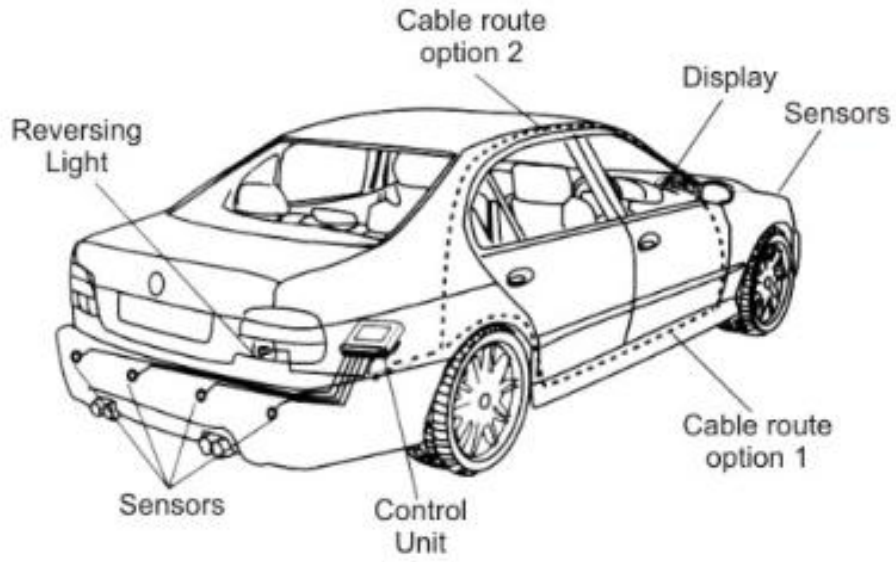
Progetto Assembly MIPS per il Corso di Architetture degli Elaboratori - A.A. 2017/2018 –

Monitoraggio di Sensori in Smart Veichles



RELAZIONE DEL PROGETTO

a cura di Fontani Alessio
consegnata in data 18/05/2018

email: alessio.fontani@stud.unifi.it

Descrizione della soluzione adottata

- Descrizione dell'algoritmo in linguaggio naturale
    - Procedure principali
    - Procedure secondarie
- Descrizione dell'algoritmo in pseudo-linguaggio
    - Procedure principali
    - Procedure secondarie
- Strutture dati utilizzate
- Motivazione delle scelte implementative

Test di corretto funzionamento

Requisiti di memoria dell'unità di monitoraggio

- Tabella di calcolo – Memoria occupata
- Copia del segmento dati
- Screenshot del segmento text
    - Spazio allocato nello stack
    - Spazio occupato dal codice

Tempo complessivo di esecuzione dell'unità di controllo

- Tabella di calcolo – Tempo complessivo di esecuzione

Ottimizzazione dell'unità di monitoraggio

Codice MIPS assembly implementato

Descrizione della soluzione adottata

Descrizione delle procedure principali in linguaggio naturale

- Prima procedura - pend
- Seconda procedura - ster
- Terza procedura - dist
- Quarta procedura - corp

La soluzione adottata, consiste in quattro procedure principali, chiamate una dopo l'altra dal metodo main.

La **prima procedura** chiama le procedure di apertura e lettura del file in ingresso pendenzaIN.txt, contenente i dati relativi al sensore di pendenza e salva i dati sullo spazio riservato di nome *buff*.
Dopo, chiama la procedura di conversione dei dati salvati in *buff,* che converte da CHAR a decimali. I dati convertiti vengono inseriti nell'array chiamato *arr.*
Successivamente esegue le istruzioni di controllo di correttezza del sensore.
Queste istruzioni controllano che i valori rientrino in un range compreso tra -60 e 60.
Se rientrano nel range, inserisce 1 nel vettore *OUT1*, altrimenti inserisce 0.
Dopo aver effettuato questa operazione per tutti i valori in ingresso, chiama la procedura di apertura del file in uscita e la procedura che scrive su file i valori in *OUT1*. I dati vengono scritti sul file correttezzaPendenzaOUT.txt.
Infine chiama la procedura che ripulisce *buff*, inserendo il valore corrispondente al carattere NULL in tutta la sua dimensione per poi tornare al metodo main.

La **seconda procedura** chiama le procedure di apertura e lettura del file in ingresso sterzoIN.txt, contenente i dati relativi al sensore dello sterzo e salva i dati sullo spazio riservato di nome *buff*.
Dopo, chiama la procedura di conversione dei dati salvati in *buff* e li inserisce convertiti nell'array *arr*, per poi iniziare con le istruzioni di controllo.
Il controllo consiste nel verificare che i valori siano compresi tra 0 e 100 e dal secondo valore in poi controlla che il valore differisca di al massimo 10 gradi rispetto al valore precedente.
Questa operazione viene effettuata sottraendo al valore più grande quello più piccolo e controllando che il risultato sia minore o uguale a 10.
Se entrambi i vincoli sono rispettati inserisce 1 nell'array di uscita *OUT2*, altrimenti inserisce 0.
Finiti i valori, viene richiamata la procedura di apertura del file in uscita e quella di scrittura dell'array *OUT2* nel file correttezzaSterzoOUT.txt.
Infine chiama la procedura che ripulisce *buff*, inserendo il valore corrispondente al carattere NULL in tutta la sua dimensione per poi tornare al metodo main.

La **terza procedura** chiama le procedure di apertura e lettura del file in ingresso distanzaIN.txt, contenente i valori relativi al sensore di distanza e salva i dati sullo spazio riservato di nome *buff*.
Poi, chiama la procedura di salvataggio del tipo di ostacolo e della distanza dallo stesso. I valori vengono salvati rispettivamente in *arr2* e *arr*.
Quest'ultima procedura, tiene conto che i valori in ingresso sono in base 16 e li trasforma in base 10, per poi inserirli nell'array *arr*.
Successivamente iniziano le operazioni di controllo.
Queste consistono nel ritenere errati i valori minori o uguali a 0 e i valori maggiori di 50. Successivamente, dal terzo valore in poi e se i valori sono corretti, controlla se il valore di distanza dell'ostacolo attuale è uguale a quello dei due ostacoli precedenti. Se lo è, controlla se il tipo è uguale a quello dei due ostacoli precedenti. Se lo sono, mette 0 nell'array *OUT3*, altrimenti inserisce 1.
Controllati tutti i valori, richiama le procedure di apertura del file di output correttezzaDistanzaOUT.txt e di scrittura di *OUT3* nel file.
Infine torna al metodo main.

La **quarta ed ultima procedura** chiamata, somma i valori di correttezza al tempo t, di tutti e tre i sensori. I valori di correttezza sono stati inseriti dalle precedenti procedure nei vettori *OUT1*, *OUT2* e *OUT3*.
Questi vettori verranno riutilizzati per salvare i valori di correttezza del sistema, secondo le varie politiche.
A seconda del risultato della somma, il programma esegue istruzioni differenti.
Se il risultato è 0 inserisce zero in tutti e tre gli array in uscita.
Se il risultato è 1 inserisce 1 nell'array di uscita *OUT3*, corrispondente alla politica P3.
Se il risultato è 2 inserisce 1 negli arrays di uscita *OUT3* e *OUT2*. *OUT2* corrisponde alla politica P2.
Se il risultato è 3 inserisce 1 in tutti gli arrays di uscita. *OUT1* corrisponde alla politica P1.
Una volta che tutti i valori di correttezza del sistema sono stati calcolati, vengono chiamate le procedure di apertura dei file in uscita (correttezzaP1.txt, correttezzaP2.txt e correttezzaP3.txt) e le procedure di scrittura su file dei risultati corrispondenti.
Infine torna al metodo main che esegue le istruzioni di chiusura del programma.

Descrizione delle procedure secondarie in linguaggio naturale

Le procedure secondarie, sono chiamate all'interno delle procedure principali.

apriL: apre i file contenenti i dati in ingresso (usa il flag ReadOnly)

apriS: apre i file su cui salvare i dati in uscita (usa il flag WriteOnly with create)

leggi: inserisce nel buffer "*buff*" i dati letti nel file di ingresso (Max 127 word)

scrivi: scrive nei file di output i dati contenuti negli arrays contenenti i dati di output

pulizia: re inizializza il buffer, ovvero cancella tutti i dati presenti sostituendoli con il valore NULL. Controlla attraverso un loop se le word sono vuote. Se non lo sono, inserisce nella word il valore NULL. Esce dal loop quando incontra la prima word vuota.

iniCon: questa procedura è utilizzata nella prima e nella seconda procedura principale.
La procedura utilizza come dati in ingresso, i dati salvati in "*buff*". Questi dati sono i valori restituiti dal sensore e sono salvati in "*buff*" come caratteri. Ogni valore è separato dagli altri da uno SPACE.
Questi valori sono degli interi rappresentati tramite caratteri.
La procedura iniCon prende in ingresso un CHAR alla volta, ovvero un byte alla volta, per trasformare i caratteri negli interi corrispondenti.
Se il carattere corrisponde ad un numero, lo trasforma nell'intero corrispondente (ovvero gli sottrae 48) e lo somma in una variabile X, dopo che X è stata moltiplicata per 10. Se il CHAR successivo è uno SPACE, salva il contenuto di X nell'array "*arr*". Quindi, ripulisce X inserendoci 0 e carica il carattere successivo.
Invece, se il CHAR successivo è un numero, lo trasforma nell'intero corrispondente e lo somma ad X dopo che X è stata moltiplicata per 10.
L'operazione di moltiplicazione per 10 viene fatta perché, se c'è un secondo numero, significa che il primo che era stato inserito in X non è un unità, ma una decina.
Questa operazione si ripete fino alla fine di "*buff*", ovvero quando il byte caricato corrisponde al carattere NULL.

inisv: Questa procedura è utilizzata nella terza procedura principale.
La procedura utilizza come dati in ingresso, i dati salvati in "*buff*". Questi dati sono i valori restituiti dal sensore di distanza e sono salvati in "*buff*" come caratteri. Ogni valore è separato dagli altri da uno SPACE.

Il primo CHAR di un valore corrisponde al tipo di ostacolo mentre i CHAR successivi rappresentano un valore numerico in base 16.
La procedura svolge due funzioni:

1. Salva il primo CHAR presente in "*buff*" ed ogni primo carattere dopo uno SPACE, ovvero salva il tipo di ostacolo.
   Se siamo al primo carattere o al primo carattere dopo uno SPACE, lo salva nell'array "*arr2*" per poi controllare il successivo e dare inizio alla seconda funzione, altrimenti continua con la seconda funzione.
2. Controlla i caratteri successivi al tipo di ostacolo fino ad incontrare uno space. In questa fase, prende i CHAR corrispondenti ai valori numerici in base 16 e li trasforma nei corrispondenti valori interi in base 10.
   In pratica, prende in ingresso il primo CHAR dopo il tipo di ostacolo, lo trasforma nell'intero corrispondente e lo somma ad X, dopo che X è stata moltiplicata per 16. Se il carattere successivo è uno SPACE, salva X nell'array "*arr*" e ripulisce X inserendoci 0, altrimenti trasforma il carattere nell'intero corrispondente e lo somma ad X dopo che X è stata moltiplicata per 16.
   Siccome il sensore di distanza restituisce valori in base 16, la moltiplicazione in questo caso è per 16 perché in uscita si vuole degli interi in base 10.
   Inoltre, durante la conversione, la procedura controlla se il carattere è una lettera o un numero. Se è una lettera, la trasforma nel decimale corrispondente sottraendo 55, altrimenti sottrae 48.

La procedura si conclude quando incontra il carattere NULL.

Descrizione delle procedure principali in pseudo-linguaggio

- Prima procedura – pend()
- Seconda procedura – ster()
- Terza procedura – dist()
- Quarta procedura – corp()

Metodo Main

```
void main(){
      pend();
      ster();
      dist();
      corp();

      fine();

}
```

Prima procedura

```
void pend() {

      apriL(pendenzaIn.txt);
      leggi(pendenzaIn.txt);
      iniCon(buff);

      int y=0;

      for(int i=0,i<100,i++){

            x = arr[i];

            if (x<60 && x>-60){
                  out1[y]="1";
                  out1[y+1]=" ";
            } else {
                  out1[y]="0";
                  out1[y+1]=" ";
            }

            y=y+2;

      }

      apriS(correttezzaPendenzaOUT.txt);
      scrivi(correttezzaPendenzaOUT.txt,char[] out1);
      pulizia();

}
```

Seconda procedura

```
void ster() {

      apriL(sterzoIn.txt);
      leggi(sterzoIn.txt);
      iniCon(buff);

      int y=0;
      int z=0;
      int res=0;

      for(int i=0, i<100, i++){

            x = arr[i];

            if ( x>0 && x<100){
                  if (i=0){
                        out2[y]="1";
                        out2[y+1]=" ";
                  } else {
                        if(z>x){
                              res= z-x;
                        } else {
                              res= x-z;
                        }
                        if (res<=10){
                              out2[y]="1";
                              out2[y+1]=" ";
                        } else {
                              out2[y]="0";
                              out2[y+1]=" ";
                        }
                  }
            } else {
                  out2[y]="0";
                  out2[y+1]=" ";
            }

            z = x;
            y = y+2;

      }

      apriS(correttezzaSterzoOUT.txt);
      scrivi(correttezzaSterzoOUT.txt,char[] out2);
      pulizia();
}
```

Terza procedura

```
void dist(){

      apriL(distanzaIn.txt);
      leggi(distanzaIn.txt);
      inisv(buff);

      int y = 0;
      char z;

      for(int i=0, i<100, i++){
            x = arr[i];
            z = arr2[i];
            if ( x>0 $$ x<=50){
                  if ( i<2 ) {
                        out3[y]="1";
                        out3[y+1]=" ";
                  } else {
                        if (x==arr[i-1] $$ x==arr[i-2] $$ z==arr2[i-1]
$$ z==arr2[i-2] ){
                              out3[y]="0";
                              out3[y+1]=" ";
                        } else {
                              out3[y]="1";
                              out3[y+1]=" ";
                        }
                  }
            } else {
                  out3[y]="0";
                  out3[y+1]=" ";
            }

            y= y+2;

      }

      apriS(correttezzaDistanzaOUT.txt);
      scrivi(correttezzaDistanzaOUT.txt,char[] out3);

}
```

Quarta procedura

```
void corp(){

	int res = 0
	int y = 0;

	for(int i=0, i<100, i++){

		res = out1[y] + out2[y] + out3[y];

		switch (res){

			case 0:		out1[y]=0;
					out2[y]=0;
					out3[y]=0;
					break;

			case 1:		out1[y]=0;
					out2[y]=0;
					out3[y]=1;
					break;

			case 2:		out1[y]=0;
					out2[y]=1;
					out3[y]=1;
					break;

			case 3:		out1[y]=1;
					out2[y]=1;
					out3[y]=1;
					break;
		}

		y= y+2;

	}

	apriS(correttezzaP1.txt);
	scrivi(correttezzaP1.txt,char[] out1);
	apriS(correttezzaP2.txt);
	scrivi(correttezzaP2.txt,char[] out2);
	apriS(correttezzaP3.txt);
	scrivi(correttezzaP3.txt,char[] out3);

}
```

Descrizione delle procedure secondarie in pseudo-linguaggio

- pulizia
- iniCon
- inisv

pulizia

```
void pulizia(){

      int i = 0;

      char x = buff[i];

      while ( x != NULL ){
            buff[i] = "0";
            i = i+1;
            x = buff[i];
      }
}
```

iniCon

```
void iniCon(){

      int i = 0;
      int j = 0;
      int num = 0;
      char x = buff[i];
      boolean neg = false;

      while (x != NULL){
            if (x != SPACE){
                  if ( x == "-" ){
                        neg = true;
                  } else {
                        x = x-48;
                        num = num*10;
                        num = num+x;
                  }
            } else {
                  if (neg){
                        num = num - 2*num;
                  }
                  arr[j] = num;
                  neg = false;
                  num = 0;
                  j=j+1;
            }
            i = i+1;
            x = buff[i];
      }
}
```

inisv

```
void inisv() {

      int i = 0;
      int j = 0;
      int z = 0;
      char x = buff[i];
      boolean sp = true;
      while (x != NULL){
            if (sp){
                  arr2[j]= x;
                  j = j+1;
                  sp = false;
            } else {
                  if ( x == SPACE){
                        arr[z]= num;
                        z= z+1;
                        sp = true;
                  } else {
                        if (x == LETTERA){
                              x = x-55;
                              num = num * 16;
                              num = num + x;
                        } else {
                              x = x-48;
                              num = num * 16;
                              num = num + x;
                              }
                  }
            }
            i = i+1;
            x = buff[i];
      }
      if (x == NULL){
            arr[z]= num;
      }
}
```

Descrizione delle strutture dati utilizzate

Per questo progetto sono state utilizzate zone di memoria contenenti stringhe e arrays.

buff = zona di memoria di 512 byte, riservata ai dati presenti sui file in input. I dati sono salvati in questa zona di memoria come caratteri.
Questa zona di memoria viene sovrascritta ogni volta che ci sono nuovi dati in ingresso, per evitare di occupare ulteriore memoria.

arr = array di 400 byte, contenente valori interi. Una volta che i valori numerici presenti in buff sono stati convertiti, vengono inseriti in questo array.
I valori dell'array vengono sovrascritti ogni volta che ci sono nuovi dati, per evitare di occupare ulteriore memoria.

arr2 = array di 100 byte, contenente caratteri. Qua vengono salvati i tipi di ostacoli del sensore di distanza.

out1, out2, out3 = arrays di 199 byte ciascuno, utilizzati per i dati in uscita. I dati sono salvati come caratteri.
Le procedure, dopo aver controllato la correttezza dei dati in ingresso, inseriscono il risultato in questi arrays.
out1 contiene i valori di correttezza del sensore di pendenza, out2 del sensore di sterzo e out3 del sensore di distanza.
Successivamente out1, out2 e out3 vengono sovrascritti coi valori di correttezza dell'intero sistema, così da evitare l'occupazione di ulteriore memoria.
Per la politica p1 i dati vengono inseriti in out1, per la politica p2 in out2 e per la politica p3 in out3.

Motivazione delle scelte implementative

Perché sono presenti procedure di conversione dati?
Le procedure di conversione dei dati in ingresso sono state implementate per semplificare la programmazione delle procedure di controllo.

Perché è presente una procedura di pulizia di buff e non per gli arrays?
La procedura di pulizia della zona dati *buff* è stata implementata per rendere *buff* riutilizzabile, così da evitare di occupare ulteriore memoria.
La zona dati *buff* a differenza degli arrays, deve essere ripulita.
Questo perché salvando su *buff* i dati presenti sul secondo file in ingresso, questi potrebbero occupare meno byte rispetto ai dati del file precedente. Questo significa che dopo l'ultimo byte del secondo file, rischiamo di trovare i byte del file precedente, falsando così i risultati.
Questo non succede negli arrays perché ogni posizione degli arrays viene sovrascritta dai nuovi dati.

Perché sono stati utilizzati gli arrays invece che le liste?
La scelta di utilizzare gli arrays dipende dal fatto che il numero di dati in ingresso è sempre lo stesso e deciso precedentemente. Usare una lista può essere utile quando non sappiamo quanti elementi abbiamo e vogliamo evitare di occupare più spazio del necessario o quando si devono effettuare eliminazioni di dati e spostamenti. Non essendo questi i casi, la scelta degli arrays è stata la più motivata.

Perché $ra è stato salvato nello stack piuttosto che salvarlo in un altro registro?
Salvare l'indirizzo contenuto in $ra nello stack ci da la sicurezza che non verrà modificato. Se salviamo $ra in un registro prima di fare la chiamata della procedura rischiamo di perderlo perchè non ci sono garanzie che la procedura chiamata non modifichi il registro in cui salviamo $ra.

Test di corretto funzionamento

Le tabelle seguenti mostrano il corretto funzionamento dell'unità di monitoraggio. Per maggiori dati, consultare i file di ingresso e di uscita.

Sensore di pendenza

Dati in ingresso dal file pendenzaIN.txt																	
20	0	60	-60	165	21	7	-34	453	8	0	234	3	4	6	78	23	45
Valori di correttezza corrispondenti, salvati in correttezzaPendenzaOUT.txt																	
1	1	0	0	0	1	1	1	0	1	1	0	1	1	1	0	1	1

Sensore di sterzo

Dati in ingresso dal file sterzoIN.txt																	
22	34	37	-23	45	51	57	212	4	9	100	0	-4	5	23	65	68	59
Valori di correttezza corrispondenti, salvati in correttezzaSterzoOUT.txt																	
1	0	1	0	0	1	1	0	0	1	0	0	0	1	0	0	1	1

Sensore di distanza

Dati in ingresso dal file distanzaIN.txt																	
B30	A30	C43	B32	B20	B20	A20	A20	A21	C16	C16	C16	C00	C17	B15	A01	B45	C78
Valori di correttezza corrispondenti, salvati in correttezzaDistanzaOUT.txt																	
1	1	0	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0

Correttezza dell'intero sistema secondo le politiche P1, P2 e P3

Sensore di pendenza																	
1	1	0	0	0	1	1	1	0	1	1	0	1	1	1	0	1	1
Sensore di sterzo																	
1	0	1	0	0	1	1	0	0	1	0	0	0	1	0	0	1	1
Sensore di distanza																	
1	1	0	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0
Numero di sensori funzionanti																	
3	2	1	1	1	3	3	2	1	3	2	0	1	3	2	1	2	2
Politica P1, ovvero tutti e tre i sensori funzionanti																	
1	0	0	0	0	1	1	0	0	1	0	0	0	1	0	0	0	0
Politica P2, ovvero almeno due sensori funzionanti																	
1	1	0	0	0	1	1	1	0	1	1	0	0	1	1	0	1	1
Politica P3, ovvero almeno un sensore funzionante																	
1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1

Requisiti di memoria dell'unità di monitoraggio

Tabella di calcolo - Memoria occupata

Elemento	Quantità	Byte occupati dal singolo elemento	Byte occupati dal totale degli elementi
stringa fnf	1	17	17
stringa pin	1	15	15
stringa sin	1	13	13
stringa din	1	15	15
stringa pout	1	27	27
stringa sout	1	25	25
stringa dout	1	27	27
stringhe puno, pdue, ptre	3	18	54
array di word (jtab)	1	16	16
buff	1	512	512
array ingresso (arr)	1	400	400
array ingresso (arr2)	1	100	100
array uscita (out1,out2,out3)	3	199	597
jal, j ,jr	60	4	240
add	6	4	24
addi	48	4	192
sub	4	4	16
mul	3	4	12
sw	8	4	32
lw	12	4	48
sb	23	4	92
lb/lbu	8	4	32
li	53	4	212
la (lui)	1	4	4
la (lui+ori)	33	8	264
syscall	7	4	28
move (addu)	16	4	64
blt (slt+bne)	5	8	40
beqz	2	4	8
bne	8	4	32
beq	10	4	40
ble (slt+beq)	1	8	8
bgt (slt+beq/slt+bne)	6	8	48
seq (beq+ori+beq+ori)	1	16	16
bge (slt+beq)	1	8	8
Spazio max allocato nello stack	2	4	8
Spazio totale occupato dai dati			1818
Spazio totale occupato dalle istruzioni			1460
Spazio totale occupato da dati + istruzioni + spazio massimo allocato nello stack			3286

La conferma della correttezza dei dati è mostrata dal segmento dati e dal segmento text nelle prossime pagine.

Copia del segmento dati

User data segment [10000000]..[10040000]

```
[10000000]..[1000ffff]  00000000
[10010000]    656c6946  746f6e20  756f6620  203a646e    F i l e   n o t   f o u n d :
[10010010]    6e657000  7a6e6564  2e4e4961  00747874    . p e n d e n z a I N . t x t .
[10010020]    72657473  4e496f7a  7478742e  73696400    s t e r z o I N . t x t . d i s
[10010030]    7a6e6174  2e4e4961  00747874  72726f63    t a n z a I N . t x t . c o r r
[10010040]    65747465  50617a7a  65646e65  4f617a6e    e t t e z z a P e n d e n z a O
[10010050]    742e5455  63007478  6572726f  7a657474    U T . t x t . c o r r e t t e z
[10010060]    7453617a  6f7a7265  2e54554f  00747874    z a S t e r z o O U T . t x t .
[10010070]    72726f63  65747465  44617a7a  61747369    c o r r e t t e z z a D i s t a
[10010080]    4f617a6e  742e5455  63007478  6572726f    n z a O U T . t x t . c o r r e
[10010090]    7a657474  3150617a  7478742e  726f6300    t t e z z a P 1 . t x t . c o r
[100100a0]    74746572  617a7a65  742e3250  63007478    r e t t e z z a P 2 . t x t . c
[100100b0]    6572726f  7a657474  3350617a  7478742e    o r r e t t e z z a P 3 . t x t
[100100c0]    00000000  004004fc  00400518  00400534    . . . . . @ . . . @ . 4 . @ .
[100100d0]    00400550  20303342  20303341  20333443    P . @ . B 3 0   A 3 0   C 4 3
[100100e0]    20323342  20303242  20303242  20303241    B 3 2   B 2 0   B 2 0   A 2 0
[100100f0]    20303241  20313241  20363143  20363143    A 2 0   A 2 1   C 1 6   C 1 6
[10010100]    20363143  20303043  20373143  20353142    C 1 6   C 0 0   C 1 7   B 1 5
[10010110]    20313041  20353442  20383743  20433241    A 0 1   B 4 5   C 7 8   A 2 C
[10010120]    20433242  20433242  20433243  20433241    B 2 C   B 2 C   C 2 C   A 2 C
[10010130]    20433241  20433241  20433241  20343242    A 2 C   A 2 C   A 2 C   B 2 4
[10010140]    20313341  20463243  20333141  20353142    A 3 1   C 2 F   A 1 3   B 1 5
[10010150]    20463342  20333241  20333242  20333243    B 3 F   A 2 3   B 2 3   C 2 3
[10010160]    20333242  20333242  20333242  20333243    B 2 3   B 2 3   B 2 3   C 2 3
[10010170]    20353441  20363343  20343242  20343243    A 4 5   C 3 6   B 2 4   C 2 4
[10010180]    20343243  20343241  20343242  20343243    C 2 4   A 2 4   B 2 4   C 2 4
[10010190]    20343243  20363341  20333441  20323142    C 2 4   A 3 6   A 4 3   B 1 2
[100101a0]    20323143  20323141  20323142  20323141    C 1 2   A 1 2   B 1 2   A 1 2
[100101b0]    20323143  20323143  20323143  20323143    C 1 2   C 1 2   C 1 2   C 1 2
[100101c0]    20353041  20353043  20373442  20363141    A 0 5   C 0 5   B 4 7   A 1 6
[100101d0]    20363142  20363142  20363143  20363143    B 1 6   B 1 6   C 1 6   C 1 6
[100101e0]    20363143  20363143  20423541  20413643    C 1 6   C 1 6   A 5 B   C 6 A
[100101f0]    20303042  20343142  20343143  20353343    B 0 0   B 1 4   C 1 4   C 3 5
[10010200]    20393041  20393042  20393043  20393041    A 0 9   B 0 9   C 0 9   A 0 9
[10010210]    20393043  20393042  20393042  20393042    C 0 9   B 0 9   B 0 9   B 0 9
[10010220]    20393042  20333243  20353441  20313042    B 0 9   C 2 3   A 4 5   B 0 1
[10010230]    20343041  20353143  20343341  20303242    A 0 4   C 1 5   A 3 4   B 2 0
[10010240]    20373143  20383141  20363542  20333443    C 1 7   A 1 8   B 5 6   C 4 3
[10010250]    20343141  20393142  20303341  20313441    A 1 4   B 1 9   A 3 0   A 4 1
[10010260]    00373842  00000000  00000000  00000000    B 8 7 . . . . . .
[10010270]..[100102d3]  00000000
[100102d4]    00000030  00000030  00000043              0 . . . 0 . . . C . . .
[100102e0]    00000032  00000020  00000020  00000020    2 . . .   . . .   . . .   . . .
[100102f0]    00000020  00000021  00000016  00000016      . . . ! . . . . . . . . . . .
[10010300]    00000016  00000000  00000017  00000015    . . . . . . . . . . . . . . . .
[10010310]    00000001  00000045  00000078  0000002c    . . . . E . . . x . . . , . . .
[10010320]    0000002c  0000002c  0000002c  0000002c    , . . . , . . . , . . . , . . .
[10010330]    0000002c  0000002c  0000002c  00000024    , . . . , . . . , . . . $ . . .
[10010340]    00000031  0000002f  00000013  00000015    1 . . . / . . . . . . . . . . .
[10010350]    0000003f  00000023  00000023  00000023    ? . . . # . . . # . . . # . . .
[10010360]    00000023  00000023  00000023  00000023    # . . . # . . . # . . . # . . .
[10010370]    00000045  00000036  00000024  00000024    E . . . 6 . . . $ . . . $ . . .
[10010380]    00000024  00000024  00000024  00000024    $ . . . $ . . . $ . . . $ . . .
[10010390]    00000024  00000036  00000043  00000012    $ . . . 6 . . . C . . . . . . .
[100103a0]    00000012  00000012  00000012  00000012    . . . . . . . . . . . . . . . .
[100103b0]    00000012  00000012  00000012  00000012    . . . . . . . . . . . . . . . .
[100103c0]    00000005  00000005  00000047  00000016    . . . . . . . G . . . . . . . .
[100103d0]    00000016  00000016  00000016  00000016    . . . . . . . . . . . . . . . .
[100103e0]    00000016  00000016  0000005b  0000006a    . . . . . . . [ . . . j . . . .
[100103f0]    00000000  00000014  00000014  00000035    . . . . . . . . . . . 5 . . . .
[10010400]    00000009  00000009  00000009  00000009    . . . . . . . . . . . . . . . .
[10010410]    00000009  00000009  00000009  00000009    . . . . . . . . . . . . . . . .
[10010420]    00000009  00000023  00000045  00000001    . . . . # . . . E . . . . . . .
```

```
[10010430]    00000004  00000015  00000034  00000020    . . . . . 4 . . .   . . .
[10010440]    00000017  00000018  00000056  00000043    . . . . . V . . . C . . .
[10010450]    00000014  00000019  00000030  00000041    . . . . . 0 . . . A . . .
[10010460]    00000087  42434142  41414242  43434341    . . . . B A C B B B A A A C C C
[10010470]    41424343  42414342  41414342  41424141    C C B A B C A B B C A A A A B A
[10010480]    42424143  42434241  41434242  43434243    C A B B A B C B B B C A C B C C
[10010490]    43434241  43424141  43414241  41434343    A B C C A A B C A B A C C C C A
[100104a0]    42414243  43434342  42434143  41434342    C B A B B C C C C A C B B C C A
[100104b0]    43414342  42424242  41424143  43424143    B C A C B B B B C A B A C A B C
[100104c0]    41434241  42414142  20302031  20302030    A B C A B A A B 1   0   0   0
[100104d0]    20312030  20302031  20312030  20302030    0   1   1   0   0   1   0   0
[100104e0]    20312030  20302030  20302030  20302030    0   1   0   0   0   0   0   0
[100104f0]    20302030  20312030  20302030  20302030    0   0   0   1   0   0   0   0
[10010500]    20302030  20302030  20302030  20302030    0   0   0   0   0   0   0   0
[10010510]    20302030  20302030  20302030  20302030    0   0   0   0   0   0   0   0
[10010520]    20302030  20302030  20302030  20302030    0   0   0   0   0   0   0   0
[10010530]    20312031  20312030  20302031  20302030    1   1   0   1   1   0   0   0
[10010540]    20302031  20312030  20302031  20302031    1   0   0   1   1   0   1   0
[10010550]    20302030  20302030  20312031  20312030    0   0   0   0   1   1   0   1
[10010560]    20312031  20312031  20312030  20302030    1   1   1   1   0   1   0   0
[10010570]    20302031  20312030  20302030  20312030    1   0   0   1   0   0   0   1
[10010580]    20302030  20302030  20312031  31302030    0   0   0   0   1   1   0   0 1
[10010590]    30203120  30203020  31203120  30203120      1   0   0   0   1   1   1   0
[100105a0]    31203120  30203020  31203120  31203020      1   1   0   0   1   1   0   1
[100105b0]    30203120  31203120  31203120  30203120      1   0   1   1   1   1   1   0
[100105c0]    31203020  31203120  31203120  31203020      0   1   1   1   1   1   0   1
[100105d0]    30203120  30203020  31203120  30203020      1   0   0   0   1   1   0   0
[100105e0]    31203120  31203120  30203020  30203020      1   1   1   1   0   0   0   0
[100105f0]    31203020  31203020  31203120  31203120      0   1   0   1   1   1   1   1
[10010600]    31203020  31203120  31203120  31203120      0   1   1   1   1   1   1   1
[10010610]    31203120  30203020  31203020  31203020      1   1   0   0   0   1   0   1
[10010620]    31203120  31203120  31203120  31203120      1   1   1   1   1   1   1   1
[10010630]    31203120  31203120  31203120  31203120      1   1   1   1   1   1   1   1
[10010640]    31203120  31203120  31203120  31203120      1   1   1   1   1   1   1   1
[10010650]    31203120  20313120  20312031  20312031      1   1   1 1   1   1   1   1
[10010660]    20312031  20312031  20312031  20312030    1   1   1   1   1   1   0   1
[10010670]    20312031  20312031  20312031  20312031    1   1   1   1   1   1   1   1
[10010680]    20312031  20312031  20312031  20312031    1   1   1   1   1   1   1   1
[10010690]    20312031  20312031  20312031  20312031    1   1   1   1   1   1   1   1
[100106a0]    20312031  20312031  20312031  20312031    1   1   1   1   1   1   1   1
[100106b0]    20312031  20312031  20312031  20312031    1   1   1   1   1   1   1   1
[100106c0]    20312031  20312031  20312031  20312031    1   1   1   1   1   1   1   1
[100106d0]    20312031  20312031  20312031  20312031    1   1   1   1   1   1   1   1
[100106e0]    20312031  20312031  20312031  20312031    1   1   1   1   1   1   1   1
[100106f0]    20312031  20312031  20312031  20312031    1   1   1   1   1   1   1   1
[10010700]    20312031  20312031  20312031  20312031    1   1   1   1   1   1   1   1
[10010710]    20312031  20312031  20312031  00002031    1   1   1   1   1   1   1   . .
[10010720]..[1003ffff]  00000000
```

Osservando il segmento dati possiamo notare che:

- Le stringhe iniziali, corrispondenti al messaggio di errore e ai nomi dei file occupano da 10010000 a 100100c1, ovvero occupano 193 byte
- L'array di word jtab occupa 4 word da 100100c4 a 100100d3, ovvero 16 byte
- *buff* occupa da 100100d4 a 100102d3, ovvero 512 byte
- L'array *arr* occupa da 100102d4 a 10010463, ovvero 400 byte
- L'array *arr* occupa da 10010464 a 100104c7, ovvero 100 byte
- L'array *out1* occupa da 100104c8 a 1001058e, ovvero 199 byte
- L'array *out2* occupa da 1001058f a 10010655, ovvero 199 byte
- L'array *out3* occupa da 10010656 a 1001071d, ovvero 199 byte

La somma di questi spazi di memoria è uguale a 1818 byte, così come calcolato nella tabella.

Screenshot del segmento text

- Spazio allocato nello stack
- Spazio occupato dal codice

Spazio allocato nello stack

Gli screenshot seguenti, mostrano in quali casi è stato raggiunto lo spazio massimo allocato nello stack.

Il valore sottolineato in rosso corrisponde all'attuale posizione dello stack pointer.
Lo spazio allocato dal programma va da 7ffff508 a 7ffff50f e quindi, come riportato nella tabella, lo spazio massimo allocato dal programma è di 8 byte.

Il valore cerchiato in rosso, corrisponde all'ultimo valore di $ra salvato prima della chiamata di un'altra procedura.
Infatti, come possiamo vedere dall'ultimo screenshot, i valori cerchiati in rosso, corrispondono esattamente all'indirizzo di ritorno delle procedure chiamate dal metodo Main.

Questi valori sono stati salvati perché le procedure chiamate dal metodo Main non sono procedure foglia, ovvero richiamano anch'esse altre procedure.
Quest'ultime invece, sono procedure foglia, perciò non è necessario allocare altri 4 byte per salvare l'indirizzo di ritorno da queste procedure.

Il valore 00400018 corrisponde all'indirizzo di ritorno alla procedura con cui il simulatore inizia ad eseguire il codice (sottolineato in rosso nell'ultimo screenshot).

```
R11 [t3] = ffffff9f
R12 [t4] = 100106bc
R13 [t5] = fffffc5
R14 [t6] = 64
R15 [t7] = 20
```

R28 [gp] = 10008000
R29 [sp] = 7ffff508
R30 [s8] = 0
R31 [ra] = 400030

```
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 10008000
R29 [sp] = 7ffff508
R30 [s8] = 0
R31 [ra] = 400030
```

```
User Stack [7ffff508]..[80000000]
[7ffff508]    00400030  00400018
[7ffff510]    00000004  7ffff5f8  7ffff5f4  7ffff5eb
[7ffff520]    7ffff5e5  00000000  7fffffc7  7fffffa7
[7ffff530]    7fffff88  7fffff5f  7fffff28  7ffffeec
[7ffff540]    7ffffebb  7ffffea5  7ffffe81  7ffffe50
[7ffff550]    7ffffe28  7ffffe14  7ffffde1  7ffffdd4
[7ffff630]    4d4f4330  4f4f544e  443d534c  72505c3a
```

User Stack [7ffff508]..[80000000]
[7ffff508] 00400030 00400018
[7ffff510] 00000004 7ffff5f8
[7ffff520] 7ffff5e5 00000000
[7ffff530] 7fffff88 7fffff5f
[7ffff540] 7ffffebb 7ffffea5

```
R29 [sp] = 7ffff508
R30 [s8] = 0
R31 [ra] = 400034
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 10010464
R12 [t4] = 30
R13 [t5] = 10010783
R14 [t6] = 1
```

R28 [gp] = 10008000
R29 [sp] = 7ffff508
R30 [s8] = 0
R31 [ra] = 400034

```
R25 [t9] = 30
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 10008000
R29 [sp] = 7ffff508
R30 [s8] = 0
R31 [ra] = 400034
```

```
User Stack [7ffff508]..[80000000]
[7ffff508]    00400034  00400018
[7ffff510]    00000004  7ffff5f8  7ffff5f4  7ffff5eb
[7ffff520]    7ffff5e5  00000000  7fffffc7  7fffffa7
[7ffff530]    7fffff88  7fffff5f  7fffff28  7ffffeec
[7ffff540]    7ffffebb  7ffffea5  7ffffe81  7ffffe50
[7ffff630]    4d4f4330  4f4f544e  443d534c  72505c3a
[7ffff640]    6172676f  5c696d6d  6d6d6f43  5c376e6f
[7ffff650]    6c6f6f54  56005c73  5f584f42  5f49534d
[7ffff660]    54534e49  5f4c4c41  48544150  5c3a433d
[7ffff670]    676f7250  206d6172  656c6946  724f5c73
[7ffff680]    656c6361  7269565c  6c617574  5c786f42
```

User Stack [7ffff508]..[80000000]
[7ffff508] 00400034 00400018
[7ffff510] 00000004 7ffff5f8
[7ffff520] 7ffff5e5 00000000
[7ffff530] 7fffff88 7fffff5f
[7ffff540] 7ffffebb 7ffffea5

```
R8   [t0] = 13
R9   [t1] = 0
R10  [t2] = 0
R11  [t3] = 43
R12  [t4] = 12
R23  [s7] = 10
R24  [t8] = 31
R25  [t9] = 30
R26  [k0] = 0
R27  [k1] = 0
R28  [gp] = 10008000
R29  [sp] = 7ffff508
R30  [s8] = 0
R31  [ra] = 400038
```

```
R28 [gp] = 10008000
R29 [sp] = 7ffff508
R30 [s8] = 0
R31 [ra] = 400038
```

```
User Stack [7ffff508]..[80000000]
[7ffff508]    00400038  00400018
[7ffff510]    00000004  7ffff5f8  7ffff5f4  7ffff5eb
[7ffff520]    7ffff5e5  00000000  7fffffc7  7fffffa7
[7ffff530]    7fffff88  7fffff5f  7fffff28  7ffffeec
[7ffff540]    7ffffebb  7ffffea5  7ffffe81  7ffffe50
[7ffff550]    7ffffe28  7ffffe14  7ffffde1  7ffffdd4
```

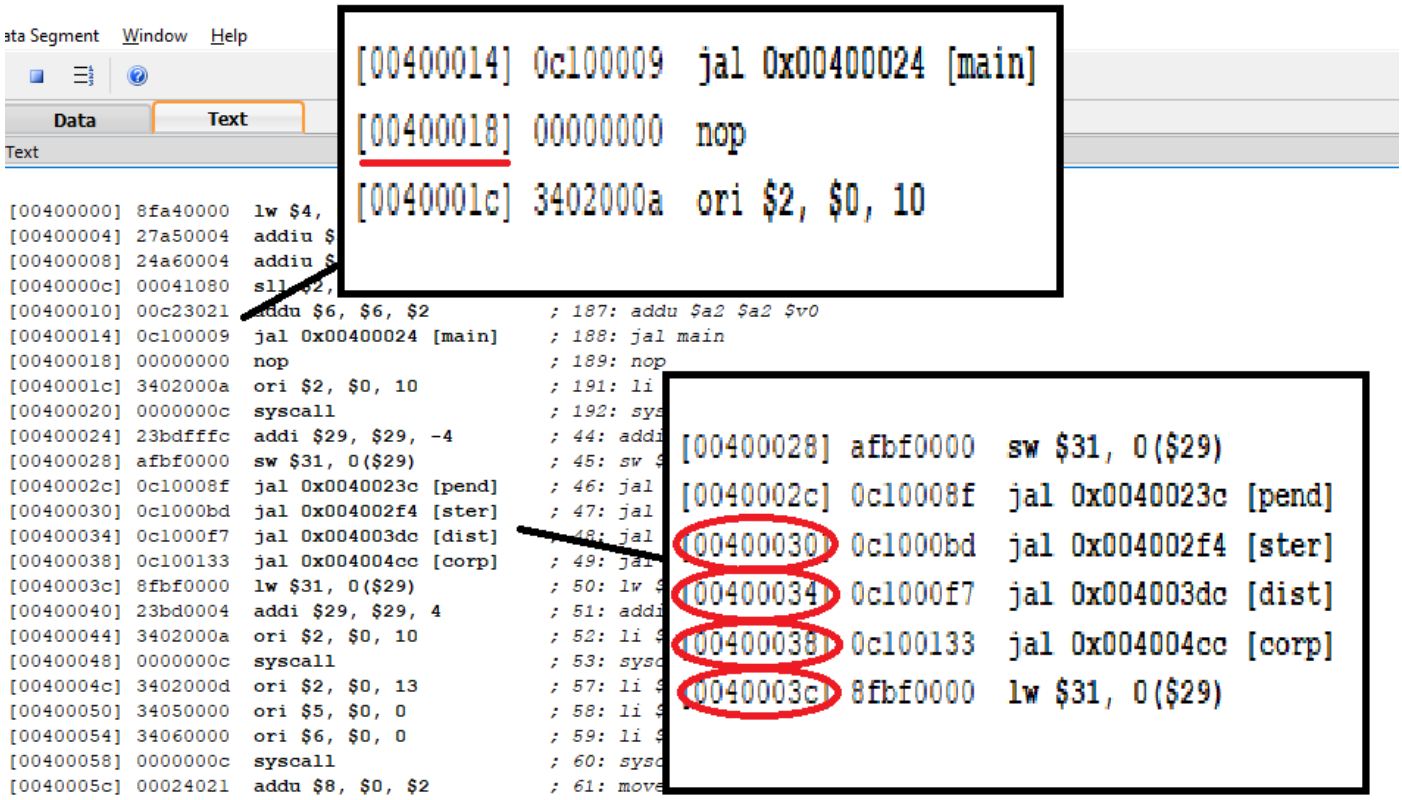
```
User Stack [7ffff508]..[80000000]
[7ffff508]    00400038  00400018
[7ffff510]    00000004  7ffff5f8
[7ffff520]    7ffff5e5  00000000
[7ffff530]    7fffff88  7fffff5f
[7ffff540]    7ffffebb  7ffffea5
```

```
R0   [r0] = 0
R1   [at] = 10010000
R2   [v0] = c7
R3   [v1] = 0
R4   [a0] = 1c
R5   [a1] = 10010782
R6   [a2] = c7
R7   [a3] = 0
R8   [t0] = 1c
R9   [t1] = 10010783
R10  [t2] = 1001084a
R11  [t3] = 0
R12  [t4] = 1
R13  [t5] = 64
R14  [t6] = 8
R15  [t7] = 4
R25  [t9] = 30
R26  [k0] = 0
R27  [k1] = 0
R28  [gp] = 10008000
R29  [sp] = 7ffff508
R30  [s8] = 0
R31  [ra] = 40003c
```

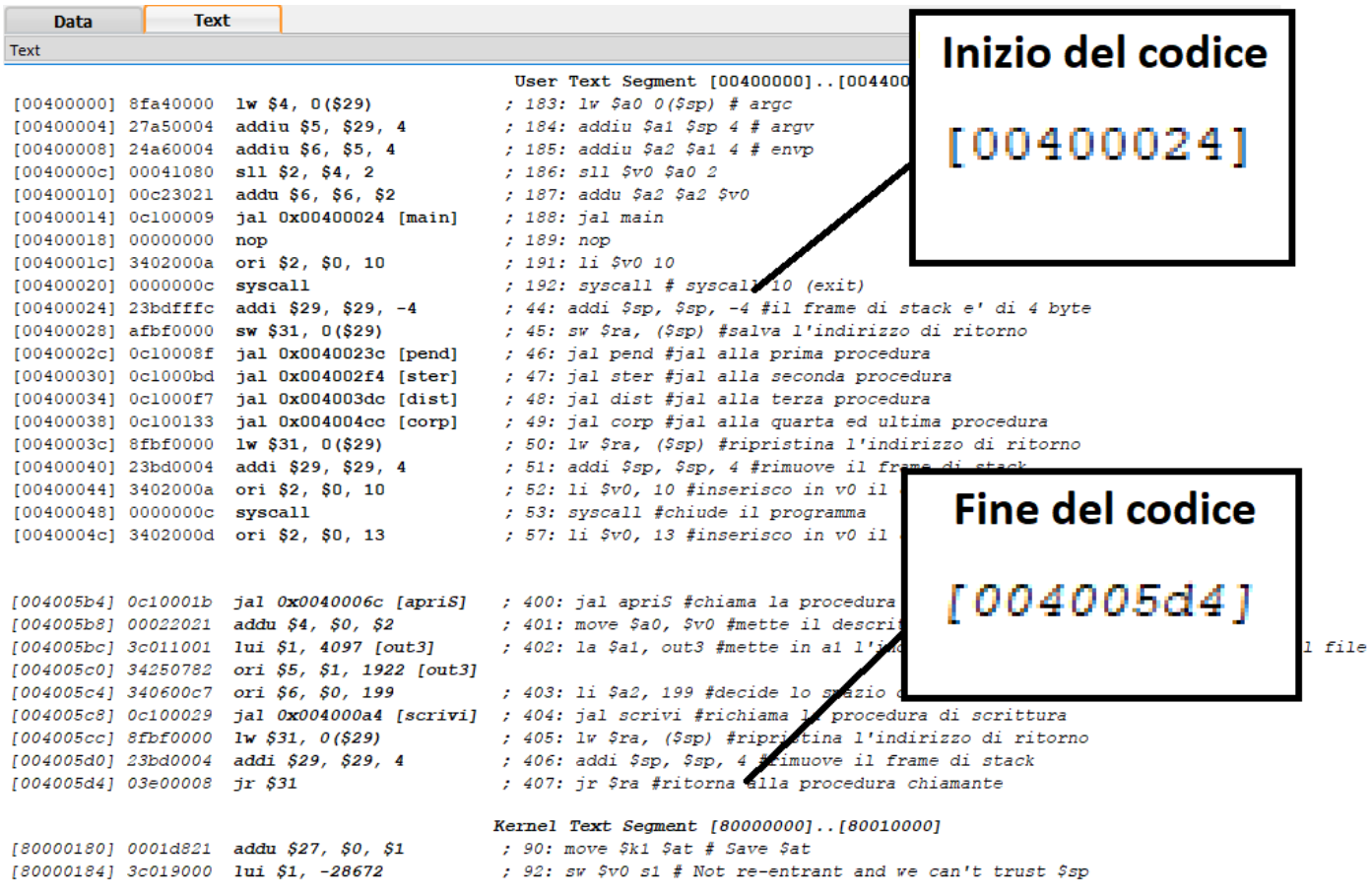
```
R28 [gp] = 10008000
R29 [sp] = 7ffff508
R30 [s8] = 0
R31 [ra] = 40003c
```

```
User Stack [7ffff508]..[80000000]
[7ffff508]    0040003c  00400018
[7ffff510]    00000004  7ffff5f8  7ffff5f4  7ffff5eb
[7ffff520]    7ffff5e5  00000000  7fffffc7  7fffffa7
[7ffff530]    7fffff88  7fffff5f  7fffff28  7ffffeec
[7ffff540]    7ffffebb  7ffffea5  7ffffe81  7ffffe50
[7ffff550]    7ffffe28  7ffffe14  7ffffde1  7ffffdd4
[7ffff630]    4d4f4330  4f4f544e  443d534c  72505c3a
[7ffff640]    6172676f  5c696d6d  6d6d6f43  5c376e6f
[7ffff650]    6c6f6f54  56005c73  5f584f42  5f49534d
[7ffff660]    54534e49  5f4c4c41  48544150  5c3a433d
[7ffff670]    676f7250  206d6172  656c6946  724f5c73
[7ffff680]    656c6361  7269565c  6c617574  5c786f42
[7ffff690]    45535500  4f525052  454c4946  5c3a433d
[7ffff6a0]    72657355  6c415c73  69737365  5355006f
[7ffff6b0]    414e5245  413d454d  7373656c  55006f69
[7ffff6c0]    44524553  49414d4f  4f525f4e  4e494d41
[7ffff6d0]    4f525047  454c4946  544f4e3d  4f4f4245
[7ffff6e0]    5355004b  4f445245  4e49414d  544f4e3d
```

```
User Stack [7ffff508]..[80000000]
[7ffff508]    0040003c  00400018
[7ffff510]    00000004  7ffff5f8
[7ffff520]    7ffff5e5  00000000
[7ffff530]    7fffff88  7fffff5f
[7ffff540]    7ffffebb  7ffffea5
```



Spazio occupato dal codice



Come possiamo vedere dallo screenshot, il codice occupa uno spazio che va da 00400024 a 004005d4, ovvero esattamente 1460 byte.

Questo conferma la correttezza della tabella. Infatti, questa mostra che lo spazio occupato dal codice è di 1460 byte.

Tempo complessivo di esecuzione dell'unità di controllo

Il calcolo del tempo di esecuzione del programma è basato sui dati in ingresso contenuti in:

- pendenzaIn.txt - file contente 319 caratteri, di cui 99 spazi, 10 numeri a tre cifre, 25 numeri a una cifra e 65 numeri a due cifre. Tra questi numeri 35 sono considerati negativi.
- sterzoIn.txt - file contente 298 caratteri, di cui 99 spazi, 10 numeri a tre cifre, 25 numeri a una cifra e 65 numeri a due cifre. Tra questi numeri 14 sono considerati negativi.
- distanzaIn.txt - file contenente 399 caratteri, di cui 99 spazi, 189 numeri e 111 caratteri (100 per il tipo di ostacolo e 11 per il valore di distanza).

Le procedure di controllo si suddividono in:

- pend – prende in ingresso i caratteri del file pendenzaIn.txt.
  Dei 100 valori in ingresso:
  - 70 sono corretti
  - 15 sono errati perché maggiori di 59
  - 15 sono errati perché minori di -59
- ster – prende in ingresso i caratteri del file sterzoIn.txt.
  Dei 100 valori in ingresso:
  - 10 sono errati perché maggiori di 99
  - 15 sono errati perché minori di 1
  - 16 sono errati perché differiscono di più di 10 gradi rispetto al valore precedente
  - 1 è corretto perche compreso tra 0 e 100 ed è il primo inserimento
  - 58 sono corretti perche differiscono di al massimo 10 gradi rispetto al valore precedente. Di questi, 29 hanno il valore precedente più grande, mentre i restanti 29 hanno il valore precedente più piccolo
- dist – prende in ingresso i caratteri del file distanzaIn.txt.
  Dei 100 valori in ingresso:
  - 2 sono corretti perché compresi tra 0 e 51 e sono i primi due valori
  - 20 valori sono errati perché non sono compresi tra 0 e 51
  - 10 valori sono errati perché il tipo d'ostacolo e la distanza dall'ostacolo sono uguali a quelli dei due valori precedenti
  - 68 valori sono corretti. Di questi: 30 hanno una distanza dall'ostacolo differente dal valore precedente, 18 hanno la distanza dall'ostacolo uguale ai due valori precedenti ma i tipi diversi, 10 hanno la distanza dall'ostacolo uguale ai due valori precedenti e il tipo uguale solo al precedente e 10 hanno valore di distanza uguale solo al valore di distanza precedente

Tabella di calcolo – Tempi di esecuzione del programma

Nome procedura	Istruzioni	Quantità	Tempo di esecuzione dell'istruzione	Numero di ripetizioni	Tempo di esecuzione totale
Main				1	
	jal	4	500	1	2000
	addi	2	500	1	1000
	sw	1	700	1	700
	lw	1	800	1	800
	li	1	500	1	500
	syscall	1	500	1	500
				Tempo di esecuzione totale nel programma	5500
apriL				3	
	li	3	500	1	1500
	move	1	500	1	500
	blt	1	1000	1	1000
	syscall	1	500	1	500
	jr	1	500	1	500
				Tempo di esecuzione totale nel programma	12000
apriS				6	
	li	3	500	1	1500
	move	1	500	1	500
	blt	1	1000	1	1000
	syscall	1	500	1	500
	jr	1	500	1	500
				Tempo di esecuzione totale nel programma	24000
leggi				3	
	li	2	500	1	1000
	la	1	1000	1	1000
	jr	1	500	1	500
				Tempo di esecuzione totale nel programma	7500
scrivi				6	
	li	1	500	1	500
	jr	1	500	1	500
				Tempo di esecuzione totale nel programma	6000
pulizia (chiamata da pend)				1	
inizio:	la	1	1000	1	1000
loop:	lw	1	800	81	64800
	beqz	1	500	81	40500
	sw	1	700	80	56000
	addi	1	500	80	40000
	j	1	500	80	40000
	jr	1	500	1	500

			Tempo di esecuzione totale nel programma		242800
pulizia (chiamata da ster)				1	
inizio:	la	1	1000	1	1000
loop:	lw	1	800	76	60800
	beqz	1	500	76	38000
	sw	1	700	75	52500
	addi	1	500	75	37500
	j	1	500	75	37500
	jr	1	500	1	500
			Tempo di esecuzione totale nel programma		227800
iniCon (chiamata da pend)				1	
inizio:	li	5	500	1	2500
	la	2	1000	1	2000
loop space:	lbu	1	800	99	79200
	beq	1	500	99	49500
	beqz	1	500	99	49500
	move	1	500	35	17500
	sub	2	500	35	35000
	sw	1	700	99	69300
	addi	2	500	99	99000
	li	2	500	99	99000
	bne	1	500	99	49500
loop segno negativo:	lbu	1	800	35	28000
	beq	2	500	35	35000
	bne	1	500	35	17500
	seq	1	2000	35	70000
	blt	1	1000	35	35000
	addi	1	500	35	17500
	j	1	500	35	17500
loop numero:	lbu	1	800	165	132000
	beq	2	500	165	165000
	bne	1	500	165	82500
	blt	1	1000	165	165000
	bgt	1	1000	165	165000
	add/addi	3	500	165	247500
	mul	1	1000	165	165000
	j	1	500	165	82500
fine:	lbu	1	800	1	800
	beq	2	500	1	1000
	beqz	1	500	1	500
	sw	1	700	1	700
	addi	2	500	1	1000
	li	2	500	1	1000
	bne	1	500	1	500
	jr	1	500	1	500
			Tempo di esecuzione totale nel programma		1983000
iniCon (chiamata da				1	

ster)					
inizio:	li	5	500	1	2500
	la	2	1000	1	2000
loop space:	lbu	1	800	99	79200
	beq	1	500	99	49500
	beqz	1	500	99	49500
	move	1	500	14	7000
	sub	2	500	14	14000
	sw	1	700	99	69300
	addi	2	500	99	99000
	li	2	500	99	99000
	bne	1	500	99	49500
loop segno negativo:	lbu	1	800	14	11200
	beq	2	500	14	14000
	bne	1	500	14	7000
	seq	1	2000	14	28000
	blt	1	1000	14	14000
	addi	1	500	14	7000
	j	1	500	14	7000
loop numero:	lbu	1	800	185	148000
	beq	2	500	185	185000
	bne	1	500	185	92500
	blt	1	1000	185	185000
	bgt	1	1000	185	185000
	add/addi	3	500	185	277500
	mul	1	500	185	92500
	j	1	500	185	92500
fine:	lbu	1	800	1	800
	beq	2	500	1	1000
	beqz	1	500	1	500
	sw	1	700	1	700
	addi	2	500	1	1000
	li	2	500	1	1000
	bne	1	500	1	500
	jr	1	500	1	500
				Tempo di esecuzione totale nel programma	1872700
Inisv (chiamata da dist)				1	
inizio:	li	4	500	1	2000
	la	3	1000	1	3000
while primo char:	lbu	1	800	100	80000
	beq	1	500	100	50000
	bne	1	500	100	50000
	sb	1	700	100	70000
	addi	3	500	100	150000
	j	1	500	100	50000
while, no primo char,	lb	1	800	189	151200
ma numero:	add/addi	3	500	189	283500
	beq	2	500	189	189000
	bne	1	500	189	94500
	blt	1	1000	189	189000

	mul	1	1000	189	189000
	j	1	500	189	94500
while, no primo char,	lb	1	800	11	8800
ma lettera:	add/addi	3	500	11	16500
	beq	2	500	11	11000
	bne	1	500	11	5500
	blt	1	1000	11	11000
	mul	1	500	11	5500
	j	2	500	11	11000
while, no primo char,	lb	1	800	99	79200
ma space:	addi	2	500	99	99000
	beq	3	500	99	148500
	bne	1	500	99	49500
	sw	1	700	99	69300
	li	2	500	99	99000
	j	1	500	99	49500
fine:	lb	1	800	1	800
	addi	1	500	1	500
	beq	2	500	1	1000
	sw	1	700	1	700
	jr	1	500	1	500
				Tempo di esecuzione totale nel programma	2310500
pend				1	
inizio:	addi	1	500	1	500
	sw	1	700	1	700
	la	3	1000	1	3000
	jal	3	500	1	1500
	move	1	500	1	500
	li	5	500	1	2500
loop valore corretto:	beq	1	500	70	35000
	lw	1	800	70	56000
	addi	3	500	70	105000
	bgt	1	1000	70	70000
	blt	1	1000	70	70000
	sb	2	700	70	98000
	j	2	500	70	70000
loop valore errato:	beq	1	500	30	15000
	lw	1	800	30	24000
	addi	3	500	30	45000
	bgt	1	1000	30	30000
	blt	1	1000	15	15000
	sb	2	700	30	42000
	j	1	500	30	15000
fine:	beq	1	500	1	500
	la	2	1000	1	2000
	jal	3	500	1	1500
	move	1	500	1	500
	li	1	500	1	500
	lw	1	800	1	800
	addi	1	500	1	500

	jr	1	500	1	500
				Tempo di esecuzione totale nel programma	705500
ster				1	
inizio:	addi	1	500	1	500
	sw	1	700	1	700
	la	3	1000	1	3000
	jal	3	500	1	1500
	move	1	500	1	500
	li	6	500	1	3000
loop primo inserimento	beq	1	500	1	500
corretto:	lw	1	800	1	800
	addi	3	500	1	1500
	ble	1	1000	1	1000
	bge	1	1000	1	1000
	sb	2	700	1	1400
	li	1	500	1	500
	move	1	500	1	500
	j	1	500	1	500
loop valore >99 && <1 :	beq	1	500	25	12500
	lw	1	800	25	20000
	addi	3	500	25	37500
	ble	1	1000	25	25000
	bge	1	1000	10	10000
	sb	2	700	25	35000
	move	1	500	25	12500
	j	1	500	25	12500
loop valore diff<=10 :	beq	1	500	58	29000
	lw	1	800	58	46400
	addi	3	500	58	87000
	ble	1	1000	58	58000
	bge	1	1000	58	58000
	bne	1	500	58	29000
	bgt	2	1000	58	116000
	sub	1	500	58	29000
	j	1	500	87	43500
	sb	2	700	58	81200
	move	1	500	58	29000
	j	1	500	58	29000
loop valore diff>10 :	beq	1	500	16	8000
	lw	1	800	16	12800
	addi	3	500	16	24000
	ble	1	1000	16	16000
	bge	1	1000	16	16000
	bne	1	500	16	8000
	bgt	2	1000	16	32000
	sub	1	500	16	8000
	j	1	500	24	12000
	sb	2	700	16	22400
	move	1	500	16	8000
	j	1	500	16	8000

fine:	beq	1	500	1	500
	la	2	1000	1	2000
	jal	3	500	1	1500
	move	1	500	1	500
	li	1	500	1	500
	lw	1	800	1	800
	addi	1	500	1	500
	jr	1	500	1	500
				Tempo di esecuzione totale nel programma	999000
dist				1	
inizio:	addi	1	500	1	500
	sw	1	700	1	700
	la	4	1000	1	4000
	move	1	500	1	500
	jal	4	500	1	2000
	li	5	500	1	2500
loop primi due valori	beq	2	500	2	2000
corretti:	lw	1	800	2	1600
	bgt	2	1000	2	4000
	j	1	500	2	1000
	sb	2	700	2	2800
	addi	4	500	2	4000
	j	1	500	2	1000
loop valore >50 $$ <1 :	beq	1	500	20	10000
	lw	1	800	20	16000
	bgt	1	1000	20	20000
	beq	1	500	10	5000
	sb	2	700	20	28000
	j	1	500	20	10000
	addi	4	500	20	40000
	j	1	500	20	10000
loop valore errato,	beq	2	500	10	10000
i due precedenti sono	lw/lb	6	800	10	48000
uguali:	bgt	2	1000	10	20000
	bne	4	500	10	20000
	sb	2	700	10	14000
	j	1	500	10	5000
	addi	4	500	10	20000
	j	1	500	10	5000
loop valore corretto,	beq	2	500	18	18000
distanza uguale, tipi	lw/lb	5	800	18	72000
diversi:	bgt	2	1000	18	36000
	bne	3	500	18	27000
	sb	2	700	18	25200
	addi	4	500	18	36000
	j	1	500	18	9000
loop valore corretto,	beq	2	500	30	30000
distanze diverse:	lw	2	800	30	48000
	bgt	2	1000	30	60000
	bne	1	500	30	15000

	sb	2	700	30	42000
	addi	4	500	30	60000
	j	1	500	30	15000
loop valore corretto,	beq	2	500	10	10000
distanze uguali, tipo	lw/lb	6	800	10	48000
del precedente uguale:	bgt	2	1000	10	20000
	bne	4	500	10	20000
	sb	2	700	10	14000
	addi	4	500	10	20000
	j	1	500	10	5000
loop valore corretto,	beq	2	500	10	10000
distanza uguale solo	lw	3	800	10	24000
al precedente	bgt	2	1000	10	20000
	bne	2	500	10	10000
	sb	2	700	10	14000
	addi	4	500	10	20000
	j	1	500	10	5000
fine:	beq	1	500	1	500
	la	2	1000	1	2000
	jal	2	500	1	1000
	move	1	500	1	500
	li	1	500	1	500
	lw	1	800	1	800
	addi	1	500	1	500
	jr	1	500	1	500
			Tempo di esecuzione totale nel programma		1047100
corp				1	
inizio:	addi	1	500	1	500
	sw	1	700	1	700
	la	4	1000	1	4000
	li	5	500	1	2500
loop:	beq	1	500	100	50000
	move	1	500	100	50000
	li	1	500	100	50000
	add/addi	8	500	100	400000
	lb	3	800	100	240000
	mul	1	500	100	50000
	lw	1	800	100	80000
	jr	1	500	100	50000
case:	sb	3	700	100	210000
	addi	3	500	100	150000
	j	1	500	100	50000
fine:	beq	1	500	1	500
	la	6	1000	1	6000
	jal	6	500	1	3000
	move	3	500	1	1500
	li	3	500	1	1500
	lw	1	800	1	800
	addi	1	500	1	500
	jr	1	500	1	500

	Tempo di esecuzione totale nel programma		1402000
Tempo complessivo di esecuzione del programma		in picosecondi	10845400
		in millisecondi	0,0108454
		in secondi	1,08454E-05

Durante lo studio del tempo complessivo di esecuzione dell'unità di controllo si è tenuto conto che alcune istruzioni sono pseudo-istruzioni.

Il tempo delle singole pseudo-istruzioni è stato calcolato sommando i tempi delle istruzioni che la compongono.

Ottimizzazione dell'unità di monitoraggio

- Come e quanto si possono diminuire i requisiti di memoria?
- Come e quanto si può diminuire il tempo complessivo di esecuzione?
- È possibile diminuire sia i requisiti di memoria che il tempo complessivo di esecuzione?

Come e quanto si possono diminuire i requisiti di memoria?

Una soluzione per ridurre i requisiti di memoria, può essere quella di eliminare gli arrays *arr* e *arr2.* In questo caso anche le procedure di conversione non sono più necessarie e otteniamo un ulteriore diminuzione dello spazio occupato dalle istruzioni.
Non è possibile fare un calcolo preciso per capire quanto spazio si è guadagnato ne assicurare che ci sia una riduzione dei requisiti di memoria.
Eliminare arr e arr2 comporta una modifica delle istruzioni nelle procedure di controllo che andrebbero ad occupare nuova memoria.
Il rischio è che le nuove istruzioni possano occupare lo spazio che si è liberato eliminando gli arrays *arr* e *arr2* e le procedure di conversione.

Come e quanto si può diminuire il tempo complessivo di esecuzione?

Una soluzione per ridurre il tempo complessivo di esecuzione consiste nell'utilizzare tre spazi di memoria buff1, buff2, buff3 al posto di buff.
Cosi possiamo salvare in buff1 i dati del file pendenzaIN.txt, in buff2 quelli del file sterzoIN.txt e in buff3 quelli di distanzaIN.txt.
Questo ci permette di eliminare la procedura pulizia e aggiungendo alcune istruzioni alla procedura *leggi* possiamo ottenere un tempo complessivo di esecuzione più veloce di circa 227800 ps.

È possibile diminuire sia i requisiti di memoria che il tempo complessivo di esecuzione?

Utilizzando la soluzione per diminuire il tempo complessivo di esecuzione presentata sopra, possiamo notare che non è possibile ottenere una riduzione dei requisiti di memoria, perché si rendono necessari altri 1024 byte per contenere tutti i dati in ingresso.

Un'altra soluzione, come mostrato sopra, può essere quella di non utilizzare i due arrays *arr* e *arr2* e fare in modo che le procedure di controllo lavorino

direttamente con i char presenti su buff. In questo modo otteniamo una riduzione dei requisiti di memoria.
Le procedure di conversione a questo punto non sono più necessarie e otteniamo una riduzione del tempo di esecuzione e un ulteriore guadagno di memoria perché le istruzioni vengono eliminate.
Questo però, non ci assicura che le procedure di controllo non subiscano un aumento del tempo di esecuzione e un aumento della memoria utilizzata dovuto alla necessita di maggiori istruzioni.
In altre parole, se il tempo di esecuzione guadagnato eliminando le procedure di conversione viene perso aggiungendo troppe istruzioni alle procedure di controllo, non abbiamo modo di diminuire il tempo di esecuzione totale.
Quindi, c'è il rischio che con questo approccio non si ottiene ne un guadagno in termini di memoria, ne un minor tempo di esecuzione.

Codice MIPS assembly implementato

```
#FONTANI ALESSIO - alessio.fontani@stud.unifi.it - Consegnato il 18/05/2018

.data

fnf:    .asciiz "File not found: "

pin:    .asciiz "pendenzaIN.txt"

sin:    .asciiz "sterzoIN.txt"

din:    .asciiz "distanzaIN.txt"

pout:   .asciiz "correttezzaPendenzaOUT.txt"

sout:   .asciiz "correttezzaSterzoOUT.txt"

dout:   .asciiz "correttezzaDistanzaOUT.txt"

puno:   .asciiz "correttezzaP1.txt"

pdue:   .asciiz "correttezzaP2.txt"

ptre:   .asciiz "correttezzaP3.txt"

jtab:   .word case0, case1, case2, case3

buff:   .space 512

arr:    .space 400

arr2:   .space 100

out1:   .space 199

out2:   .space 199

out3:   .space 199


.text

.globl main

main:   addi $sp, $sp, -4       #il frame di stack e' di 4 byte
        sw $ra, ($sp)           #salva l'indirizzo di ritorno
        jal pend                #jal alla prima procedura
        jal ster                #jal alla seconda procedura
        jal dist                #jal alla terza procedura
        jal corp                #jal alla quarta ed ultima procedura
        lw $ra, ($sp)           #ripristina l'indirizzo di ritorno
        addi $sp, $sp, 4        #rimuove il frame di stack
fine:   li $v0, 10              #inserisco in v0 il codice di chiusura programma
        syscall                 #chiude il programma

#APERTURA FILE PER LETTURA

apriL:  li $v0, 13              #inserisco in v0 il codice per aprire file
        li $a1, 0               #read-only flag
        li $a2, 0               #(ignored)
        syscall
        move $t0, $v0           #sposto il descrittore in t0
        blt $t0,0,err1          #se il file non esiste restituisce errore
        jr $ra                  #torna al chiamante
```

```
#APERTURA FILE PER SCRITTURA

apriS:  li $v0, 13              #inserisco in v0 il codice per aprire file
        li $a1, 1               #write-only flag
        li $a2, 0               #(ignored)
        syscall
        move $t0, $v0           #sposto il descrittore in t0
        blt $t0,0,err1          #se il file non esiste va ad err1
        jr $ra                  #torna al chiamante

#LETTURA FILE

leggi:  li $v0, 14              #inserisco in v0 il codice per la lettura del file
        la $a1, buff            #scelgo dove mettere i dati
        li $a2, 508             #lunghezza del buff
        syscall
        jr $ra                  #torna al chiamante

#SCRITTURA FILE

scrivi: li $v0, 15              #inserisco in v0 il codice di scrittura su file
        syscall
        jr $ra                  #torna al chiamante

#ERRORE FILE NON TROVATO

err1:   li $v0, 4               #inserisco in v0 il codice per il print-string
        move $t0, $a0
        la $a0, fnf             #metto in a0 l'indirizzo al messaggio di errore
        syscall                 #stampa il messaggio d'errore
        move $a0, $t0           #metto in a0 l'indirizzo al nome del file mancante
        syscall                 #stampa il file mancante
        j fine                  #jump a fine

#PULIZIA BUFFER

pulizia:la $t0, buff            #carico l'indirizzo di buff
rin:    lw $t1, ($t0)           #mette in t1 una word del buff
        beqz $t1, back          #se t1 e' null esce dal loop (rin)
        sw $zero, ($t0)         #mette null nell'indirizzo t0
        addi $t0, $t0, 4        #aumenta l'indirizzo
        j rin                   #jump a rin
back:   jr $ra                  #torna all'indirizzo chiamante


#CONVERSIONE DA CHAR AD INTEGER

iniCon: la $s3, buff            #carica in s3 l'indirizzo di buff
        la $s5, arr             #carica in s5 l'indirizzo di arr
        li $t6, 0               #variabile usata per controllare se un numero e'
                                #negativo
        li $s2, 0               #inizializzo s2, variabile contenente l'intero ottenuto
                                #dalla conversione
        li $t3, 32              #valore corrispondente a SPACE
        li $t4, 45              #valore corrispondente a -
        li $t5, 10              #valore di moltiplicazione
loop:   lbu $t1, ($s3)          #carica in t1 un carattere del buffer
        beq $t1, $t3 , FIN      #se il carattere e' SPACE salta a FIN
        beq $t1, $zero, FIN     #se il carattere e' NULL salta a FIN
        bne $t1, $t4, GO        #se non e' un meno salta a GO
        seq $t6, $t1, $t4       #mette t6 ad 1, cosi che capisco che devo convertire il
                                #numero in un negativo
GO:     blt $t1, 48, error      #controlla che il carattere sia un numero(ascii<'0'),
                                #se non lo e' va ad error
        bgt $t1, 57, error      #controlla che il carattere sia un numero(ascii>'9'),
                                #se non lo e' va ad error
        addi $t1, $t1, -48      #converte il numero da char a decimale
```

```
        mul $s2, $s2, $t5       #moltiplica per 10 il valore in s2
        add $s2, $s2, $t1       #somma la nuova unita', che in presenza di nuova cifra,
                                #viene moltiplicata per 10 al prossimo giro
        addi $s3, $s3, 1        #incremento indirizzo
        j loop                  #jump per riiniziare il loop
FIN:    beqz $t6, cont          #se il numero e' positivo salta, altrimenti
        move $s4, $s2           #trasforma il numero da positivo a negativo
        sub $s2, $s2,$s4        #facendo sostanzialmente x=x-2x
        sub $s2, $s2,$s4
cont:   sw $s2, ($s5)           #mette il numero nell'array
        addi $s5, $s5, 4        #aumenta l'indirizzo dell'array
        addi $s3, $s3, 1        #aumenta l'indirizzo del buffer
        li $s2, 0               #pulisce s2 per il numero successivo
        li $t6, 0               #la variabile di controllo dei num. negativi torna a 0
        bne $t1, $zero, loop    #se t1 e' zero significa che gli input sono finiti ed
                                #esegue il ritorno
        jr $ra                  #altrimenti riinizia il loop

error:  addi $s3, $s3, 1        #aumenta l'indirizzo del buffer dopo aver trovato un
                                #char che non e' un numero
        j loop                  #jump a loop


#SALVATAGGIO TIPO E DISTANZA

inisv:  li $t0, 0               #variabile di controllo primo char
        li $s0, 0               #inizializzazione di s0
        li $t6, 32              #valore corrispondente a SPACE
        li $t7, 16              #valore di moltiplicazione
        la $t2, buff            #carico l'indirizzo di buff in t2
        la $t4, arr2            #carico l'indirizzo di arr2 in t4
        la $t5, arr             #carico l'indirizzo di arr in t5
while:  lb $t3, ($t2)           #carico in t3 un char del buff
        addi $t2, $t2, 1        #aumento l'indirizzo
        beq $t3, $zero, AZZ     #se e' fine buffer, va ad esc
        bne $t0, $zero, DOV     #salta a DOV se non e' il primo char
        sb $t3, ($t4)           #mette il tipo in arr2
        addi $t4, $t4, 1        #aumenta l'indirizzo in t4
        addi $t0, $t0, 1        #mette la variabile di controllo ad 1
        j while                 #jump a while
DOV:    beq $t3, $t6, AZZ       #se il char e' uno SPACE va ad AZZ
        blt $t3, 65, NUM        #salta se il carattere e' un numero
        addi $t3, $t3, -55      #trova il decimale corrispondente al char lettera
        j AVA                   #continua da AVA
NUM:    addi $t3, $t3, -48      #trova il decimale corrispondente al char numero
AVA:    mul $s0, $s0, $t7       #moltiplica per 16 il valore in s0
        add $s0, $s0, $t3       #somma la nuova unita', che in presenza di nuova cifra,
                                #viene moltiplicata per 16 al prossimo giro
        j while                 #jump a while2
AZZ:    sw $s0, ($t5)           #mette il valore della distanza in arr
        beq $t3, $zero, ESC     #se il buffer e' finito esce
        li $t0, 0               #mette 0 in t0, facendo capire alla procedura che il
                                #prossimo char e' un primo char
        addi $t5, $t5, 4        #aumenta l'indirizzo di arr
        li $s0, 0               #pulisce s0 per il numero successivo
        j while                 #jump a while2
ESC:    jr $ra                  #torna all'indirizzo chiamante

#CONTROLLO PENDENZA

pend:   addi $sp, $sp, -4       #alloca 4 byte nello stack
        sw $ra, 0($sp)          #salva l'indirizzo di ritorno
        la $a0, pin             #Imposto pendenzaIn.txt come file da aprire
        jal apriL               #Chiamo la procedura per aprire un file in readonly
        move $a0, $v0           #Metto in $a0 il descrittore per leggere il file
        jal leggi               #Chiamo la procedura per leggere il file, mette i dati
                                #in buff
```

```
        jal iniCon              #Chiamo la procedura per la conversione

        li $t1, 0               #contatore per concludere il loop
        li $t7, 32              #variabile contenente il decimale corrispondente allo
                                #spazio
        li $t8, 49              #variabile contenente il decimale corrispondente all' 1
        li $t9, 48              #variabile contenente il decimale corrispondente allo 0
        li $t5, 100             #variabile controllo fine array
        la $t2, arr             #carico l'indirizzo dell'array
        la $t4, out1            #carico l'indirizzo in cui inserire i valori di output
loopP:  beq $t1, $t5, prin1     #quando arriva al centesimo valore, va a prin1 per
                                #stampare out1 e quindi esce dal loopP
        lw $t3, ($t2)           #mette in t3 i valori dell'array
        addi $t2, $t2, 4        #aumenta l'indirizzo dell'array
        bgt $t3, 59, false      #se t3 e' maggiore di 59 salta a false
        blt $t3, -59,false      #se t3 e' minore di -59 salta a false
        sb $t8, ($t4)           #mette uno nei dati in output
        j true                  #jump all'etichetta true
false:  sb $t9, ($t4)           #mette zero nei dati in output
true:   sb $t7, 1($t4)          #mette uno spazio dopo il valore di correttezza
        addi $t4, $t4, 2        #aumenta l'indirizzo di out1
        addi $t1, $t1, 1        #aumenta il contatore( il contatore conclude il ciclo
                                #quando arriva a 100)
        j loopP                 #riparte il loopP

prin1:  la $a0, pout            #carica in a0 il file da aprire
        jal apriS               #chiama la procedura di apertura del file, write-only
        move $a0, $v0           #mette il descrittore in a0
        la $a1, out1            #mette in a1 l'indirizzo di out1, out1 viene scritto
                                #nel file
        li $a2, 199             #decide lo spazio da riservare
        jal scrivi              #richiama la procedura di scrittura
        jal pulizia             #richiama la procedura di pulizia di buff
        lw $ra, ($sp)           #ripristina l'indirizzo di ritorno
        addi $sp, $sp, 4        #rimuove lo stack frame
        jr $ra                  #jump alla procedura chiamante

#CONTROLLO STERZO

ster:   addi $sp, $sp, -4       #alloca 4 byte nello stack
        sw $ra, ($sp)           #salva l'indirizzo di ritorno
        la $a0, sin             #Imposto sterzoIn.txt come file da aprire
        jal apriL               #Chiamo la procedura per aprire un file in readonly
        move $a0, $v0           #Metto in $a0 il descrittore per leggere il file
        jal leggi               #Chiamo la procedura per leggere il file, mette i dati
                                #in buff
        jal iniCon              #Chiamo la procedura per la conversione

        li $t0, 0               #inizializzo t0 a 0, e' il contatore per uscire prima
                                #della fine dell'array
        li $t1, 100             #variabile controllo fine array
        la $t3, arr             #carico l'indirizzo dell'array
        la $t5, out2            #carico l'indirizzo in cui inserire i valori in output
        li $t6, 0               #variabile che, quando e' a zero, significa che il loop
                                #e' al suo primo ciclo
        li $s0, 32              #variabile contenente il decimale corrispondente allo
                                #spazio
        li $s1, 48              #variabile contenente il decimale corrispondente allo
                                #zero
        li $s2, 49              #variabile contenente il decimale corrispondente all'1
loop1:  beq $t0, $t1, prin2     #esce dal loop1 se si e' raggiunto l'ultimo valore
        lw $t4, ($t3)           #carica in t4 i valori in arr
        addi $t3, $t3, 4        #aumenta l'indirizzo di arr
        ble $t4, $zero, false2  #salta se il valore e' minore di 1
        bge $t4, $t1, false2    #salta se il valore e' maggiore di 99
        bne $t6, $zero, check   #se non e' il primo inserimento salta a check
        sb $s2, ($t5)           #mette 1 nell'array in uscita (out2)
```

```
        li $t6, 1               #mette ad 1 t6, cosi capiamo che il primo valore e'
                                #stato letto
        j true2                 #jump a true2
check:  bgt $t7, $t4, cas2      #se t7 e' maggiore di t4 va a cas2
        sub $s5, $t4, $t7       #mette il risultato di t4-t7 in s5
        j check2                #jump a check2
cas2:   sub $s5, $t7, $t4       #mette il risultato di t7-t4 in s5
check2: bgt $s5, 10, false2     #se s5>10 va a false
        sb $s2, ($t5)           #mette 1 nell'array in uscita (out2)
        j true2                 #jump a true2
false2: sb $s1, ($t5)           #mette 0 nell'array in uscita (out2)
true2:  sb $s0, 1($t5)          #mette uno spazio nell'array in uscita (out2)
        addi $t5, $t5, 2        #aumenta l'indirizzo di out2
        addi $t0, $t0, 1        #aumenta il contatore (che quando arriva a 100 esce dal
                                #loop)
        move $t7, $t4           #mette in t7 il valore in t4 per il controllo
                                #successivo, t7 sara' il valore precedente
        j loop1                 #jump a loop1

prin2:  la $a0, sout            #carica in a0 il file da aprire
        jal apriS               #chiama la procedura di apertura del file, write-only
        move $a0, $v0           #mette il descrittore in a0
        la $a1, out2            #mette in a1 l'indirizzo di out2, out2 viene scritto
                                #nel file
        li $a2, 199             #decide lo spazio da riservare
        jal scrivi              #richiama la procedura di scrittura
        jal pulizia             #richiama la procedura di pulizia di buff
        lw $ra, ($sp)           #ripristina l'indirizzo di ritorno
        addi $sp, $sp, 4        #rimuove lo stack frame
        jr $ra                  #jump alla procedura chiamante

#CONTROLLO DISTANZA

dist:   addi $sp, $sp, -4       #Alloca 4 byte nello stack
        sw $ra, ($sp)           #Salva l'indirizzo di ritorno
        la $a0, din             #Imposto distanzaIn.txt come file da aprire
        jal apriL               #Chiamo la procedura per aprire un file in readonly
        move $a0, $v0           #Metto in $a0 il descrittore per leggere il file
        jal leggi               #Chiamo la procedura per leggere il file, mette i dati
                                #in buff
        jal inisv               #Chiamo la procedura di salvataggio tipo di ostacolo

        li $t0, 100             #variabile controllo fine array
        li $t6, 0               #contatore fine array
        li $s3, 48              #variabile con valore decimale corrispondente al CHAR 0
        li $s4, 49              #variabile con valore decimale corrispondente al CHAR 1
        li $s5, 32              #variabile con valore decimale corrispondente al CHAR
SPACE
        la $s0, arr             #carico in s0 l'indirizzo di arr
        la $s1, arr2            #carico in s1 l'indirizzo di arr2
        la $s2, out3            #carico in s2 l'indirizzo di out3
loop3:  beq $t6, $t0, prin3     #se l'array e' finito va a prin3
        lw $t5, ($s0)           #carico in t5 il valore all'indirizzo s0
        bgt $t5, 50, false3     #salta a false3 se il valore e' maggiore di 50
        beq $t5, $zero, false3  #salta a false3 se il valore e' 0
        bgt $t6, 1, CAS         #se siamo alla terza cifra, salta a CAS
        j true3                 #jump a true3
CAS:    lw $t4, -4($s0)         #mette il valore precedente a quello corrente, in t4
        bne $t5, $t4, true3     #se i valori sono diversi, salta a true3
        lw $t3, -8($s0)         #carica il valore due posizioni piu indietro di quello
                                #corrente, in t3
        bne $t3, $t5, true3     #se i valori sono diversi, salta a true3
        lb $t4, -1($s1)         #mette il tipo del valore precedente a quello corrente
                                #in t4
        lb $t3, ($s1)           #mette il tipo del valore corrente in t3
        bne $t3, $t4, true3     #se i tipi sono diversi salta a true3
        lb $t4, -2($s1)         #mette il tipo del valore due posizioni piu indietro di
```

```
                                #quello corrente, in t4
        bne $t3, $t4, true3     #se i tre valori precedenti sono diversi, salta a true3
false3: sb $s3, ($s2)           #mette zero come risultato in out3
        j GO3                   #jump a GO3
true3:  sb $s4, ($s2)           #mette uno come risultato in out3
GO3:    sb $s5, 1($s2)          #mette SPACE in out3+1
        addi $s2, $s2, 2        #aumenta l'indirizzo di out3
        addi $s0, $s0, 4        #aumenta l'indirizzo di arr
        addi $s1, $s1, 1        #aumenta l'indirizzo di arr2
        addi $t6, $t6, 1        #aumenta il contatore
        j loop3                 #jump a loop3

prin3:  la $a0, dout            #carica in a0 il file da aprire
        jal apriS               #chiama la procedura di apertura del file, write-only
        move $a0, $v0           #mette il descrittore in a0
        la $a1, out3            #mette in a1 l'indirizzo di out3, out3 viene scritto
                                #nel file
        li $a2, 199             #decide lo spazio da riservare
        jal scrivi              #richiama la procedura di scrittura
        lw $ra, ($sp)           #ripristina l'indirizzo di ritorno
        addi $sp, $sp, 4        #rimuove il frame di stack
        jr $ra                  #jump alla procedura chiamante

corp:   addi $sp, $sp, -4       #alloca 4 byte nello stack
        sw $ra, ($sp)           #salva l'indirizzo di ritorno
        la $t0, out1            #carico in t0 l'indirizzo di t
        la $t1, out2            #carico in t1 l'indirizzo di out2
        la $t2, out3            #carico in t2 l'indirizzo di out3
        la $s4, jtab            #carico in s4 l'indirizzo della jumptable
        li $t8, 0               #contatore per controllo fine array
        li $s5, 48              #variabile con il decimale corrispondente a CHAR 0
        li $s6, 49              #variabile con il decimale corrispondente a CHAR 1
        li $t3, 100             #variabile controllo fine array
        li $t4, 4               #valore di moltiplicazione
loop4:  beq $t8, $t3, prin4     #se l'array e' finito va a prin4
        li $t6, 0               #inizializzo la variabile t6
        move $s7, $s4           #metto in s7 l'indirizzo della jump table
        addi $t8, $t8, 1        #aumento il contatore
        lb $s0, ($t0)           #metto in s0 il valore all'indirizzo $t0
        lb $s1, ($t1)           #metto in s1 il valore all'indirizzo $t1
        lb $s2, ($t2)           #metto in s2 il valore all'indirizzo $t2
        addi $s0, $s0, -48      #trasformo il valore in s0 in decimale, per poter
                                #effettuare la somma
        addi $s1, $s1, -48      #trasformo il valore in s1 in decimale, per poter
                                #effettuare la somma
        addi $s2, $s2, -48      #trasformo il valore in s2 in decimale, per poter
                                #effettuare la somma
        add $t6, $t6, $s0       #sommo in t6 il valore di correttezza presente in s0
        add $t6, $t6, $s1       #sommo in t6 il valore di correttezza presente in s1
        add $t6, $t6, $s2       #sommo in t6 il valore di correttezza presente in s2
        mul $t6, $t6, $t4       #moltiplico per 4 il valore in t6
        add $s7, $s7, $t6       #metto in s7 il valore corrispondente all'indirizzo
                                #contenente l'indirizzo a cui saltare
        lw $s7, ($s7)           #metto in s7 l'indirizzo a cui saltare
        jr $s7                  #jump all'indirizzo contenuto in s7

case0:  sb $s5, ($t2)           #metto 0 in out3 (che corrisponde a correttezzaP3)
        sb $s5, ($t1)           #metto 0 in out2 (che corrisponde a correttezzaP2)
        sb $s5, ($t0)           #metto 0 in out1 (che corrisponde a correttezzaP1)
        addi $t0, $t0, 2        #aumento l'indirizzo
        addi $t1, $t1, 2        #aumento l'indirizzo
        addi $t2, $t2, 2        #aumento l'indirizzo
        j loop4                 #jump a loop4

case1:  sb $s6, ($t2)           #metto 1 in out3 (che corrisponde a correttezzaP3)
        sb $s5, ($t1)           #metto 0 in out2 (che corrisponde a correttezzaP2)
        sb $s5, ($t0)           #metto 0 in out1 (che corrisponde a correttezzaP1)
```

```
        addi $t0, $t0, 2        #aumento l'indirizzo
        addi $t1, $t1, 2        #aumento l'indirizzo
        addi $t2, $t2, 2        #aumento l'indirizzo
        j loop4                 #jump a loop4

case2:  sb $s6, ($t2)           #metto 1 in out3 (che corrisponde a correttezzaP3)
        sb $s6, ($t1)           #metto 1 in out2 (che corrisponde a correttezzaP2)
        sb $s5, ($t0)           #metto 0 in out1 (che corrisponde a correttezzaP1)
        addi $t0, $t0, 2        #aumento l'indirizzo
        addi $t1, $t1, 2        #aumento l'indirizzo
        addi $t2, $t2, 2        #aumento l'indirizzo
        j loop4                 #jump a loop4

case3:  sb $s6, ($t2)           #metto 1 in out3 (che corrisponde a correttezzaP3)
        sb $s6, ($t1)           #metto 1 in out2 (che corrisponde a correttezzaP2)
        sb $s6, ($t0)           #metto 1 in out1 (che corrisponde a correttezzaP1)
        addi $t0, $t0, 2        #aumento l'indirizzo
        addi $t1, $t1, 2        #aumento l'indirizzo
        addi $t2, $t2, 2        #aumento l'indirizzo
        j loop4                 #jump a loop4

prin4:  la $a0, puno            #carica in a0 il file da aprire
        jal apriS               #chiama la procedura di apertura del file, write-only
        move $a0, $v0           #mette il descrittore in a0
        la $a1, out1            #mette in a1 l'indirizzo di out1, out1 viene scritto
                                #nel file
        li $a2, 199             #decide lo spazio da riservare
        jal scrivi              #richiama la procedura di scrittura
        la $a0, pdue            #carica in a0 il file da aprire
        jal apriS               #chiama la procedura di apertura del file, write-only
        move $a0, $v0           #mette il descrittore in a0
        la $a1, out2            #mette in a1 l'indirizzo di out2, out2 viene scritto
                                #nel file
        li $a2, 199             #decide lo spazio da riservare
        jal scrivi              #richiama la procedura di scrittura
        la $a0, ptre            #carica in a0 il file da aprire
        jal apriS               #chiama la procedura di apertura del file, write-only
        move $a0, $v0           #mette il descrittore in a0
        la $a1, out3            #mette in a1 l'indirizzo di out3, out3 viene scritto
                                #nel file
        li $a2, 199             #decide lo spazio da riservare
        jal scrivi              #richiama la procedura di scrittura
        lw $ra, ($sp)           #ripristina l'indirizzo di ritorno
        addi $sp, $sp, 4        #rimuove il frame di stack
        jr $ra                  #ritorna alla procedura chiamante
```