ; 117: mflo $v0
[00400184] afa20008 sw $2, 8($29) ; 118: sw $v0,8($sp)
[00400188] 8fbf0004 lw $31, 4($29) ; 119: lw $ra,4($sp)
[0040018c] 23bd000c addi $29, $29, 12 ; 120: addi $sp,$sp,12
[00400190] 03e00008 jr $31 ; 121: jr $ra
[00400194] 8fa40000 lw $4, 0($29) ; 123: lw $a0,0($sp)
[00400198] 24840001 addiu $4, $4, 1 ; 124: addu $a0,$a0,1
[0040019c] 0c100035 jal 0x004000d4 [scan] ; 125: jal scan
[004001a0] 8fbf0004 lw $31, 4($29) ; 126: lw $ra,4($sp)
[004001a4] 23bd000c addi $29, $29, 12 ; 127: addi $sp,$sp,12
[004001a8] 03e00008 jr $31 ; 128: jr $ra
[004001ac] 8fa40000 lw $4, 0($29) ; 130: lw $a0,0($sp) # si copia l'indirizzo che era stato dedicato all'inizio della stringa in $a0
[004001b0] 24840006 addiu $4, $4, 6 ; 132: addu $a0,$a0,6
[004001b4] 00046821 addu $13, $0, $4 ; 133: move $t5,$a0 # salviamo il contenuto di $a0
[004001b8] 90880000 lbu $8, 0($4) ; 134: lbu $t0,($a0) # si carica il primo carattere in $t0
[004001bc] 34090061 ori $9, $0, 97 ; 135: li $t1,97 # in ASCII 97 corrisponde al carattere 'a'
[004001c0] 0109502a slt $10, $8, $9 ; 137: slt $t2,$t0,$t1 # setta $t2 a 1 nel caso in cui $t0 è minore di 97
[004001c4] 15400002 bne $10, $0, 8 [sum1-0x004001c4]
[004001c8] 0c10012b jal 0x004004ac [printtop] ; 139: jal printtop
[004001cc] 00000001 add $10, $10, $10 ; 140: move $t0,$t0+$t1

```

- Viene chiamato il JAL **scan** del primo valore di somma(3,4), cioè 3. Alla fine di questa chiamata sarà restituito il valore 3 nel registro \$v0

```

QTSpim
File Simulator Registers Text Segment Data Segment Window Help
Int Regs [16] Data Text
Int Regs [16]
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 10010000
R2 [v0] = 3
R3 [v1] = 0
R4 [a0] = 1001006e
R5 [a1] = 10010067
R6 [a2] = 400
R7 [a3] = 0
R8 [t0] = 33
R9 [t1] = 61
R10 [t2] = 1
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 1001006e
R14 [t6] = e
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] = 7fffff7d0
R30 [s8] = 0
R31 [ra] = 4001d4

[004000d4] 23bdfff4 addi $29, $29, -12 ; 61: addi $sp,$sp,-12 # diminuisco il valore dello stack pointer di 12, facendo spazio per 12/4=3 word.
[004000d8] afa40000 sw $4, 0($29) ; 62: sw $a0,0($sp) # salva l'indirizzo della stringa del chiamante
[004000dc] afbf0004 sw $31, 4($29) ; 63: sw $ra,4($sp) # salva l'indirizzo di ritorno
[004000e0] afa00008 sw $0, 8($29) ; 64: sw $0,8($sp) # la terza parola allocata nello stack viene dedicata al risultato che sarà contenuto in $v0
[004000e4] 24840002 addiu $4, $4, 2 ; 67: addu $a0,$a0,2 # verifico se e' una somma o una sottrazione
[004000e8] 90880000 lbu $8, 0($4) ; 68: lbu $t0,($a0) # leggo un carattere della stringa (pseudoistruzione)
[004000ec] 34090074 ori $9, $0, 116 ; 69: li $t1,'t'
[004000f0] 1109005c beg $8, $9, 368 [subt-0x004000f0]
[004000f4] 3409006d ori $9, $0, 109 ; 71: li $t1,'m'
[004000f8] 1109002d beg $8, $9, 180 [sum-0x004000f8]
[004000fc] 2484fffe addiu $4, $4, -2 ; 76: addu $a0,$a0,-2 # si torna all'inizio della stringa
[00400100] 90880000 lbu $8, 0($4) ; 77: lbu $t0,($a0) # leggo un carattere della stringa (pseudoistruzione)
[00400104] 34090064 ori $9, $0, 100 ; 78: li $t1,'d'
[00400108] 11090086 beg $8, $9, 536 [divi-0x00400108]
[0040010c] 34090070 ori $9, $0, 112 ; 80: li $t1,'p'
[00400110] 110900b9 beg $8, $9, 740 [prod-0x00400110]
[00400114] 34090029 ori $9, $0, 41 ; 83: li $t1,')'
[00400118] 1109001f beg $8, $9, 124 [ind-0x00400118]
[0040011c] 3409002c ori $9, $0, 44 ; 85: li $t1,',
[00400120] 1109001d beg $8, $9, 116 [ind-0x00400120]
[00400124] 34090020 ori $9, $0, 32 ; 87: li $t1,' '
[00400128] 1109001b beg $8, $9, 108 [ind-0x00400128]
[0040012c] 3409002d ori $9, $0, 45 ; 90: li $t1,'-'
[00400130] 1109000e beg $8, $9, 56 [nega-0x00400130]
[00400134] 3102000f andi $2, $8, 15 ; 92: andi $v0,$t0,0x0F # and logico bit a bit
[00400138] 24840001 addiu $4, $4, 1 ; 95: addu $a0,$a0,1 # incremento di 1 la posizione sulla stringa
[0040013c] 90880000 lbu $8, 0($4) ; 96: lbu $t0,($a0) # leggo il carattere
[00400140] 29090030 slti $9, $8, 48 ; 98: slti $t1,$t0,48
[00400144] 15000007 bne $9, $8, 28 [label-0x00400144]

```

- Si ritorna allo **scan** di somma(3,4) e si aggiorna il valore della terza word allocata sullo stack con il valore di ritorno \$v0. Possiamo notare che il valore dello stack pointer è uguale al valore precedente alla chiamata di JAL **scan** del primo valore

```

QTSpim
File Simulator Registers Text Segment Data Segment Window Help
Int Regs [16] Data Text
Int Regs [16]
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 10010000
R2 [v0] = 3
R3 [v1] = 0
R4 [a0] = 1001006f
R5 [a1] = 10010067
R6 [a2] = 400
R7 [a3] = 0
R8 [t0] = 34
R9 [t1] = 1
R10 [t2] = 61
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 1001006f
R14 [t6] = e
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] = 7fffff7d0
R30 [s8] = 0
R31 [ra] = 4001d4

[00400108] 81d10004 lw $31, 4($29) ; 119: lw $ra,4($sp)
[0040018c] 23bd000c addi $29, $29, 12 ; 120: addi $sp,$sp,12
[00400190] 03e00008 jr $31 ; 121: jr $ra
[00400194] 8fa40000 lw $4, 0($29) ; 123: lw $a0,0($sp)
[00400198] 24840001 addiu $4, $4, 1 ; 124: addu $a0,$a0,1
[0040019c] 0c100035 jal 0x004000d4 [scan] ; 125: jal scan
[004001a0] 8fbff004 lw $31, 4($29) ; 126: lw $ra,4($sp)
[004001a4] 23bd000c addi $29, $29, 12 ; 127: addi $sp,$sp,12
[004001a8] 03e00008 jr $31 ; 128: jr $ra
[004001ac] 8fa40000 lw $4, 0($29) ; 130: lw $a0,0($sp) # si copia l'indirizzo che era stato dedicato all'inizio della stringa in $a0
[004001b0] 24840006 addiu $4, $4, 6 ; 132: addu $a0,$a0,6
[004001b4] 00046821 addu $13, $0, $4 ; 133: move $t5,$a0 # salviamo il contenuto di $a0
che durante questa etichetta viene modificato
[004001b8] 90880000 lbu $8, 0($4) ; 134: lbu $t0,($a0) # si carica il primo carattere in $t0
[004001bc] 34090061 ori $9, $0, 97 ; 135: li $t1,97 # in ASCII 97 corrisponde al carattere 'a'
[004001c0] 0109502a slt $10, $8, $9 ; 137: slt $t2,$t0,$t1 # setta $t2 a 1 nel caso in cui $t0 è minore di 97
[004001c4] 15400002 bne $10, $0, 8 [sum1-0x004001c4]
[004001c8] 0c10012b jal 0x004004ac [printop] ; 139: jal printop
[004001cc] 000d2021 addu $4, $0, $13 ; 142: move $a0,$t5
[004001d0] 0c100035 jal 0x004000d4 [scan] ; 143: jal scan # il risultato di scan viene salvato in v0
[004001d4] afa20008 sw $2, 8($29) ; 145: sw $v0,8($sp) # salvo nello stack il valore trovato dallo scan
[004001d8] 00046821 addu $13, $0, $4 ; 146: move $t5,$a0
[004001dc] 24840001 addiu $4, $4, 1 ; 148: addu $a0,$a0,1 # aggiungo 1 per saltare la virgola
[004001e0] 90880000 lbu $8, 0($4) ; 149: lbu $t0,($a0) # carica il carattere
[004001e4] 340a0061 ori $10, $0, 97 ; 150: li $t2,97 # in ASCII 97 corrisponde al carattere 'a'
[004001e8] 010a482a slt $9, $8, $10 ; 152: slt $t1,$t0,$t2
[004001ec] 15200002 bne $9, $0, 8 [sum2-0x004001ec] ; 153: bnez $t1,sum2
[004001f0] 0c10012b jal 0x004004ac [printop] ; 155: jal printop
[004001f4] 000d2021 addu $4, $0, $13 ; 157: move $a0,$t5
[004001f8] 0c100035 jal 0x004000d4 [scan] ; 158: jal scan

```

- Finito lo **scan** del secondo valore di somma(3,4), si inserisce in \$v0 il valore 7 e si torna al chiamante

The screenshot shows the QtSpim debugger interface. The assembly window displays the following code:

```

    .text
    .globl _start
_start:
    li $t1, 1
    li $t2, 2
    add $t3, $t1, $t2
    li $v0, 8
    addu $v0, $t3, $v0
    li $ra, _main
    jr $sp, $ra
_main:
    li $t1, 3
    li $t2, 4
    add $t3, $t1, $t2
    li $v0, 8
    addu $v0, $t3, $v0
    li $ra, _exit
    jr $sp, $ra
_exit:
    li $v0, 10
    syscall

```

The registers window shows the following values:

Register	Value
HI	0
LO	0
R0 [r0]	0
R1 [at]	10010000
R2 [v0]	7
R3 [v1]	0
R4 [a0]	10010071
R5 [a1]	10010067
R6 [a2]	400
R7 [a3]	0
R8 [t0]	10010071
R9 [t1]	7
R10 [t2]	61
R11 [t3]	0
R12 [t4]	0
R13 [t5]	1001006f
R14 [t6]	e
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]	7fffff7e8
R30 [s8]	0
R31 [ra]	4001a0

- Si torna al chiamante somma(somma(1,2),somma(3,4)), a questo punto viene prelevato dallo stack il valore di \$v0 e viene sommato al valore di ritorno dell'ultima scann (sarà effettuata la somma 3+7). Si aggiorna quindi il valore di ritorno \$v0 con il risultato ottenuto

The screenshot shows the QtSpim debugger interface. The assembly window displays the following code:

```

    .text
    .globl _start
_start:
    li $t1, 1
    li $t2, 2
    add $t3, $t1, $t2
    li $v0, 8
    addu $v0, $t3, $v0
    li $ra, _main
    jr $sp, $ra
_main:
    li $t1, 3
    li $t2, 4
    add $t3, $t1, $t2
    li $v0, 8
    addu $v0, $t3, $v0
    li $ra, _exit
    jr $sp, $ra
_exit:
    li $v0, 10
    syscall

```

The registers window shows the following values:

Register	Value
HI	0
LO	0
R0 [r0]	0
R1 [at]	10010000
R2 [v0]	7
R3 [v1]	0
R4 [a0]	10010071
R5 [a1]	10010067
R6 [a2]	400
R7 [a3]	0
R8 [t0]	10010071
R9 [t1]	7
R10 [t2]	61
R11 [t3]	0
R12 [t4]	0
R13 [t5]	1001006f
R14 [t6]	e
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]	7fffff800
R30 [s8]	0
R31 [ra]	4001fc

- Si aggiorna \$ra con il valore di ritorno contenuto nella seconda parola allocata nello stack e si torna al chiamante della JAL `scan` di somma(`somma(1,2),somma(3,4)`), che in questo caso è proprio il main

```

QtSpim
File Simulator Registers Text Segment Data Segment Window Help
Int Regs [16] Data Text
Int Regs [16]
Text
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 10010000
R2 [v0] = a
R3 [v1] = 0
R4 [a0] = 10010071
R5 [a1] = 10010067
R6 [a2] = 400
R7 [a3] = 0
R8 [t0] = 10010071
R9 [t1] = a
R10 [t2] = 61
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 1001006f
R14 [t6] = e
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] = 7fffff80c
R30 [s8] = 0
R31 [ra] = 40009c

[004001a8] 00046821 addu $13, $0, $4 ; 146: move $t0,$a0
[004001dc] 24840001 addiu $4, $4, 1 ; 148: addu $a0,$a0,1 # aggiungo 1 per saltare la virgola
[004001e0] 90880000 lbu $8, 0($4) ; 149: lbu $t0,($a0) # carica il carattere
[004001e4] 340a0061 ori $10, $0, 97 ; 150: li $t2,97 # in ASCII 97 corrisponde al carattere 'a'
[004001e8] 010a482a slt $9, $8, $10 ; 152: slt $t1,$t0,$t2
[004001ec] 15200002 bne $9, $0, 8($sum2) ; 153: bnez $t1,sum2
[004001f0] 0c10012b jal 0x04004ac [printop] ; 155: jal printop
[004001f4] 00020221 addu $4, $0, $13 ; 157: move $a0,$t5
[004001f8] 0c100035 jal 0x04000d4 [scan] ; 158: jal scan
[004001fc] 8fa90008 lw $9, 8($29) ; 160: lw $t1,8($sp)
[00400200] 00491020 add $2, $2, $9 ; 161: add $v0,$v0,$t1 # sommo il primo operando con il secondo operando trovato
[00400204] afa20008 sw $2, 8($29) ; 162: sw $v0,8($sp) # salvo il risultato nello stack
[00400208] 00044021 addu $8, $0, $4 ; 164: move $t0,$a0
[0040020c] 00024821 addu $9, $0, $2 ; 165: move $t1,$v0 # salvo il risultato per poter eseguire le stampe
[00400210] 3c011001 lui $1, 4097 [retsum] ; 166: la $a0,retsum
[00400214] 34240459 ori $4, $1, 1113 [retsum]
[00400218] 34020004 ori $2, $0, 4 ; 167: li $v0,4
[0040021c] 0000000c syscall ; 168: syscall
[00400220] 00092021 addu $4, $0, $9 ; 169: move $a0,$t1 # stampo il risultato
[00400224] 34020001 ori $2, $0, 1 ; 170: li $v0,1
[00400228] 0000000c syscall ; 171: syscall
[0040022c] 3c011001 lui $1, 4097 [par] ; 172: la $a0,par
[00400230] 342404b2 ori $4, $1, 1202 [par]
[00400234] 34020004 ori $2, $0, 4 ; 173: li $v0,4
[00400238] 0000000c syscall ; 174: syscall
[0040023c] 3c011001 lui $1, 4097 [newl] ; 176: la $a0,newl # vado a capo
[00400240] 34240457 ori $4, $1, 1111 [newl]
[00400244] 34020004 ori $2, $0, 4 ; 177: li $v0,4
[00400248] 0000000c syscall ; 178: syscall
[0040024c] 00082021 addu $4, $0, $8 ; 179: move $a0,$t0
[00400250] 00091021 addu $2, $0, $9 ; 180: move $v0,$t1 # copio il risultato in $v0
[00400254] 8fbff004 lw $31, 4($29) ; 182: lw $ra,4($sp)
[00400258] 23bd000c addi $29, $29, 12 ; 183: addi $sp,$sp,12
[0040025c] 03e00008 jr $31 ; 184: jr $ra

```

- Prima della chiusura i valori dei registri saranno questi:

```

QtSpim
File Simulator Registers Text Segment Data Segment Window Help
Int Regs [16] Data Text
Int Regs [16]
Text
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 10010000
R2 [v0] = a
R3 [v1] = 0
R4 [a0] = e
R5 [a1] = 10010067
R6 [a2] = 400
R7 [a3] = 0
R8 [t0] = a
R9 [t1] = a
R10 [t2] = 61
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 1001006f
R14 [t6] = e
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] = 7fffff80c
R30 [s8] = 0
R31 [ra] = 4000d0

[004004d8] sc011001 lus $1, 4097 [rightarrow]; 352: la $a0,rightarrow # stampa la stringa con id freccia
[004004bc] 342404b4 ori $4, $1, 1204 [rightarrow]
[004004c0] 34020004 ori $2, $0, 4 ; 353: li $v0,4
[004004c4] 0000000c syscall ; 354: syscall
[004004c8] 90a80000 lbu $8, 0($5) ; 355: lbu $t0,($a1)
[004004cc] 34090029 ori $9, $0, 41 ; 356: li $t1,''
[004004d0] 11090006 beq $8, $9, 24 [end-0x04004d0]; 359: beq $t0,$t1,end # nel caso in cui il carattere sotto esame è una parentesi allora si deve chiudere la stampa
[004004d4] 00082021 addu $4, $0, $8 ; 360: move $a0,$t0
[004004d8] 3402000b ori $2, $0, 11 ; 361: li $v0, 11
[004004dc] 0000000c syscall ; 362: syscall # stampa il carattere letto
[004004e0] 24a50001 addiu $5, $5, 1 ; 363: addu $a1,$a1,1 # incremento di 1 la posizione sulla stringa
[004004e4] 08100132 j 0x04004c8 [iter] ; 364: j iter # ed itera
[004004e8] 24a50001 addiu $5, $5, 1 ; 367: addu $a1,$a1,1 # incremento di 1 la posizione sulla stringa
[004004ec] 34040029 ori $4, $0, 41 ; 368: li $a0,''
[004004f0] 3402000b ori $2, $0, 11 ; 369: li $v0,11
[004004f4] 0000000c syscall ; 370: syscall
[004004f8] 3c011001 lui $1, 4097 [newl] ; 371: la $a0,newl # si va a capo
[004004fc] 34240457 ori $4, $1, 1111 [newl]
[00400500] 34020004 ori $2, $0, 4 ; 372: li $v0,4
[00400504] 0000000c syscall ; 373: syscall
[00400508] 8fbff004 lw $31, 4($29) ; 375: lw $ra,4($sp)
[0040050c] 23bd000c addi $29, $29, 12 ; 376: addi $sp,$sp,12 # e si dealloca
[00400510] 03e00008 jr $31 ; 377: jr $ra
[00400514] 34020010 ori $2, $0, 16 ; 379: li $v0, 16 # syscall per la chiusura del file
[00400518] 000e2021 addu $4, $0, $14 ; 380: move $a0, $t6 # scarico il descrittore del file
[0040051c] 0000000c syscall ; 381: syscall
[00400520] 03e00008 jr $31 ; 382: jr $ra
[00400524] 34020004 ori $2, $0, 4 ; 384: li $v0, 4
[00400528] 3c041001 lui $4, 4097 [fnf] ; 385: la $a0, fnf # stampa della stringa 'file not found'
[0040052c] 0000000c syscall ; 386: syscall
[00400530] 3402000a ori $2, $0, 10 ; 388: li $v0, 10
[00400534] 0000000c syscall ; 389: syscall

```

1.5 Codice

Abbiamo cercato di indentare il codice come appare sul simulatore MARS ma, purtroppo, non siamo riusciti a mantenere lo stesso tabbing, a causa della strana gestione degli spazi di L^AT_EX.

```

1 # Bindi Giovanni 5530804 giovanni.bindi@stud.unifi.it
2 # Puliti Gabriele 5300140 gabriele.puliti@stud.unifi.it
3 # Lippi Lorenzo 6221250 lorenzo.lippi@stud.unifi.it
4 # 01-06-2017
5
6 .data
7 fnf: .ascii "File non trovato"
8 file: .asciiz "/media/wabri/KINGSTON/repositoryGit/bitbucket/repoae/ese1/chiamate.txt"
9 buffer: .space 1024
10 newl: .asciiz "\n"
11 retsum: .asciiz "<--somma--return("
12 retsub: .asciiz "<--sottrazione--return("
13 retpro: .asciiz "<--prodotto--return("
14 retdiv: .asciiz "<--divisione--return("
15 par: .asciiz ")"
16 rightarrow:.asciiz "-->"
17 finres: .asciiz "<--result("
18 divzero:.asciiz "Errore, divisione per zero!"
19
20 .text
21 .globl main
22
23 main:
24     li $v0, 13      # syscall per aprire il file
25     la $a0, file    # carico il nome del file
26     li $a1, 0        # flag solo lettura
27     li $a2, 0        # (ignored)
28     syscall
29     move $t6, $v0    # salvo il descrittore del file
30     blt $v0, 0, err # goto error
31     li $v0, 14      # syscall per leggere dal file
32     move $a0, $t6    # carico il descrittore del file
33     la $a1, buffer # carico l'indirizzo del buffer
34     li $a2, 1024    # grandezza del buffer
35     syscall
36     la $a0,rightarrow
37     li $v0,4
38     syscall
39     li $v0,4
40     la $a0,buffer
41     syscall
42     li $v0,4
43     la $a0,newl
44     syscall
45     la $a0,buffer
46     jal scan         # salta all'indirizzo dell'etichetta 'scan' salvando nel registro $ra
47     = PC+4 (controllato con debugging)
48     move $t0,$v0
49     la $a0,finres
50     li $v0,4
51     syscall
52     move $a0,$t0      # stampa il risultato trovato nello scan
53     li $v0, 1
54     syscall
55     la $a0,par
56     li $v0,4
57     syscall
58     jal close
59     j quit           # si salta all'etichetta che fara' terminare il programma
60 scan:
61     addi $sp,$sp,-12   # diminuisco il valore dello stack pointer di 12, facendo spazio
62     per 12/4=3 word.
63     sw $a0,0($sp)      # salva l'indirizzo della stringa
64     sw $ra,4($sp)       # salva l'indirizzo di ritorno del chiamante
65     sw $0,8($sp)        # la terza parola allocata nello stack viene dedicata al
66     risultato che sara' contenuto in $v0
67             # controllo se e' somma o sottrazione, controllo il terzo valore della
68     stringa

```

```

65      # se e' una t allora sottrae altrimenti controlla se e' una m di somma
66  addu $a0,$a0,2    # verifico se e' una somma o una sottrazione
67  lbu $t0,($a0)     # leggo un carattere della stringa (pseudoistruzione)
68  li $t1,'t'
69  beq $t0,$t1,subt # controllo se e' una sottrazione
70  li $t1,'m'
71  beq $t0,$t1,sum   # controllo se e' una somma
72      # se si arriva in questo punto significa che non e' ne una somma[U+FFFD]n
prodotto
73      # si controlla se e' un prodotto o una divisione
74      # in questo caso si controlla se il valore iniziale e' una p o una d
75  addu $a0,$a0,-2    # si torna all'inizio della stringa
76  lbu $t0,($a0)     # leggo un carattere della stringa (pseudoistruzione)
77  li $t1,'d'
78  beq $t0,$t1,divi  # controllo se e' una divisione
79  li $t1,'p'
80  beq $t0,$t1,prod  # controllo se e' un prodotto
81      # nel caso in cui si trovi caratteri diversi dai precedenti, si continua
a leggere la stringa
82  li $t1,')'
83  beq $t0,$t1,ind
84  li $t1,','
85  beq $t0,$t1,ind
86  li $t1,','
87  beq $t0,$t1,ind
88      # se il carattere letto e' un meno si va a negare il numero successivo
89  li $t1,'_'
90  beq $t0,$t1,nega
91  andi $v0,$t0,0x0F  # and logico bit a bit (immediato) per passare da ASCII ad intero
92 loop:
93      # se si arriva a questo punto significa che e' stato trovato un numero
94  addu $a0,$a0,1      # incremento di 1 la posizione sulla stringa
95  lbu $t0,($a0)      # leggo il carattere
96      # controlliamo il caso in cui il carattere letto non sia un numero
97  slti $t1,$t0,48
98  bnez $t1,dealloc   # esce se non e' un numero altrimenti continua
99      # se si arriva a questo punto significa che siamo nel caso in cui il
numero letto e' di piu' cifre
100 andi $t0,$t0,0x0F   # and logico bit a bit (immediato) per passare da ASCII ad intero
101 li $t1,10            # carico in t1 il numero 10
102 mult $v0,$t1          # moltiplico il numero trovato prima per 10
103 mflo $v0              # mettiamo il contenuto della moltiplicazione precedente nel registro
$v0
104 add $v0,$v0,$t0        # sommo al risultato il numero appena letto
105 j loop                # itero finche' continuo a leggere dei numeri
106 dealloc:
107  addi $sp,$sp,12      # dealloco dallo stack
108  jr $ra
109 nega:
110      # procedura finalizzata a negare il numero antecedente al carattere '_'
111  lw $a0,0($sp)
112  addu $a0,$a0,1
113  jal scan             # andiamo a scannerizzare il numero che verra' negato
114  li $t0,-1             # per negare il numero bastera' semplicemente negare il risultato
salvato nello stack
115  mult $v0,$t0
116  mflo $v0
117  sw $v0,8($sp)
118  lw $ra,4($sp)
119  addi $sp,$sp,12
120  jr $ra
121 ind:
122  lw $a0,0($sp)
123  addu $a0,$a0,1
124  jal scan
125  lw $ra,4($sp)
126  addi $sp,$sp,12
127  jr $ra
128 sum:
129  lw $a0,0($sp)        # si copia l'indirizzo che era stato dedicato all'inizio della
stringa in $a0
130      # dato che siamo nel caso della somma, il primo argomento si trovera' al
6 carattere della stringa

```

```

131 addu $a0,$a0,6
132 move $t5,$a0      # salviamo il contenuto di $a0 che durante questa etichetta viene
133     modificato
134 lbu $t0,($a0)      # si carica il primo carattere in $t0
135 li $t1,97          # in ASCII 97 corrisponde al carattere 'a',
136     # nel caso in cui il carattere sia minore di 'a' allora significa che e'
137     un numero
138 slt $t2,$t0,$t1    # setta $t2 a 1 nel caso in cui $t0 e' minore di 97
139 bnez $t2,sum1      # qualora invece $t2=0 possiamo eseguire la somma
140 jal printop
141 sum1:
142     # estraiamo il valore di $a0 dallo stack che sara' l'argomento passato
143     alla chiamata scan
144 move $a0,$t5
145 jal scan          # il risultato di scan viene salvato in v0
146 sw $v0,8($sp)      # salvo nello stack il valore trovato dallo scan
147 move $t5,$a0
148     # andiamo a leggere il secondo operando
149 addu $a0,$a0,1      # aggiungo 1 per saltare la virgola
150 lbu $t0,($a0)      # carica il carattere
151 li $t2,97          # in ASCII 97 corrisponde al carattere 'a'
152     # nel caso in cui il carattere caricato in $t0 sia di 97 allora si va a
153 sum2
154 slt $t1,$t0,$t2
155 bnez $t1,sum2      # se non salta significa che siamo in corrispondenza di un altro
156     operatore
157 jal printop
158 sum2:              # secondo operando della somma
159 move $a0,$t5
160 jal scan
161     # torna qui quando trova un numero e viene effettuata l'operazione di
162     somma
163 lw $t1,8($sp)
164 add $v0,$v0,$t1      # sommo il primo operando con il secondo operando trovato
165 sw $v0,8($sp)      # salvo il risultato nello stack
166     #————stampo il risultato della somma————
167 move $t0,$a0
168 move $t1,$v0          # salvo il risultato per poter eseguire le stampe
169 la $a0,retsum
170 li $v0,4
171 syscall
172 move $a0,$t1          # stampo il risultato
173 li $v0,1
174 syscall
175 move $a0,$t1          # stampo il risultato
176 li $v0,4
177 syscall
178 move $a0,$t0
179 move $v0,$t1          # copio il risultato in $v0
180     # l'operazione di somma e' stata conclusa, ora si deve tornare al
181     chiamante della somma
182 lw $ra,4($sp)
183 addi $sp,$sp,12
184 jr $ra
185 subt:
186     lw $a0,0($sp)      # leggo il puntatore corrente nella stringa dallo stack
187     # salto al primo operando: si salta al 12esimo carattere: 11 sono i
188     caratteri di 'sottrazione' piu' 1 per la parentesi
189     addu $a0,$a0,12
190 move $t5,$a0
191 lbu $t1,($a0)          # carico su $t1 il carattere corrente
192 li $t2,97              # in ASCII 97 corrisponde al carattere 'a',
193     # se il carattere letto e' minore di 97 allora significa che e'
     un numero quindi salta a subt1
     bnez $t3,subt1         # se $t1 e' settato a 1 significa che dobbiamo leggere il primo
     operatore
     li $t2,122            # in ASCII 122 corrisponde al carattere 'z'

```

```

194    slt $t3,$t2,$t1      # se il carattere letto risulta minore di 97 sto leggendo un
195    bnez $t3,subt1
196    jal printop      # altrimenti e' una lettera quindi salta a printop
197 subt1:
198    move $a0,$t5
199    jal scan
200          # torna qui quando trova un numero
201    sw $v0,8($sp)      # salvo nello stack il primo argomento letto
202    move $t5,$a0
203    addu $a0,$a0,1      # aggiungo 1 per saltare la virgola
204    lbu $t1,($a0)      # si legge il carattere
205    li $t2,97      # in ASCII 97 corrisponde al carattere 'a'
206    slt $t3,$t1,$t2
207    bnez $t3,subt2      # come prima se si trova un numero significa che abbiamo trovato
208          il secondo operatore quindi si salta a subt2
209    jal printop      # se e' una lettera significa che l'operando e' un altro operatore
210 subt2:
211    move $a0,$t5
212    jal scan
213          # torna qui quando trova un numero
214    lw $t1,8($sp)
215    sub $v0,$t1,$v0      # sottraggo il primo con il secondo
216    sw $v0,8($sp)      # salvo il risultato nello stack
217          #————stampo il risultato della sottrazione————
218    move $t3,$a0
219    move $t0,$v0      # salvo il risultato per poter eseguire le stampe
220    la $a0,retsub
221    li $v0,4
222    syscall
223    move $a0,$t0      # stampo il risultato
224    li $v0,1
225    syscall
226    la $a0,par
227    li $v0,4
228    syscall
229          #
230    la $a0,newl      # vado a capo
231    li $v0,4
232    syscall
233    move $a0,$t3
234    move $v0,$t0      # riporto il risultato in $v0 e ritorno al chiamante
235    lw $ra,4($sp)
236    addi $sp,$sp,12
237    jr $ra
238 divi:
239    lw $a0,0($sp)      # estraggo il puntatore al carattere corrente nella stringa dallo
240          stack
241          addu $a0,$a0,10      # salto al primo operando (9 caratteri per 'divisione' ed 1
242          per la parentesi)
243    move $t5,$a0      # salvo l'indirizzo in $t0
244    lbu $t3,($a0)      # e carico il primo carattere in $t3
245    li $t2,97      # dal momento che in ASCII 97 corrisponde al carattere 'a'
246    slt $t4,$t3,$t2      # se il carattere letto risulta minore di 97 sto leggendo un
247          numero
248    bnez $t4,divi1      # e posso andare ad eseguire la divisione
249    jal printop
250 divi1:
251    move $a0,$t5      # eseguo la scan dell primo operatore
252    jal scan
253          # torna qui quando trova un numero
254    sw $v0,8($sp)      # salvo nello stack il primo operando della divisone appena letto
255    move $t5,$a0      # riacquisisco l'indice sulla stringa
256    addu $a0,$a0,1      # ci aggiungo 1 per saltare la virgola
257    lbu $t3,($a0)      # e carico in $t3 cio' che vado a leggere
258    li $t2,97      # dal momento che in ASCII 97 corrisponde al carattere 'a'
259    slt $t1,$t3,$t2      # se il carattere letto risulta minore di 97 sto leggendo un
260          numero
261    bnez $t1,divi2      # e posso andare ad eseguire la divisione
262    jal printop
263 divi2:
264    move $a0,$t5
265    jal scan

```

```

261          # torna qui quando trova un numero
262  beqz $v0,diverror  # se il numero letto e' zero allora si stampa un messaggio di
263  errore
264  lw $t1,8($sp)
265  div $t1,$v0          # divido il primo con il secondo
266  mflo $v0
267  sw $v0,8($sp)        # salvo il risultato nello stack
268          #————stampo il risultato della divisione————
269  move $t4,$a0
270  move $t0,$v0          # salvo il risultato per poter eseguire le stampe
271  la $a0,retdiv
272  li $v0,4
273  syscall
274  move $a0,$t0          # stampo il risultato
275  li $v0,1
276  syscall
277  la $a0,par
278  li $v0,4
279  syscall
280          #
281  la $a0,newl    # vado a capo
282  li $v0,4
283  syscall
284  move $a0,$t4
285  move $v0,$t0          # riacquisisco i valori
286  lw $ra,4($sp)        # e ritorno al chiamante
287  addi $sp,$sp,12
288  jr $ra
289 diverror:
290  la $a0,divzero      # stampo la stringa di errore per la divisione per 0
291  li $v0,4
292  syscall
293  li $v0,10          # ed esco
294  syscall
295 prod:
296  lw $a0,0($sp)        # estraggo il puntatore al carattere corrente nella stringa dallo
297  stack
298  addu $a0,$a0,9       # salto al primo operando (8 caratteri di 'prodotto' ed 1 di '('
299  move $t5,$a0          # salvo l'indirizzo in $t0
300  lbu $t3,($a0)        # e carico il primo carattere in $t3
301  li $t2,97            # dal momento che in ASCII 97 corrisponde al carattere 'a'
302  slt $t4,$t3,$t2       # se il carattere letto risulta minore di 97 sto leggendo un
303  numero
304  bnez $t4,prod1       # e posso continuare con il prodotto
305  jal printop
306 prod1:
307  move $a0,$t5
308  jal scan
309          # torna qui quando trova un numero
310  sw $v0,8($sp)        # salvo nello stack il primo operando letto
311  addu $a0,$a0,1       # incremento l'indice di 1 per saltare la virgola
312  move $t5,$a0          # salvo l'indirizzo del carattere corrente in $t0
313  lbu $t3,($a0)        # e carico il primo carattere in $t3
314  li $t2,97            # dal momento che in ASCII 97 corrisponde al carattere 'a'
315  slt $t4,$t1,$t2       # se il carattere letto risulta minore di 97 sto leggendo un
316  numero
317  bnez $t4,prod2       # e posso eseguire il prodotto
318  jal printop
319 prod2:
320  move $a0,$t5
321  jal scan
322          # torna qui quando trova un numero
323  lw $t1,8($sp)        # riacquisisco il primo operando
324  mult $v0,$t1          # ed eseguo il prodotto
325  mflo $v0
326  sw $v0,8($sp)        # salvo il risultato nello stack
327          #————stampo il risultato del prodotto————
328  move $t4,$a0
329  move $t5,$v0          # salvo il risultato per poter eseguire le stampe
          #
          la $a0,retpro
          li $v0,4
          syscall
          move $a0,$t5          # stampo il risultato

```

```

330    li $v0,1
331    syscall
332    la $a0,par
333    li $v0,4
334    syscall
335    la $a0,newl      # e vado a capo
336    li $v0,4
337    syscall
338    #
339    move $a0,$t4
340    move $v0,$t5      # riacquisisco i valori
341    lw $ra,4($sp)     # e ritorno al chiamante
342    addi $sp,$sp,12    # dealloco
343    jr $ra
344 printop:           # questa procedura serve a stampare l'operazione fatta dal chiamante
345    addi $sp,$sp,-12
346    sw $ra,4($sp)      # salva l'indirizzo di ritorno del chiamante
347    move $a1,$a0
348    la $a0,rightarrow  # stampa la stringa con la freccia
349    li $v0,4
350    syscall
351 iter:              # stampa la stringa finche non trova la parentesi chiusa
352    lbu $t0,($a1)
353    li $t1,)'
354    beq $t0,$t1,end    # nel caso in cui il carattere sotto esame e' una parentesi
355    allora si deve chiudere la stampa
356    move $a0,$t0
357    li $v0,11
358    syscall            # stampa il carattere letto
359    addu $a1,$a1,1      # incremento di 1 la posizione sulla stringa
360    j iter              # ed itera
361 end:               # se si arriva qui la stampa dell'operazione effettuata e' stata
362 conclusa
363    addu $a1,$a1,1      # incremento di 1 la posizione sulla stringa
364    li $a0,)'
365    li $v0,11
366    syscall
367    la $a0,newl        # si va a capo
368    li $v0,4
369    syscall
370    lw $ra,4($sp)      # si prende il valore di $ra precedente all'operazione effettuata
371    addi $sp,$sp,12    # e si dealloca
372    jr $ra
373 close:             li $v0,16      # syscall per la chiusura del file
374    move $a0,$t6        # scarico il descrittore del file
375    syscall
376    jr $ra
377 err:               li $v0,4
378    la $a0,fnf        # stampa della stringa 'file not found'
379    syscall
380 quit:              li $v0,10
381    syscall

```

primo.s

2 Secondo Esercizio

Scheduler di processi:

In un sistema operativo, la gestione dei task (o processi) è una questione di primaria importanza che viene spesso riferita col nome di scheduling. Per i nostri scopi, un task è definito dalle seguenti informazioni: un identificatore numerico (ID), un nome (NOME TASK, stringa che può contenere al massimo 8 caratteri), una priorità (PRIORITÀ, numero compreso fra 0 e 9) e il numero di cicli di esecuzione che richiede per essere completato (ESECUZIONI RIMANENTI). Si supponga che un task possa richiedere al più 99 cicli di esecuzione per essere completato. Lo scheduler è una funzionalità (del sistema operativo) che gestisce una coda in cui i task vengono mantenuti ordinati in base alla politica di scheduling adottata. In questo esercizio supporremo l'esistenza di due possibili politiche di scheduling:

PRIORITÀ : Si ordinano i task in modo decrescente rispetto al valore di PRIORITÀ ad essi associato (quindi da 9 a 0). Il task da eseguire per primo sarà quello a cui è stato associato il numero di PRIORITÀ più basso.

- I task con uguale valore di PRIORITÀ dovranno essere ordinati in modo decrescente rispetto al loro identificatore numerico ID.

ESECUZIONI RIMANENTI : Si ordinano i task in modo crescente rispetto al numero di ESECUZIONI RIMANENTI necessarie per il loro completamento. Il task da eseguire per primo sarà quello che necessita del maggior numero di ESECUZIONI RIMANENTI per essere completato.

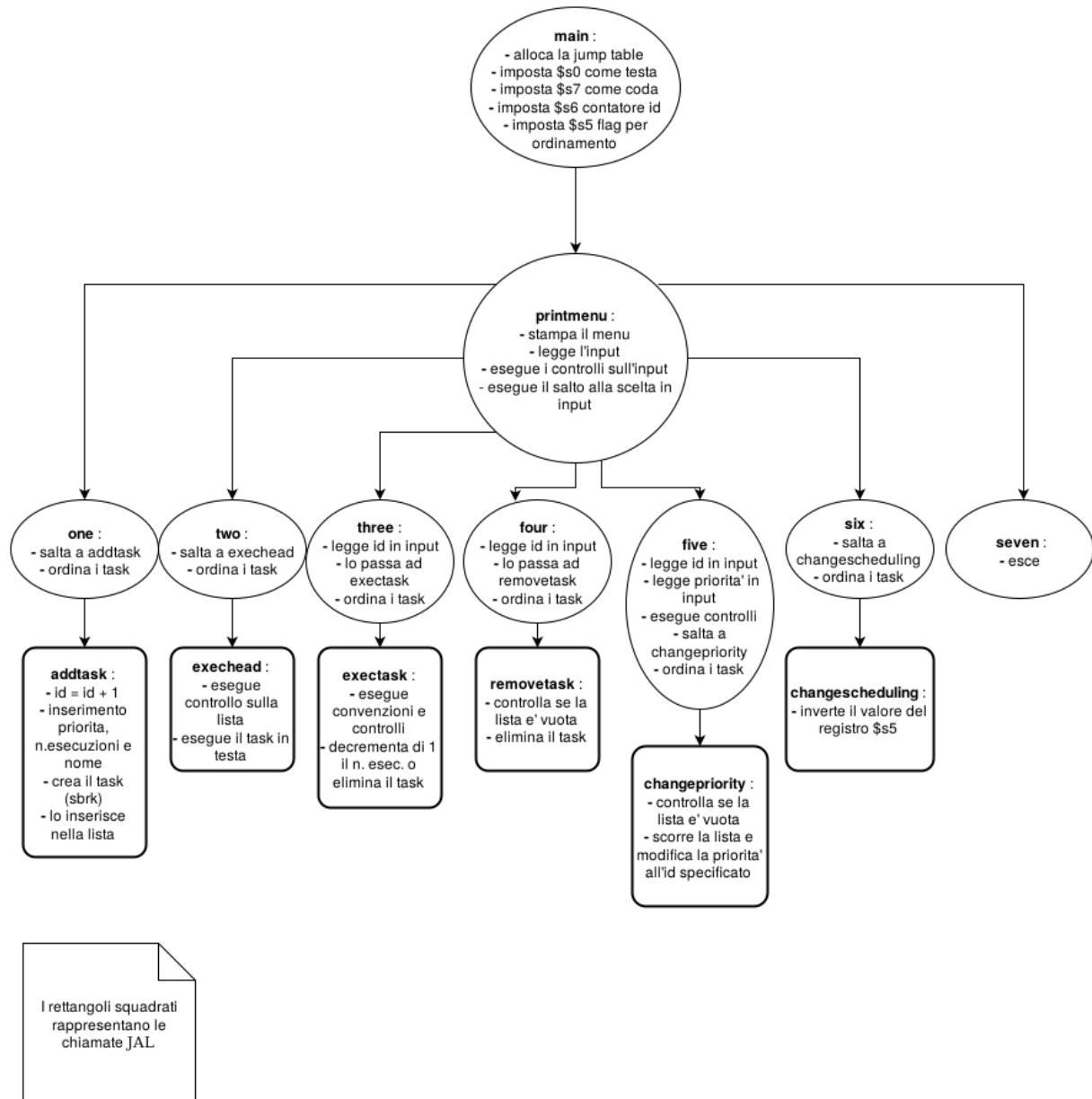
- I task con lo stesso numero di ESECUZIONI RIMANENTI dovranno essere ordinati in modo decrescente rispetto al loro identificatore numerico ID.

Utilizzando *QtSpim*, scrivere e provare un programma in assembly MIPS che simuli la funzionalità di scheduling dei task di un sistema operativo. In particolare il programma, attraverso un menu di opzioni, dovrà consentire di:

1. Inserire un nuovo task, richiedendo la sua priorità, il nome ed il numero di esecuzioni necessarie per il suo completamento (l'identificatore ID è univoco e viene assegnato automaticamente dal programma). Il task inserito dovrà poi essere posizionato nella coda a seconda della politica di scheduling utilizzata;
2. Eseguire il task che si trova in testa alla coda;
3. Eseguire il task il cui identificatore ID è specificato dall'utente;
4. Eliminare il task il cui identificatore ID è specificato dall'utente;
5. Modificare la PRIORITÀ del task il cui identificatore ID è specificato dall'utente;
6. Cambiare la politica di scheduling utilizzata, passando dalla (1) alla (2) e viceversa;
7. Uscire dal programma.

L'esecuzione di un task con ESECUZIONI RIMANENTI = 1 comporta l'eliminazione del task stesso dalla coda, altrimenti il numero di ESECUZIONI RIMANENTI viene decrementato di 1 e l'ordinamento della coda aggiornato a seconda della politica di scheduling utilizzata.

2.1 Schema logico



N.B. : In questo semplice schema non si è tenuto conto delle sottofunzioni foglia che permettono il corretto funzionamento delle "macro" procedure esposte.

2.2 Strutture dati

Per quanto *self-explaining* dal nome e, soprattutto, dal codice, riportiamo la lista delle stringhe e delle strutture dati utilizzate nella risoluzione di questo esercizio.

- **jumptable** : tabella per le varie labels
- **buffer** : buffer di 8 byte che viene utilizzato come memoria temporanea dove vengono salvati i nomi dei tasks
- **menuinput** : stringa che contiene un messaggio che viene stampato quando l'utente deve effettuare una scelta dal menu
- **err** : stringa che contiene un messaggio di errore che viene stampato quando l'utente effettua una scelta non corretta
- **perr** : stringa che contiene un messaggio di errore che viene stampato quando l'utente cerca di inserire una priorità non corretta
- **neerror** : stringa che contiene un messaggio di errore che viene stampato quando l'utente cerca di inserire un numero di esecuzioni non corretto
- **newline** contiene il carattere newline
- **space1..3** : stringhe di spazi che vengono utilizzate al momento della stampa dei tasks
- **taskmenuprint1..4** : stringhe di richiesta degli inserimenti (ID, priorità, numero esecuzioni..)
- **choice1..7** : stringhe delle 7 scelte nel menu (inserimento, esecuzione...)
- **stack** : utilizzato per memorizzare l'indirizzo di ritorno delle chiamate a procedura, come richiesto dalle convenzioni

Oltre alle stringhe appena elencate si fa anche uso di due registri temporanei preservati

- **\$s5** : flag per determinare se la lista dei task deve essere ordinata per priorità oppure per numero di esecuzioni (\$s5=0 priorità altrimenti se \$s5=1 per numero di esecuzioni)
- **\$s6** : contatore univoco che rappresenta l'ID assegnato automaticamente ad ogni task e che incrementa ad ogni accesso

2.3 Funzioni

- **main** : funzione di inizializzazione. Prepara la jump table allocandoci le varie label che corrispondono alle scelte che l'utente può compiere e, successivamente, inizializza i due registri preservati \$s0,\$s7 a zero, i quali saranno rispettivamente testa e coda della lista dei task.
- **printmenu** : funzione che stampa il menu di scelta, legge la scelta dell'utente e ne controlla la correttezza. Calcola poi l'offset corretto e salta alla label corrispondente

Tutte le seguenti funzioni si occupano, oltre a gestire ogni effettiva scelta operabile dall'utente, di ordinare la lista dei task (previa controllo del flag \$s5).

- **one** : label che esegue una jal all'effettiva procedura, la quale permette di inserire un nuovo task nella lista concatenata.
- **two** : label che esegue una jal all'effettiva procedura, la quale permette di eseguire il task che si trova in testa alla lista.
- **three** label che permette all'utente di inserire l'ID di un task che vuole eseguire.
- **four** : label che permette all'utente di inserire l'ID di un task che vuole eliminare.
- **five** : label che permette all'utente di inserire l'ID e la priorità di un task del quale vuole modificare la priorità.
- **six** : label che esegue una jal all'effettiva procedura che permette di modificare il valore del registro \$s5 e quindi di switchare politica di ordinamento.
- **seven** : label che permette di uscire dal programma

2.3.1 Descrizione delle funzioni

La procedura `addtask` per prima cosa incrementa di 1 il contenuto del registro `$s6`, ovvero l'ID del task da inserire, dopodiché richiede di inserirne la priorità, il numero di esecuzioni ed il nome (eseguendo i relativi controlli sulla correttezza degli input). Esegue poi una `syscall` per l'allocazione dinamica di un blocco di 24 bytes (nei primi 20 rispettivamente vengono riservati 4 per l'ID, 4 per la priorità, 4 per il numero di esecuzioni rimanenti ed 8 per il nome). ID, priorità e numero di esecuzioni rimanenti vengono inseriti e nelle giuste posizioni del record grazie a delle `store word`, mentre il nome viene caricato carattere per carattere con delle `store word`. L'inserimento di un nuovo task implica inoltre la verifica della presenza di altri elementi nella lista per decidere se quest'ultimo dovrà essere posto in testa alla coda dei task oppure se dovrà essere inserito in ultima posizione (ovvero negli ultimi 4 bytes riservati dinamicamente dalla chiamata `sbrk`).

La procedura `exechead` controlla innanzitutto che sia presente almeno un elemento in testa alla lista. Se la verifica ha successo legge il numero di esecuzioni rimanenti di quest'ultimo e, se questo numero è maggiore di 1 allora il numero di esecuzioni rimanenti viene semplicemente decrementato di 1, altrimenti il task viene rimosso dalla lista. Infine viene aggiornato l'elemento in testa se viene eseguita una rimozione. La procedura `exectask` riceve come argomento l'ID del processo da eseguire e controlla per prima cosa se la catena di task sia vuota o meno. Nel rispetto delle convenzioni per le chiamate a procedura viene anche allocata una posizione nello `stack` per salvare l'indirizzo di ritorno del chiamante, dal momento che questa procedura potrebbe chiamare a sua volta `removetask` nel caso in cui il task con l'ID specificato dall'utente abbia un numero di esecuzioni minore o uguale ad 1. Successivamente implementa un ciclo che viene ripetuto finché non viene trovato l'ultimo elemento della catena. In questo ciclo si confronta l'ID di ogni task con l'ID inserito dall'utente e, nel caso siano uguali, si procede al controllo sul numero di esecuzioni rimanenti. Qualora questo numero sia minore o uguale ad 1 si procede ad eliminare il task, altrimenti il numero viene solamente decrementato di una unità. La procedura `removetask` riceve come argomento l'ID del task da eliminare e controlla che effettivamente via sia almeno un elemento nella catena, in caso contrario esce dal programma. Successivamente valuta se l'elemento da eliminare è quello in testa e, nel caso, lo elimina, aggiornando l'indirizzo del primo elemento al task successivo a quello appena rimosso. Se la ricerca non ha successo viene eseguito un ciclo che termina solo quando l'elemento successivo a quello corrente è nullo, finché vi sono elementi nella lista. Viene quindi modificato il puntatore dell'elemento precedente a quello da eliminare, facendolo puntare all'elemento successivo a quello da eliminare. La procedura `changepriority` riceve come primo argomento l'ID del task del quale si vuole modificare la priorità. Viene poi confrontato l'ID in input con ogni ID presente all'interno della catena e, quando ha successo, esce dal ciclo e va a modificare la priorità del task con quella specificata dall'utente. La procedura `changepolicy` semplicemente opera sul valore del registro `$s5`, il quale se è a zero indica che i task devono essere ordinati secondo priorità altrimenti per numero di esecuzioni rimanenti. Quando invocata si occupa di invertire il valore del suddetto registro.

Per quanto riguarda le funzioni che si occupano dell'ordinamento dei task: Esse sono `prioritysort` e `remaningexecsort`. Entrambe le funzioni si basano sull'algoritmo di ordinamento `bubblesort` con flag di swap (da noi implementato in un registro temporaneo). Questo algoritmo si basa sul confronto di elementi adiacenti che vengono scambiati se non sono nella giusta posizione. Il flag di swap indica se nel ciclo sia stato effettuato almeno uno scambio; viene controllato che via sia almeno un task nella lista e viene eseguito un loop che scandisce la lista tante volte fin quando non vengono più eseguiti scambi, ovvero quando il valore del flag torna ad essere 0. Non si è tenuto conto di scelte implementative particolarmente efficienti, dal momento che lo scambio di due elementi avviene per valori effettivi contenuti nei registri, e non dei semplici puntatori (cosa che permetterebbe molti accessi in meno alla memoria). Il `bubblesort` non è di per sé un algoritmo efficiente, dato che ha una complessità dell'ordine di $O(n^2)$. In una implementazione realistica si sarebbe comunque tenuto conto di questi fattori.

2.4 Screenshots

2.4.1 Simulazioni

- Inserimento del primo task *tuno*

```
Console
1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci
Inserisci un numero da 1 a 7

1
Inserire la priorita' del task
4
Inserire il numero di esecuzioni del task
3
Inserire il nome del task
tuno
+-----+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+-----+
| 0 | 4 | tuno | 3 |
1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci
Inserisci un numero da 1 a 7
|
```

- Inserimento del secondo task *tdue*

```
Console
Inserire il nome del task
tuno
+-----+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+-----+
| 0 | 4 | tuno | 3 |
1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci
Inserisci un numero da 1 a 7

1
Inserire la priorita' del task
7
Inserire il numero di esecuzioni del task
10
Inserire il nome del task
tdue
+-----+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+-----+
| 1 | 7 | tdue | 10 |
| 0 | 4 | tuno | 3 |
1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci
Inserisci un numero da 1 a 7
```

- Inserimento del terzo task *trei*

```
Console
+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+
| 1 | 7 | tdue | 10 |
| 0 | 4 | tuno | 3 |
| 2 | 3 | ttre | 2 |
1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci
Inserisci un numero da 1 a 7

1
Inserire la priorita' del task
3
Inserire il numero di esecuzioni del task
2
Inserire il nome del task
ttre
+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+
| 1 | 7 | tdue | 10 |
| 0 | 4 | tuno | 3 |
| 2 | 3 | ttre | 2 |
1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci
Inserisci un numero da 1 a 7
```

- Esecuzione del task in testa (Il numero di esecuzioni di *tuno* viene decrementato di uno)

```
Console
3
Inserire il numero di esecuzioni del task
2
Inserire il nome del task
ttre
+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+
| 1 | 7 | tdue | 10 |
| 0 | 4 | tuno | 3 |
| 2 | 3 | ttre | 2 |
1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci
Inserisci un numero da 1 a 7

2
+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+
| 1 | 7 | tdue | 9 |
| 0 | 4 | tuno | 3 |
| 2 | 3 | ttre | 2 |
1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci
Inserisci un numero da 1 a 7
```

- Esecuzione del task con ID = 2 (Il numero di esecuzioni di *trei* viene decrementato di uno)

```
Console
Inserisci un numero da 1 a 7

2
+-----+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+-----+
| 1 | 7 | tdue | 9 |
| 0 | 4 | tuno | 3 |
| 2 | 3 | tre | 2 |
1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci
Inserisci un numero da 1 a 7

3
Inserire ID del task
2
+-----+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+-----+
| 1 | 7 | tdue | 9 |
| 0 | 4 | tuno | 3 |
| 2 | 3 | tre | 1 |
1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci
Inserisci un numero da 1 a 7
```

- Eliminazione del task con ID = 0 (Il task *tuno* viene rimosso dalla lista)

```
Console
Inserisci un numero da 1 a 7

3
Inserire ID del task
2
+-----+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+-----+
| 1 | 7 | tdue | 9 |
| 0 | 4 | tuno | 3 |
| 2 | 3 | tre | 1 |
1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci
Inserisci un numero da 1 a 7

4
Inserire ID del task
0
+-----+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+-----+
| 1 | 7 | tdue | 9 |
| 2 | 3 | tre | 1 |
1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci
Inserisci un numero da 1 a 7
```

- Viene modificata la priorità del task con ID = 2 (La priorità di *ttre* passa da 3 ad 8, salendo quindi in testa alla lista)

```

4
Inserire ID del task
0
+-----+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+-----+
| 1 | 7 | tdue | 9 |
| 2 | 3 | ttre | 1 |

1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci

Inserisci un numero da 1 a 7

5
Inserire ID del task
2
Inserire la priorita' del task
8
+-----+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+-----+
| 2 | 8 | ttre | 1 |
| 1 | 7 | tdue | 9 |

1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci

Inserisci un numero da 1 a 7

```

- Cambio della politica di scheduling e conseguente switch tra i due processi (da PRIORITÀ ad ESECUZIONI RIMANENTI)

```

Inserisci un numero da 1 a 7

5
Inserire ID del task
2
Inserire la priorita' del task
3
+-----+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+-----+
| 1 | 7 | tdue | 9 |
| 2 | 3 | ttre | 1 |

1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci

Inserisci un numero da 1 a 7

6
+-----+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI |
+-----+-----+-----+
| 2 | 3 | ttre | 1 |
| 1 | 7 | tdue | 9 |

1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare la politica di scheduling (default PRIORITA')
7) Esci

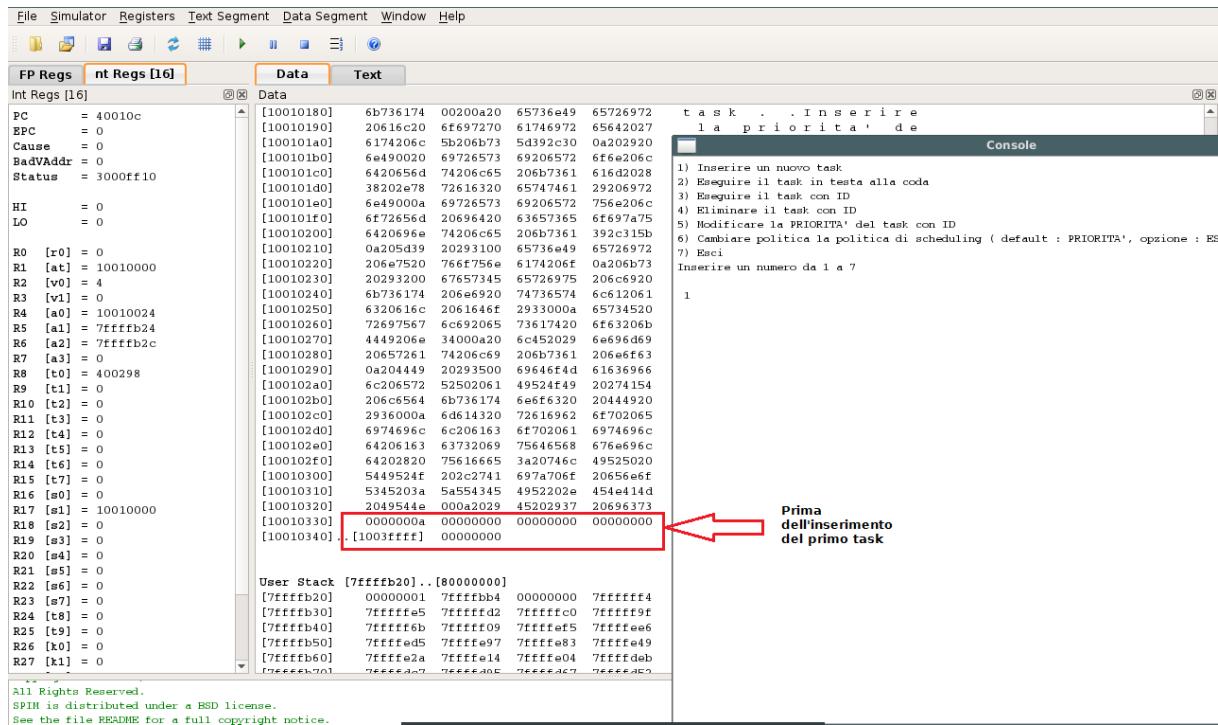
Inserisci un numero da 1 a 7

```

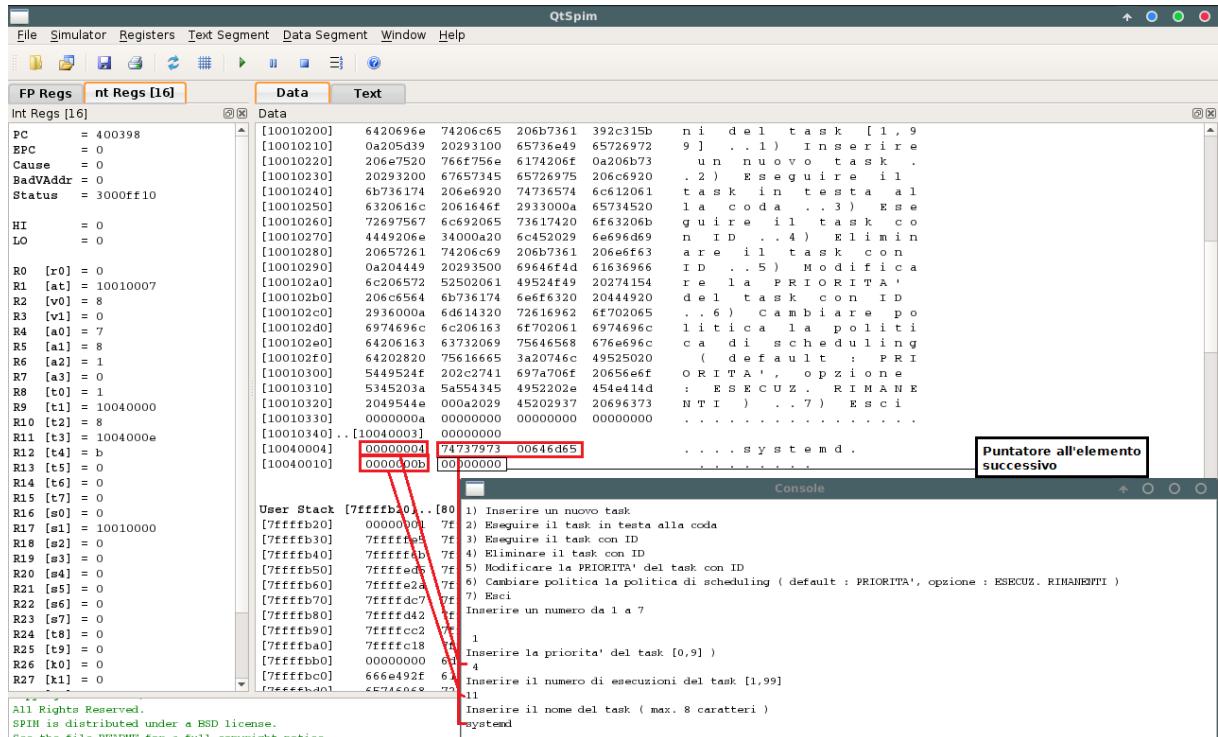
2.4.2 Gestione della Memoria

Mostriamo qui l'andamento della memoria dinamica (*heap*) in alcuni casi d'interesse, ovvero l'inserimento di due tasks, l'ordinamento e l'esecuzione di uno di essi.

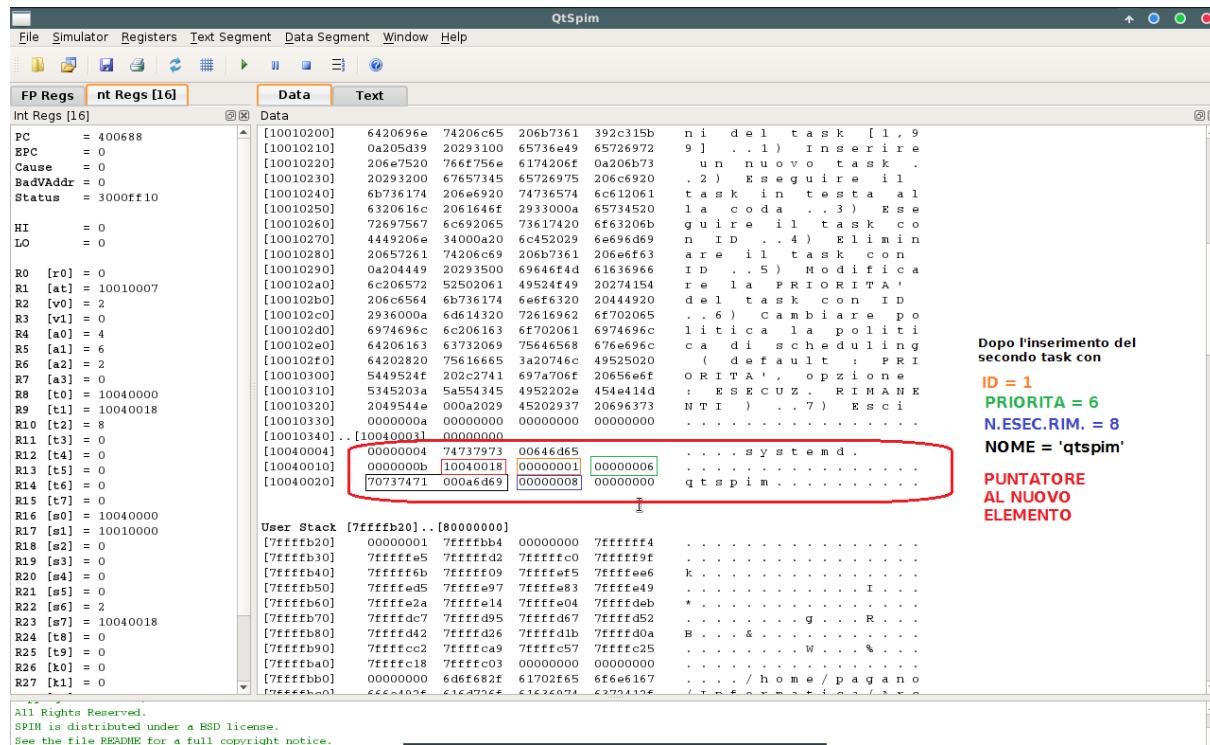
Inizializzazione dei 24 bytes con la chiamata `sbrk`



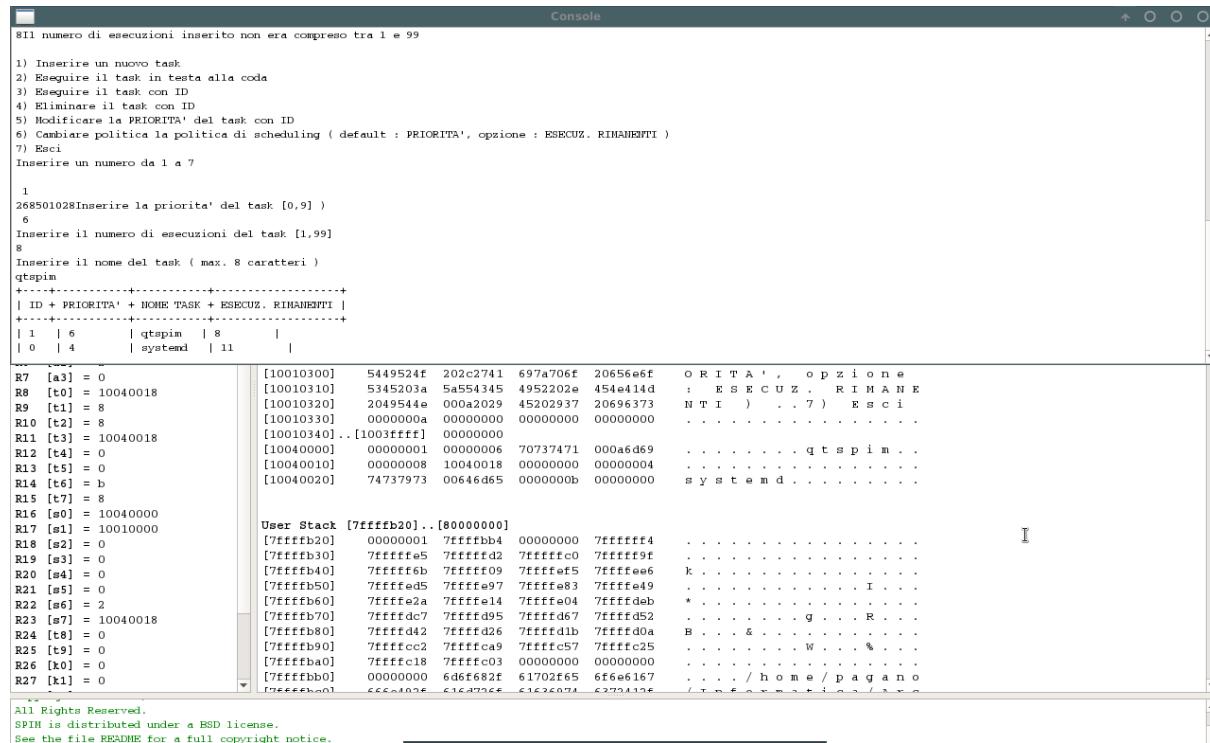
- Dopo l'inserimento del primo task con ID = 0, prioritá = 4, numero di esecuzioni (rimanenti) = 11 e nome = 'systemd'



- Dopo l'inserimento del secondo task con ID = 1, prioritá = 6, numero di esecuzioni (rimanenti) = 8 e nome = 'qtspim'



- Dopo l'ordinamento dei due tasks per prioritá (default)



- Dopo l'esecuzione del task in testa ('qtspim' modifica il numero di esecuzioni da 8 a 7.

The screenshot shows the Win32 API Simulator interface. The top menu bar includes File, Simulator, Registers, Text Segment, Data Segment, Window, Help. The registers window shows FP Regs and Int Regs [16]. The assembly window displays assembly code with labels like .Data and .Text. The memory dump window shows memory starting at address 10010200. The bottom right corner features a terminal window titled "Console" with text related to task scheduling.

```

File Simulator Registers Text Segment Data Segment Window Help
FP Regs Int Regs [16] Data Text
Int Regs [16]
PC = 400614
EPC = 0
Cause = 0
BadAddr = 0
Status = 3000ff10
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 10010000
R2 [v0] = 4
R3 [v1] = 0
R4 [a0] = 100100f0
R5 [a1] = 4
R6 [a2] = 2
R7 [a3] = 0
R8 [t0] = 10040018
R9 [t1] = 8
R10 [t2] = 8
R11 [t3] = 10040018
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = b
R15 [t7] = 8
R16 [s0] = 10040000
R17 [s1] = 10010000
R18 [s2] = 0
R19 [s3] = 0
R20 [a4] = 0
R21 [a5] = 0
R22 [a6] = 2
R23 [s7] = 10040018
R24 [t8] = 0
R25 [t9] = 0
R26 [x0] = 0
R27 [x1] = 0
[10010200] 6420696e 74206c65 206b7361 392c315b
[10010210] 0a205d39 20293100 65736e49 65726972
[10010220] 206e7520 676f756e 6174206f 0a206b73
[10010230] 20293200 67657345 65726975 206c6920
[10010240] 61736174 206e6920 74736574 6c612061
[10010250] 6320616c 20616464 2933000a 65734520
[10010260] 72697567 6692065 73617420 6f63206b
[10010270] 444920e6 30004020 6c452029 6e696d89
[10010280] 20657261 74206c69 206b7361 206e6f63
[10010290] 0a204449 20293500 69646f4d 61636966
[100102a0] 6c206572 52502061 49524f49 20274154
[100102b0] 206c6564 6b736174 66e6f320 20449420
[100102c0] 2936000a 64614320 72616962 6f702065
[100102d0] 6974696c 62206163 6f702061 6974696e
[100102e0] 64206163 63732065 75645658 676e696c
[100102f0] 64202820 75616665 3a20746c 49525020
[10010300] 5449524f 202c2741 69770476 20656f6f
[10010310] 5345203a 5a554345 4952202e 4544414d
[10010320] 2049544e 0000a209 45202937 20596373
[10010330] 00000000 00000000 00000000 00000000
[10010340] .. [1003ffff] 00000000
[10040000] 00000001 00000006 70737471 000a6d69
[10040010] 00000007 10040018 00000000 00000004
[10040020] 74737973 00646d65 0000000b 00000000
User Stack [7fffffb20]..[80000000]
[7ffffb20] 00000001 7fffffb4 00000000 7fffff4
[7ffffb30] 7fffffe5 7fffffd2 7fffffc0 7fffff9
[7ffffb40] 7fffffb6 7fffff09 7fffff45 7fffffe6
[7ffffb50] 7fffffd5 7fffffe9 7fffff83 7fffff49
[7ffffb60] 7fffffe2a 7fffff14 7fffffe04 7fffffdb
[7ffffb70] 7ffffdc7 7fffffd45 7fffffd67 7fffffd52
[7ffffb80] 7fffffd42 7fffffd57 7fffffd1b 7fffffd0a
[7ffffb90] 7fffffc2 7fffffc9a 7fffffc57 7fffffc25
[7ffffba0] 7fffffc18 7fffffc0 00000000 00000000
[7ffffbb0] 00000000 606f682f 61702615 6f6e6167
[7ffffbb1] cccca07e c1c277ce c1e2e07a c277a17e

```

Inserire il nome del task (max. 8 caratteri)
qtspim
+-----+-----+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIHAEMENTI |
+-----+-----+-----+-----+
| 1 | 6 | qtspim | 8 |
| 0 | 4 | systemd | 11 |
1) Inserire un nuovo task
2) Eseguire il task in testa alla coda
3) Eseguire il task con ID
4) Eliminare il task con ID
5) Modificare la PRIORITA' del task con ID
6) Cambiare politica la politica di scheduling (default : PRIORITA', opzione : E
7) Esci
Inserire un numero da 1 a 7
2/
+-----+-----+-----+-----+
| ID + PRIORITA' + NOME TASK + ESECUZ. RIHAEMENTI |
+-----+-----+-----+-----+
| 1 | 6 | qtspim | 7 |
| 0 | 4 | systemd | 11 |
* g . . . R . .
B . . . & . . . M . . . % . .
. / h o m e / p a g a n o
/ t h e r e a s o n s / s a v e

2.5 Codice

```

1 # Bindi Giovanni 5530804 giovanni.bindi@stud.unifi.it
2 # Puliti Gabriele 5300140 gabriele.puliti@stud.unifi.it
3 # Lippi Lorenzo 6221250 lorenzo.lippi@stud.unifi.it
4 # data_consegna
5 .data
6 jump_table: .space 28 #tabella per 7 scelte
7 buffer: .space 8
8 menuinput: .asciiz "Inserire un numero da 1 a 7\n\n"
9 err: .asciiz "Il numero inserito non era compreso tra 1 e 7 \n\n"
10 perror: .asciiz "La priorita' inserita non era compresa tra 0 e 9 \n\n"
11 enerror: .asciiz "Il numero di esecuzioni inserito non era compreso tra 1 e 99 \n\n"
12 newline: .asciiz "\n"
13 space1: .asciiz " "
14 space2: .asciiz " "
15 space3: .asciiz " "
16 endstring: .asciiz "Quitting. \n\n"
17 printmenubar: .asciiz "+-----+-----+-----+ \n"
18 printmenulabels: .asciiz "| ID + PRIORITA' + NOME TASK + ESECUZ. RIMANENTI | \n"
19 taskmenuprint1: .asciiz "Inserire ID del task \n"
20 taskmenuprint2: .asciiz "Inserire la priorita' del task [0,9] ) \n"
21 taskmenuprint3: .asciiz "Inserire il nome del task ( max. 8 caratteri )\n"
22 taskmenuprint4: .asciiz "Inserire il numero di esecuzioni del task [1,99] \n"
23 choice1: .asciiz "1) Inserire un nuovo task \n"
24 choice2: .asciiz "2) Eseguire il task in testa alla coda \n"
25 choice3: .asciiz "3) Eseguire il task con ID \n"
26 choice4: .asciiz "4) Eliminare il task con ID \n"
27 choice5: .asciiz "5) Modificare la PRIORITA' del task con ID \n"
28 choice6: .asciiz "6) Cambiare politica la politica di scheduling ( default : PRIORITA'
', opzione : ESECUZ. RIMANENTI ) \n"
29 choice7: .asciiz "7) Esci \n"
30 .text
31 .globl main
32 main:
33 la $s1, jump_table      # preparo la jump table (allocandoci le varie label)
34 la $t0, one
35 sw $t0, 0($s1)
36 la $t0, two
37 sw $t0, 4($s1)
38 la $t0, three
39 sw $t0, 8($s1)
40 la $t0, four
41 sw $t0, 12($s1)
42 la $t0, five
43 sw $t0, 16($s1)
44 la $t0, six
45 sw $t0, 20($s1)
46 la $t0, seven
47 sw $t0, 24($s1)
48 move $s0,$zero           # indice di testa, settato di default a 0
49 move $s7,$zero           # indice di coda, settato di default a 0
50 move $s6,$zero           # registro che tiene il conto del numero di task create
51 move $s5,$zero           # s5=0 ordina per priorita s5=1 ordina esec. rimanenti
52 printmenu:                # stampo le stringhe relative al menu di scelta
53 la $a0,choice1
54 li $v0,4
55 syscall
56 la $a0,choice2
57 li $v0,4
58 syscall
59 la $a0,choice3
60 li $v0,4
61 syscall
62 la $a0,choice4
63 li $v0,4
64 syscall
65 la $a0,choice5
66 li $v0,4
67 syscall
68 la $a0,choice6
69 li $v0,4
70 syscall

```

```

71    la $a0,choice7
72    li $v0,4
73    syscall
74    la $a0, menuinput      # stampa la richiesta di inserimento di un numero compreso
75    tra 1 e 7
76    li $v0, 4
77    syscall
78        li $v0, 5          # legge la scelta
79    syscall
80    move $t2, $v0          # salvo l'intero letto in $t2
81    sle $t0, $t2, $zero
82    bne $t0, $zero, choice_err    # errore se la scelta <=0
83    li $t0,7
84    sle $t0, $t2, $t0
85    beq $t0, $zero, choice_err    # errore se la scelta >=7
86    addi $t2, $t2, -1      # tolgo 1 da scelta perche' prima azione nella jump table (in
87    posizione 0) corrisponde alla 1 scelta del case
88    add $t0, $t2, $t2
89    add $t0, $t0, $t0      # preparo l'offset della loadword per il salto alla scelta
90    fatta
91    add $t0, $t0, $s1      # sommo all'indirizzo del primo case l'offset appena
92    calcolato
93    lw $t0, 0($t0)        # $t0 = indirizzo a cui devo saltare
94    jr $t0                  # salto all'indirizzo calcolato
95 one:
96    move $a0,$s6
97    jal addtask
98    move $s6,$v0
99    beqz $s5,prioritysort    # se s5=0 ordina con priorita altrimenti per
100   j remainingexecsort    # esecuzioni rimanenti
101 two:
102   move $a0,$s0
103   jal execfront
104   move $s0,$v0
105   beqz $s5,prioritysort    #se s5=0 ordina con priorita altrimenti per
106   j remainingexecsort    #esecuzioni rimanenti
107 three:
108   la $a0,taskmenuprint1    # richiedo di inserire l'id del task
109   li $v0,4
110   syscall
111   li $v0,5          # leggo l'id del task
112   syscall
113   move $a0,$v0          # e lo salvo in a0
114   move $a1,$s0
115   move $a2,$s7
116   jal exectask
117   move $s0,$v0
118   move $s7,$v1
119   beqz $s5,prioritysort    #se s5=0 ordina con priorita altrimenti per
120   j remainingexecsort    #esecuzioni rimanenti
121 four:
122   la $a0,taskmenuprint1    # richiedo di inserire l'id del task
123   li $v0,4
124   syscall
125   li $v0,5          # leggo l'id del task
126   syscall
127   move $a0,$v0          # lo salvo in a0
128   move $a1,$s0
129   move $a2,$s7
130   jal removetask
131   move $s0,$v0
132   move $s7,$v1
133   beqz $s5,prioritysort    # se s5=0 ordina con priorita altrimenti per
134   j remainingexecsort    # esecuzioni rimanenti
135   j printmenu
136 five:
137   la $a0,taskmenuprint1    # richiedo di inserire l'id del task
138   li $v0,4
139   syscall
140   li $v0,5          # leggo l'id del task
141   syscall
142   move $t1,$v0          # lo salvo in t1 per poi salvarlo in a0 in seguito

```

```

140 la $a0,taskmenuprint2      # richiedo di inserire la priorita' del task
141 li $v0,4
142 syscall
143 li $v0,5
144 syscall          # leggo la priorita'
145             # controllo sulla priorita
146 li $t0,-1
147 sle $t0, $v0, $t0
148 bnez $t0, priority_err    # errore se la priorita' <=-1
149 li $t0,9
150 sle $t0, $v0, $t0
151 beqz $t0, priority_err    # errore se la priorita' >=9
152 move $a0,$t1                # sposto l'id nel registro a0 per passarlo come argomento
153 move $a1,$v0                # sposto le esecuzioni rimanenti nel registro a1 per passarlo
154             come argomento
155 move $a2,$s0
156 jal changepriority
157 beqz $s5,prioritysort      # se s5=0 ordina con priorita' altrimenti per
158 j remainingexecsort        # esecuzioni rimanenti
159 j printmenu
160 six:
161 move $a0,$s5
162 jal changescheduling
163 move $s5,$v0
164 beqz $s5,prioritysort      # se s5=0 ordina con priorita' altrimenti per
165 j remainingexecsort        # esecuzioni rimanenti
166 j printmenu
167 seven:
168 j quit
169 addtask:
170 move $a2,$a0
171 addi $a2,$a2,1              # aggiungo 1 al contatore degli id (al primo inserimento id = 0)
172 li $v0, 4
173 la $a0, taskmenuprint2      # richiedo di inserire la priorita' del task
174 syscall
175 li $v0, 5
176 syscall          # leggo la priorita'
177 move $t3,$v0                # salvo l'intero letto in t3 per eseguire i controlli
178             # controllo sulla priorita
179 li $t0,-1
180 sle $t0, $t3, $t0          # setto t0 a 1 se la priorita' <=-1
181 bnez $t0, priority_err    # errore se la priorita' <=-1
182 li $t0,9
183 sle $t0, $t3, $t0          # setto t0 a 1 se la priorita' <= 0
184 beqz $t0, priority_err    # errore se la priorita' >9
185 li $v0, 4
186 la $a0, taskmenuprint4      # richiedo di inserire il numero di esecuzioni
187 syscall
188 li $v0, 5
189 syscall          # leggo il numero di esecuzioni
190 move $t4,$v0                # sposto il numero di esecuzioni in t4 per eseguire i controlli
191             # controllo sul numero di esecuzioni
192 li $t0,0
193 sle $t0, $t4, $t0          # t0 = 1 se num.esec. <= 0
194 bnez $t0, execnumb_err      # errore se num.esec. <=0
195 li $t0,99
196 sle $t0, $t4, $t0          # t0 = 1 se num.esec. <= 99
197 beqz $t0, execnumb_err      # errore se num.esec >=99
198             # inizio inserzione nuovo elemento-----00
199 li $v0, 9
200 li $a0, 24
201 syscall          # chiamata sbrk: alloco dinamicamente 24 bytes (salvo l'indirizzo di allocazione in v0)
202 sw $s6, 0($v0)            # salvo nella prima parola il valore dell'id del task
203 sw $t3, 4($v0)            # nella seconda la priorita
204 sw $t4, 16($v0)           # nella quinta parola il numero di esecuzioni
205 move $t3,$v0                # sposto l'indirizzo di memoria in t3 (il nodo diventa t3)
206 li $v0, 4
207 la $a0, taskmenuprint3      # richiedo l'inserimento del nome del task
208 syscall
209 li $v0,8

```

```

209 la $a0,buffer           # carico l'indirizzo del buffer
210 li $a1,8                 # numero massimo di caratteri da leggere (8)
211 syscall
212 move $t1,$t3             # inserisco un carattere alla volta
213     # salvo in t1 l'indirizzo della testa del record dei campi
214     # del task
215     li $a0,0               # imposto l'indice sul buffer a 0
216     addi $t3,$t3,7          # offset
217     readname:              # loop per leggere i caratteri inseriti
218         lb $a3,buffer($a0)  # leggo il carattere (all'inizio con indice 0)
219         addi $a0,$a0,1        # incremento l'indice di 1
220         li $t2,8
221         beq $a0,$t2,readname2 # quando l'indice=8 salta a readname2
222         addi $t3,$t3,1          # calcolo l'offset
223         sb $a3,0($t3)          # inserisco il carattere nel nodo
224         bne $a3,readname       # finche l'indice e' diverso da 0 continua
225     readname2:              # sposto di nuovo l'indirizzo di memoria allocata
226         move $t3,$t1            # dinamicamente in t3
227         sw $zero,20($t3)          # imposto elemento successivo (il quarto) = 0
228         bne $s0,$zero,addlast    # se s0!=0 (coda non vuota) vai a addlast
229             # altrimenti (prima inserzione) restituisco il nodo
230             # cosi con puntatore next nullo
231         move $s0,$t3
232         move $s7,$t3
233     point:                  # se la coda e' non vuota
234         move $v0,$a2
235         jr $ra
236     addlast:                # salvo il contenuto di t3 nell'indirizzo di memoria
237         sw $t3,20($s7)
238         contenuto in s7+20
239         move $s7,$t3
240         j point
241     removetask:              # se la coda e' vuota
242         beqz $a1,point3
243         move $t0,$a1            # t0 = testa della coda
244         lw $t3,0($t0)           # leggo id
245         beq $t3,$a0,removefront # se il nodo da eliminare non e la testa
246         lw $t1,20($t0)           # elemento successivo al corrente
247     removeloop:              # se il nodo successivo e' zero ho finito
248         beqz $t1,point3
249         beq $t1,$a2,removerear # se il nodo successivo e' la coda (la fine)
250             # altrimenti
251         lw $t3,0($t1)           # leggo id del successivo
252         bne $t3,$a0,nextone     # se e' diverso da a0
253         lw $t4,20($t1)           # leggo il successivo del successivo
254         sw $t4,20($t0)
255     point3:                  # se la coda e' vuota
256         move $v0,$a1
257         move $v1,$t0
258         jr $ra
259     removefront:             # se la coda e' non vuota
260         lw $a1,20($a1)
261         j point3
262     removerear:              # se la coda e' vuota
263         sw $zero,20($t0)
264         j point3
265     nextone:                 # leggo l'elemento successivo
266         lw $t0,20($t0)
267         lw $t1,20($t0)           # aggiorno il successivo
268         j removeloop
269     exectask:                # essendo una chiamata a procedura devo preservare il valore
270         addi $sp,$sp,-4
271         di $ra
272         sw $ra,0($sp)
273         move $t0,$a1
274     execloop:                # e lo carico nello stack
275         beqz $t0,point4
276         lw $t1,0($t0)           # altrimenti metto in t0 la testa
277             # ed inizio il loop
278             # se la coda e' vuota vado a deallocare lo stack
279             # leggo l'id
280             # se i due id non sono uguali passa al successivo
281             # leggo esecuzioni rimanenti

```

```

277    slti $t4,$t2,2          # setto t4=1 se il numero di esecuzioni rimanenti e' minore
278    di 2
279    beqz $t4,execdecrease   # se t4=0 vuol dire che t2>2 quindi posso
280    decrementare di 1
281    # altrimenti t2<=1
282    move $a0,$t1             # passo l'id come argomento
283    jal removetask           # e quindi posso andare ad eliminare il task
284    move $a1,$v0
285    move $a2,$v1
286    lw $ra,0($sp)            # convenzione su $ra
287 point4:
288    move $v0,$a1
289    move $v1,$a2
290    addi $sp,$sp,4           # dealloco dallo stack (convenzione per $ra il cui valore e'
291    stato
292    jr $ra                  # modificato in questa procedura
293 nexttask:
294    lw $t0,20($t0)           # leggo l'elemento successivo
295    j execloop               # e ripeto il procedimento
296 execdecrease:
297    addi $t2,$t2,-1          # decremento di 1 le esecuzioni rimanenti
298    sw $t2,16($t0)           # ed inserisco il valore aggiornato nello stack
299    j point4
300 changepriority:
301    move $t0,$a2              # sposto in t0 la testa
302 changeloop:
303    beqz $t0,point5           # ho finito (ho raggiunto la endstring della catena)
304    lw $t1,0($t0)              # leggo id
305    bne $a0,$t1,nextelement   # se i due id non sono uguali passa al successivo
306    # altrimenti modifico priorita
307    sw $a1,4($t0)
308 point5:
309    jr $ra
310 nextelement:
311    lw $t0,20($t0)           # leggo l'elemento successivo
312    j changeloop
313 changescheduling:
314    beqz $a0,addone            # se s5= allora diventa 1 quindi con esecuzioni
315    rimanenti
316    addi $a0,$a0,-1           # altrimenti diventa 0
317 pointrr:
318    move $v0,$a0
319    jr $ra
320 addone:
321    addi $a0,$a0,1
322    j pointrr
323 print:                      # loop di stampa
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344

```

```

320    la $a0,printmenubar
321    li $v0,4
322    syscall
323    la $a0,printmenulabels
324    li $v0,4
325    syscall
326    la $a0,printmenubar
327    li $v0,4
328    syscall
329    move $t0,$s0               # t0 = testa. t0 verrà usato come puntatore per scorrere
330 printloop:
331    beqz $t0,printmenu
332    li $a0,'|'
333    li $v0,11
334    syscall
335    la $a0,space1
336    li $v0,4
337    syscall
338    lw $a0,0($t0)              # id
339    li $v0,1
340    syscall
341    la $a0,space2
342    li $v0,4
343    syscall
344    li $a0,'|'

```

```

345      li $v0,11
346      syscall
347      la $a0, space1
348      li $v0, 4
349      syscall
350      lw $a0, 4($t0)      # priorita
351      li $v0, 1
352      syscall
353      la $a0, space3
354      li $v0, 4
355      syscall
356      li $a0, '|'
357      li $v0,11
358      syscall
359      la $a0, space1
360      li $v0, 4
361      syscall
362      #—————stampo il nome un carattere alla volta—————
363      move $t3,$t0
364      addi $t0,$t0,7      # posizione corretta del nodo-1 (1 byte prima del terzo
365      elemento del nodo)
366      li $t1,0
367      printnameloop:
368      addi $t0,$t0,1      # calcolo l'offset
369      lb $a0, 0($t0)      # leggo il carattere nel nodo
370      addi $t1,$t1,1
371      li $t2,8
372      beq $t1,$t2,finalprint
373      li $t2,97            # in ASCII il carattere 'a' corrisponde al valore 97
374      slt $t4,$a0,$t2      #
375      bnez $t4,finalprint # se il valore letto e' una lettera continua a leggere
376      li $t2,122           # in ASCII il carattere 'z' corrisponde al valore 122
377      slt $t4,$t2,$a0      # 'a' < t4 < 'z'
378      bnez $t4,finalprint # e quindi eseguo il controllo su tutto l'alfabeto
379
380      li $v0, 11          # e stampo il carattere
381      syscall
382      j printnameloop
383      finalprint:
384      move $t0,$t3
385      la $a0, space2      # stampa degli spazi
386      li $v0, 4
387      syscall
388      li $a0, '|'
389      li $v0,11
390      syscall
391      la $a0, space1
392      li $v0, 4
393      syscall
394      lw $a0, 16($t0)     # delle esecuzioni rimanenti
395      li $v0, 1
396      syscall
397      la $a0, space3
398      li $v0, 4
399      syscall
400      li $a0, '|'
401      li $v0,11
402      syscall
403      la $a0, newline      # a capo
404      li $v0, 4
405      syscall
406      lw $t0, 20($t0)     # fine della stampa
407
408      execfront:
409      beqz $a0,point2      # se la coda e vuota
410      lw $t0,16($a0)        # leggo le esecuzioni rimanenti
411      li $t1,1
412      beq $t0,$t1,removefirst # quando rimane 1 esecuzione elimino il task in testa
413      addi $t0,$t0,-1       # altrimenti decrementa
414      sw $t0,16($a0)        # e salva il valore aggiornato
415      point2:
416      move $v0,$a0          # salto all'indirizzo del task successivo (vedi sotto)

```

```

416    jr $ra
417 removefirst:
418    lw $t5,20($a0)
419    move $a0,$t5
420    j point2
421    #—————ordinamento per priorita—————
422 prioritiesort:
423    li $t3,0          # flag per lo swap
424    beqz $s0,printmenu # se la lista e vuota esci
425    move $t0,$s0        # copia in t0 l'indirizzo della testa
426    lw $t1,20($t0)      # t1 diventa il successivo della testa
427 loop:
428    beqz $t1, check      # quando e' uguale a zero ho finito la scansione della
429    lista e verifico
430    lw $a0,4($t0)        # leggo la priorita
431    lw $a1,4($t1)        # leggo la priorita del successivo
432    slt $t4,$a1,$a0
433    bnez $t4,else        # se t4=0 vuole dire che la priorita' del successivo <
434    priorita' attuale quindi incremento il puntatore
435    beq $a0,$a1,idsort   # se le priorita sono uguali ordino per id
436    # se sono diverse
437    lw $t6,0($t0)        # swappo id
438    lw $t7,0($t1)
439    sw $t6,0($t1)
440    sw $t7,0($t0)
441    lw $t6,4($t0)        # swappo priorita'
442    lw $t7,4($t1)
443    sw $t6,4($t1)
444    sw $t7,4($t0)
445    lw $t6,8($t0)        # swappo nome
446    lw $t7,8($t1)
447    sw $t6,8($t1)
448    sw $t7,8($t0)
449    lw $t6,12($t0)
450    lw $t7,12($t1)
451    sw $t6,12($t1)
452    sw $t7,12($t0)
453    lw $t6,16($t0)       # swappo numero esec. rimanenti
454    lw $t7,16($t1)
455 update:
456    addi $t3,$t3,1        # flag di swap =1 (c'e' stato uno scambio)
457    lw $t0,20($t0)        # leggo l'elemento successivo
458    lw $t1,20($t0)        # aggiorno il successivo
459    j loop
460
461 idsort:
462    lw $a0,0($t0)        # leggo l'id
463    lw $a1,0($t1)        # leggo l'id del successivo
464    slt $t4,$a1,$a0
465    bnez $t4,$zero,else   # swappo id
466    lw $t6,0($t0)
467    lw $t7,0($t1)
468    sw $t6,0($t1)
469    sw $t7,0($t0)
470    lw $t6,4($t0)        # swappo priorita'
471    lw $t7,4($t1)
472    sw $t6,4($t1)
473    sw $t7,4($t0)
474    lw $t6,8($t0)        # swappo nome
475    lw $t7,8($t1)
476    sw $t6,8($t1)
477    sw $t7,8($t0)
478    lw $t6,12($t0)
479    lw $t7,12($t1)
480    sw $t6,12($t1)
481    sw $t7,12($t0)
482    lw $t6,16($t0)       # swappo numero esec. rimanenti
483    lw $t7,16($t1)
484    sw $t6,16($t1)
485    sw $t7,16($t0)
486 j update

```

```

487 check:          # verifico se flag di swap = 1
488     bnez $t3,prioritysort      # se e' cosi' continuo il ciclo altrimenti ho finito
489     j print                   # e vado alla stampa
490 remaningexecsort:           #—————ordinamento per esecuzioni rimanenti
491     li $t3,0                  # flag per lo swap di due task
492     beqz $s0,printmenu        # se la lista e' vuota stampa il menu di scelta
493     move $t0,$s0              # sposto in t0 la testa
494     lw $t1,20($t0)            # ed in t1 il successivo della testa
495 loop2:
496     beqz $t1,check2          # quando il successivo e' zero ho finito la scansione della
497     lista e verifico
498     lw $a0,16($t0)            # leggo le esecuzioni rimanenti
499     lw $a1,16($t1)            # leggo le esecuzioni rimanenti del successivo
500     slt $t4,$a0,$a1
501     bne $t4,$zero,else2      # se t4=0 vuole dire che n.esec.rim del success. < n.e.rim.
502     attuali quindi incremento il puntatore
503     beq $a0,$a1,idsort2      # se le priorita' sono uguali ordino per id
504     # se sono diverse
505     lw $t6,0($t0)             # swappo id
506     lw $t7,0($t1)
507     sw $t6,0($t1)
508     sw $t7,0($t0)
509     lw $t6,4($t0)             # swappo priorita
510     lw $t7,4($t1)
511     sw $t6,4($t1)
512     sw $t7,4($t0)
513     lw $t6,8($t0)             # swappo nome
514     lw $t7,8($t1)
515     sw $t6,8($t1)
516     sw $t7,8($t0)
517     lw $t6,12($t0)
518     lw $t7,12($t1)
519     sw $t6,12($t1)
520     sw $t7,12($t0)
521     lw $t6,16($t0)            # swappo numero esec. rimanenti
522     lw $t7,16($t1)
523     sw $t6,16($t1)
524     sw $t7,16($t0)
525 update2:
526     addi $t3,$t3,1             # flag di swap = 1 (c'e' stato uno scambio)
527     lw $t0,20($t0)             # leggo l'elemento successivo
528     lw $t1,20($t0)             # aggiorno il successivo
529     j loop2
530 idsort2:
531     lw $a0,0($t0)             # leggo l'id
532     lw $a1,0($t1)             # leggo l'id del successivo
533     slt $t4,$a1,$a0
534     bnez $t4,else2
535     lw $t6,0($t0)             # swappo id
536     lw $t7,0($t1)
537     sw $t6,0($t1)
538     sw $t7,0($t0)
539     lw $t6,4($t0)             # swappo priorita
540     lw $t7,4($t1)
541     sw $t6,4($t1)
542     sw $t7,4($t0)
543     lw $t6,8($t0)             # swappo nome
544     lw $t7,8($t1)
545     sw $t6,8($t1)
546     sw $t7,8($t0)
547     lw $t6,12($t0)
548     lw $t7,12($t1)
549     sw $t6,12($t1)
550     sw $t7,12($t0)
551     lw $t6,16($t0)            # # swappo numero esec. rimanenti
552     lw $t7,16($t1)
553     sw $t6,16($t1)
554     sw $t7,16($t0)
555     j update2
556 check2:          # verifico se il flag di swap = 1
557     bne $t3,$zero,remaningexecsort # se e' cosi' continuo il ciclo altrimenti ho finito
558     j print

```

```

557 choice_err:                                # stampa della stringa la stringa di errore
558     li $v0, 4
559     la $a0, err
560     syscall
561     j printmenu                            # ritorna alla richiesta di inserimento di un numero tra 1 e
562     7
563 priority_err:                               # stampa la stringa di errore per priorita'
564     li $v0, 4
565     la $a0, perror
566     syscall
567 execnumb_err:                               # stampa la stringa di errore per esecuzioni rimanenti
568     li $v0, 4
569     la $a0, enerror
570     syscall
571     j printmenu                            # ritorna alla richiesta di inserimento di un numero tra 1 e
572     7
573 quit:                                       # stampa messaggio di uscita e esce
574     li $v0, 4
575     la $a0, endstring
576     syscall
577     li $v0, 10
578     syscall
579 else:                                         # leggo 1 elemento successivo
580     lw $t0,20($t0)
581     lw $t1,20($t0)                         # aggiorno il successivo
582     j loop
583 else2:                                        # leggo 1 elemento successivo
584     lw $t0,20($t0)
585     lw $t1,20($t0)                         # aggiorno il successivo
586     j loop2

```

secondo.s