

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

Informazioni sull' Autore:

- Andrea Bartilucci, mat. 5645760
andreabartilucci@stud.unifi.it

Data di Consegna:

- 03 Luglio 2018

Descrizione della soluzione adottata

Il design della soluzione adottata prevede l'esecuzione, nel processo principale, di due procedure : la prima ha il compito di "accendere" l'unità di controllo, lanciando per la prima volta le procedure di analisi e computazione, la seconda ha invece il compito di proseguire il calcolo, chiamando le procedure precedenti che tuttavia riprendono dall'ultimo valore letto.

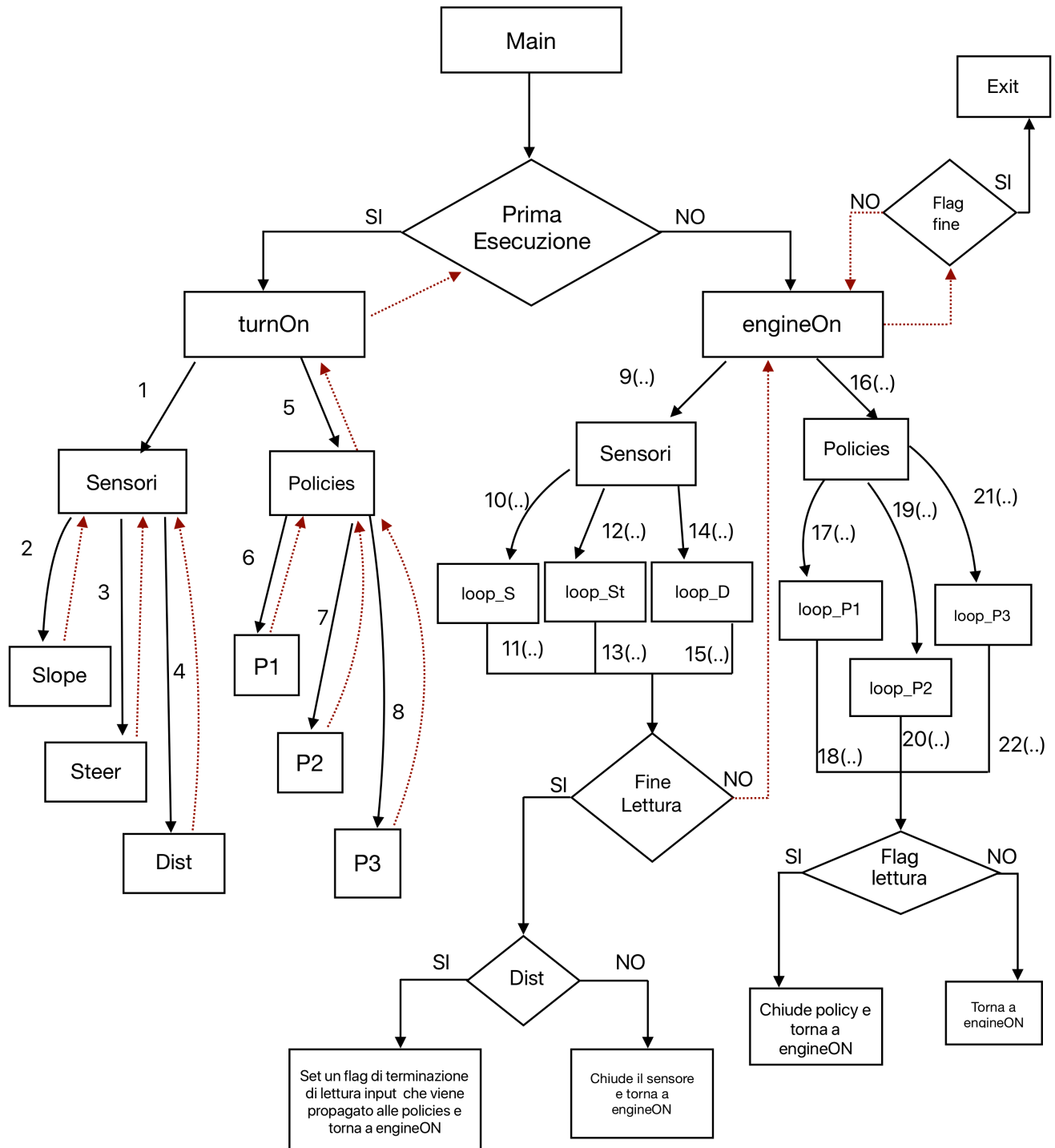
Durante la fase di progettazione era stato elaborato inizialmente un design che prevedesse una serie di chiamate annidate tra le varie procedure, tuttavia al fine di favorire una maggior modalità e ridurre le probabilità di errori dovuti al salvataggio e ripristino dei registri tra le varie chiamate è stato scelto un approccio diverso, che permettesse maggior leggibilità del codice e potesse essere esteso, con poche modifiche della procedura principale, anche ad altri sensori e/o specifiche.

Per la lettura dei file di input si è resa necessaria l'allocazione statica di memoria come buffer di lettura, di dimensioni diverse rispetto al sensore considerato, in quanto differenti le specifiche.

La struttura dati utilizzata principalmente per la conservazione dei dati tra le chiamate è lo Stack, per poter ripristinare i registri necessari a riprendere l'esecuzione dal punto in cui si era interrotta.

L'esecuzione del programma prevede l'analisi di un input per istante di tempo per ciascun sensore, input che viene elaborato e, in base alle specifiche proprie del sensore, viene scritto il valore booleano in un array, il cui indirizzo di partenza viene propagato a tutte le procedure di elaborazione di sensori e politiche di aggregazione. Il controllo passa poi alle funzioni di elaborazione delle policies, che al termine dell'esecuzione tornano al main, punto da cui viene ripreso il controllo del primo sensore.

Flow-Chart



Descrizione delle strutture dati utilizzate

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

- **slope_IN_buffer** : buffer di lettura di massimo 400 byte per il sensore di pendenza, spazio calcolato con la formula (Possibili valori ammessi da specifiche compreso il segno ' - ' = 3 + spazio = 4 * 100 rilevamenti = 400).
- **slope_IN_path** : percorso in cui è memorizzato il file .txt da analizzare per il sensore di pendenza.
- **steer_IN_buffer** : buffer di lettura di massimo 400 byte per il sensore di sterzo, spazio calcolato con la formula (Possibili valori ammessi da specifiche = 2 + spazio = 3* 100 rilevamenti = 300).
- **steer_IN_path** : percorso in cui è memorizzato il file .txt da analizzare per il sensore di sterzo.
- **dist_IN_buffer** : buffer di lettura di massimo 400 byte per il sensore di distanza, spazio calcolato con la formula (Possibili valori ammessi da specifiche = 3 + spazio = 4* 100 rilevamenti = 300).
- **dist_IN_path** : percorso in cui è memorizzato il file .txt da analizzare per il sensore di distanza.
- **slope_OUT_buffer** : buffer di scrittura di 200 byte per il sensore di pendenza, spazio calcolato con la formula (Possibili valori ammessi = {0,1} = 1 + spazio = 2 * 100 rilevamenti = 200).
- **slope_IN_path** : percorso in cui è memorizzato il file .txt da scrivere per il sensore di pendenza.
- **steer_IN_buffer** : buffer di scrittura di 200 byte per il sensore di sterzo, spazio calcolato con la formula (Possibili valori ammessi = {0,1} = 1 + spazio = 2 * 100 rilevamenti = 200).
- **steer_IN_path** : percorso in cui è memorizzato il file .txt da scrivere per il sensore di sterzo.
- **dist_IN_buffer** : buffer di scrittura di 200 byte per il sensore di distanza, spazio calcolato con la formula (Possibili valori ammessi = {0,1} = 1 + spazio = 2 * 100 rilevamenti = 200).
- **dist_IN_path** : percorso in cui è memorizzato il file .txt da scrivere per il sensore di distanza.

- **pol_1_buffer** : buffer di scrittura di 200 byte per la Policy 1, spazio calcolato con la formula (Possibili valori ammessi = $\{0,1\} = 1 + \text{spazio} = 2 * 100$ rilevamenti = 200).
- **pol_1_path** : percorso in cui è memorizzato il file .txt da scrivere per il sensore di pendenza.
- **pol_2_buffer** : buffer di scrittura di 200 byte per la Policy 2, spazio calcolato con la formula (Possibili valori ammessi = $\{0,1\} = 1 + \text{spazio} = 2 * 100$ rilevamenti = 200).
- **pol_2_path** : percorso in cui è memorizzato il file .txt da scrivere per il sensore di sterzo.
- **pol_3_buffer** : buffer di scrittura di 200 byte per la Policy 3, spazio calcolato con la formula (Possibili valori ammessi = $\{0,1\} = 1 + \text{spazio} = 2 * 100$ rilevamenti = 200).
- **pol_3_path** : percorso in cui è memorizzato il file .txt da scrivere per il sensore di distanza.
- **correctness_array** : array di 3 elementi riservato per la memorizzazione della variabile booleana che esprime la correttezza di ogni sensore.

Descrizione delle procedure utilizzate

- **main** : procedura principale, assegna al registro \$t9 l'indirizzo di memoria di correctness_array.
- **turnOn** : procedura di primo avvio del programma, assegna al registro \$a1 indirizzo di correctness_array da passare alle procedure di elaborazione dei sensori, rispettivamente incrementato in modo da puntare alla locazione corretta. Chiama poi le procedure di elaborazione delle policies.
- **engineOn** : procedura di recupero del programma a partire dalla lettura successiva, chiama le procedure di recupero di ogni sensore.
Al termine della chiamata loop_D salva il registro restituito \$v0 in \$a3 che indica l'eventuale terminazione della lettura dei dati in input (assumendo lo stesso numero di input, al termine della lettura dell'ultimo input dell'ultimo sensore, in base all'ordine scelto, è possibile terminare le letture) e propaga tale registro alle procedure di recupero delle policies.
Nel caso in cui l'ultima policies restituisca come risultato in \$v0 il valore zero, il programma si arresta.
Altrimenti itera.

- **slope_S** : Alloca stack di 24 byte, salva indirizzo di ritorno e indirizzo array di correttezza su registro temporaneo \$t1.
Esegue le procedure di apertura del file di input e di creazione del file di output, al termine delle quali passa al controllo del primo dato in input.
- **loop_SI** : Alloca stack di 24 byte per puntare all'area di memoria di competenza, salva indirizzo di ritorno e ripristina da SF i registri precedentemente utilizzati.
Effettua controllo per verificare eventuale terminazione del file di input (\$t3, numero di elementi residui da leggere pari a zero) e carica sul registro \$s2 il primo byte da leggere.
- **sign_check** : controlla che il byte contenuto su \$s2 non sia uno spazio, in caso lo sia controlla che non ve ne siano due consecutivi, altrimenti stampa messaggio di carattere sconosciuto.
Se invece si tratta di uno spazio passa il controllo a num_check
- **num_check** : controlla che il byte contenuto su \$s2 appartenga al range, in ASCII, del dizionario di interesse { 0,1,..,9}, in caso contrario stampa messaggio di carattere sconosciuto.
- **first_val** : verifica se il byte contenuto su \$s2, ovvero il primo numero intero del dato in input, è maggiore del numero 5 in ASCII : nel caso lo fosse, passa il controllo a sec_val, altrimenti analizza due byte successivi a quello attuale; date le specifiche per cui il sensore è funzionante in un range (-60, 60), se il byte è uno spazio si ha sensore funzionante, altrimenti abbiamo una lettura di un input formato da almeno in 3 interi, pertanto si ha sensore malfunzionante.
- **sec_val** : nel caso di byte su \$s2 > 5, controlla il byte successivo : se è uno spazio si ha una lettura di un input formato da un solo intero, pertanto sensore funzionante; viceversa, il sensore non sarà funzionante perché l'input è fuori range.
Inoltre è stato aggiunto un ulteriore controllo di terminazione, qualora il byte contenuto su \$s2 sia l'elemento di fine file (null).
- **slope_NW** : carica su registro \$t4 il valore booleano 0 codificato in ASCII e chiama write_slope.
- **slope_W** : carica su registro \$t4 il valore booleano 1 codificato in ASCII e chiama write_slope.

- **write_slope** : memorizza il byte contenuto in \$t4 all'indirizzo di memoria puntato da \$t1 (indirizzo correctness_array) e all'indirizzo di memoria puntato da \$s7 (indirizzo del buffer di uscita per la scrittura su file).
Chiama procedura di scrittura sensore e spazio, incrementando opportunamente indirizzo del buffer di uscita.
- **to_next** : procedura che scorre l'input finché non trova un byte diverso dallo spazio, una volta trovato salva su SF i registri utilizzati , carica indirizzo di ritorno del chiamante, dealloca lo stack e ritorna al chiamante.
- **steer_S** : alloca stack di 52 byte (24 utilizzati per memorizzare i registri utilizzati da slope_S, i restanti per il sensore in analisi).
Esegue le procedure di apertura del file di input e di creazione del file di output, al termine delle quali passa al controllo del primo dato in input.
- **loop_St** : Alloca stack di 52 byte per puntare all'area di memoria di competenza, salva indirizzo di ritorno e ripristina da SF i registri precedentemente utilizzati.
Effettua controllo per verificare eventuale terminazione del file di input (\$t3, numero di elementi residui da leggere pari a zero) e carica sul registro \$s2 il primo byte da leggere.
Carica su \$t6 il valore 1, usato come flag per la prima esecuzione per la verifica della specifica $|s(t) - s(t-1)| \leq 10$.
- **range_check** : controlla che il byte contenuto su \$s2 appartenga al range, in ASCII, del dizionario di interesse { 0,1,..,9}, in caso contrario stampa messaggio di carattere sconosciuto.
- **sec_num** : controlla il secondo numero intero dell'input, se il byte contenuto su \$s2 è uno spazio, allora l'input è formato solo da un intero e il sensore funziona, altrimenti controllo il terzo numero.
- **third_num**: carica il terzo elemento dell'input, se non è uno spazio allora si tratta di un input formato da almeno 3 interi, per cui il sensore dalle specifiche risulta non funzionante.
Se invece è uno spazio ripristina l'indirizzo del buffer di lettura \$s1 e il registro \$t3, numero di input residui da leggere.

- **str_to_int**: procedura che converte da stringa ASCII a intero decimale l'input. Contiene un controllo per la prima esecuzione in $t=0$, per cui viene trascurato il vincolo $|s(t) - s(t-1)| \leq 10$ in quanto $s(t=-1)$ non esiste.
- **prev_check**: procedura che realizza il vincolo $|s(t) - s(t-1)| \leq 10$, calcolando la differenza $s(t) - s(t-1)$ e passando il controllo alla procedura di normalizzazione se il valore è negativo o al controllo del vincolo se il valore è positivo.
- **neg_val**: procedura che calcola il valore assoluto del numero convertito effettuando il complementare del numero e sommandovi 1.
- **pos_val**: controllo $|s(t) - s(t-1)| \leq 10$.
- **steer_NW**: carica su registro $\$t4$ il valore booleano 0 codificato in ASCII e chiama write_steer.
- **steer_W**: carica su registro $\$t4$ il valore booleano 1 codificato in ASCII e chiama write_steer.
- **write_steer**: memorizza il byte contenuto in $\$t4$ all'indirizzo di memoria puntato da $\$t1$ (indirizzo correctness_array) e all'indirizzo di memoria puntato da $\$s7$ (indirizzo del buffer di uscita per la scrittura su file). Chiama procedura di scrittura sensore e spazio, incrementando opportunamente indirizzo del buffer di uscita. Salva su $\$s5$, registro deputato alla memorizzazione del valore letto all'istante $s(t-1)$, il valore attuale $s(t)$, che verrà poi salvato su SF.
- **to_nxt**: procedura che scorre l'input finché non trova un byte diverso dallo spazio, una volta trovato salva su SF i registri utilizzati, carica indirizzo di ritorno del chiamante, dealloca lo stack e ritorna al chiamante.

- **dist_S** : alloca stack di 84 byte (24 utilizzati per memorizzare i registri utilizzati da slope_S, 28 per memorizzare i registri utilizzati da steer_S, i restanti per il sensore in analisi).
Esegue le procedure di apertura del file di input e di creazione del file di output, al termine delle quali passa al controllo del primo dato in input.
Inizializza i registri \$s3, utilizzato per il calcolo della somma parziale legata alla conversione da esadecimale a decimale, \$s4, utilizzato per memorizzare le occorrenze ancora lecite per ostacoli mobili a stessa distanza, \$s5 per conservare il valore $d2(t-1)d3(t-1)$.
- **loop_D** : Alloca stack di 84 byte per puntare all'area di memoria di competenza, salva indirizzo di ritorno e ripristina da SF i registri precedentemente utilizzati.
Effettua controllo per verificare eventuale terminazione del file di input (\$t3, numero di elementi residui da leggere pari a zero) e carica sul registro \$s2 il primo byte da leggere.
Inizializza il registro \$s3 a zero per la somma parziale della conversione.
- **lett_check** : controlla che il byte contenuto su \$s2 appartenga al range, in ASCII, del dizionario di interesse { A,B }, in caso contrario stampa messaggio di carattere sconosciuto.
- **char_check** : controlla il secondo numero intero dell'input, se il byte contenuto su \$s2 è uno spazio o è il carattere di fine input passa il controllo alla procedura conversion_end, altrimenti controlla se il secondo elemento di input è una lettera o un numero : nel primo caso passa il controllo alla procedura di normalizzazione, nel secondo converte da ASCII a intero decimale effettuando sottrazione opportuna (-48).
- **is_letter**: effettua controllo che il secondo numero intero dell'input, lettera, appartenga al dizionario delle lettere usate per la codifica esadecimale { A,B , C, .. , F}, in caso contrario stampa messaggio di carattere sconosciuto.
Effettua allora sottrazione opportuna (-55) per ottenere il corrispondente numero intero decimale.
- **normalize_in**: procedura che effettua la conversione dell'elemento in analisi in codifica esadecimale : memorizza il secondo elemento di input (convertito in intero decimale) nel registro \$s3, nel momento in cui analizza il terzo elemento di input (sempre convertito in intero decimale) effettua moltiplicazione per 16 del secondo elemento e vi somma il terzo in modo da ottenere la codifica di $d2(t)d3(t)$ in codifica esadecimale.

- **conversion_end:** procedura che effettua controllo per realizzare i vincoli $d2(t)d3(t) \leq 50$ e $d2(t)d3(t) \neq 0$.
Controlla il primo elemento di input (ovvero, in caso di input regolare, la lettera che identifica ostacolo fisso o mobile), nel caso di lettera B passa il controllo alla procedura `save_mov_object`, nel caso di lettera A invece ripristina il numero di occorrenze di ostacoli mobili consentite e salta a `dist_W` in quanto il sensore rispetta le specifiche e risulta correttamente funzionante.
- **save_mov_object:** procedura che controlla che il valore $d2(t)d3(t)$ sia uguale alla lettura di tale valore all'istante $(t-1)$: nel caso di esito positivo passa al controllo `dist_check`, altrimenti aggiorna su SF il valore $d2(t)d3(t)$ (che diventerà $d2(t-1)d3(t-1)$ alla prossima iterazione) e resetta il numero di occorrenze di ostacoli mobili.
- **dist_check:** effettua sottrazione di un elemento dal registro `$s4`.
Se il contenuto di tale registro è pari a 0, allora non sono consentiti ulteriori ostacoli mobili, pertanto il sensore non funziona e passa il controllo a `dist_error`. In caso contrario aggiorna il valore $d2(t)d3(t)$ su SF e salta a `dist_W`
- **dist_error:** aggiorna il valore $d2(t)d3(t)$ su SF, il sensore non funziona correttamente in base alle specifiche fornite.
- **dist_NW :** carica su registro `$t4` il valore booleano 0 codificato in ASCII e chiama `write_dist`.
- **dist_W :** carica su registro `$t4` il valore booleano 1 codificato in ASCII e chiama `write_dist`.
- **write_dist :** memorizza il byte contenuto in `$t4` all'indirizzo di memoria puntato da `$t1` (indirizzo `correctness_array`) e all'indirizzo di memoria puntato da `$s7` (indirizzo del buffer di uscita per la scrittura su file).
Chiama procedura di scrittura sensore e spazio, incrementando opportunamente indirizzo del buffer di uscita.
Controlla il numero di input residui da leggere ed eventualmente chiude le operazioni di lettura e scrittura sul sensore.
- **nxt_val :** procedura che scorre l'input finché non trova un byte diverso dallo spazio o dall'elemento di fine lettura, nel primo caso salva su SF i registri utilizzati , carica indirizzo di ritorno del chiamante, dealloca lo stack e ritorna al chiamante, nel secondo caso passa il controllo alla procedura di chiusura sensore.

- **P1** : Alloca 96 byte su SF (84 per i sensori di lettura, 12 usati da questa procedura), memorizza indirizzo di ritorno su SF, carica su \$s3 contenuto ricevuto come parametro attraverso \$a1, ovvero indirizzo dell'array di correttezza.
Carica sui registri \$t1, \$t2,\$t3 i byte memorizzati rispettivamente in prima, seconda e terza posizione dell'array di correttezza.
Chiama la procedura per la creazione del file di output per P1.
- **loop_P1** : Alloca 96 byte su SF (84 per i sensori di lettura, 12 usati da questa procedura), ripristina da SF i registri precedentemente utilizzati per la scrittura su file. Carica sui registri \$t1, \$t2,\$t3 i byte memorizzati rispettivamente in prima, seconda e terza posizione dell'array di correttezza.
- **P1_ex** : procedura che realizza la prima politica di aggregazione effettuando un AND bitwise tra tutti e tre i registri.
Memorizza il byte contenuto in \$t4 a partire dall'indirizzo del buffer di uscita \$s1 e passa il controllo alla stampa su file, incrementando opportunamente l'indirizzo del buffer
Effettua controllo su \$a3, che è stato passato come parametro.
Qualora il contenuto di esso sia 0, significa che è stato scritto il valore di correttezza dell'intero sistema relativo all'ultimo input, per tutti i sensori, pertanto è possibile terminare le operazioni su questa policy.
In caso contrario stampa anche spazio, salva su SF i registri utilizzati e dealloca lo stack, ritornando il controllo al chiamante.
- **P2** : Alloca 108 byte su SF (84 per i sensori di lettura, 12 per P1 e 12 da questa procedura), memorizza indirizzo di ritorno su SF, carica su \$s3 contenuto ricevuto come parametro attraverso \$a1, ovvero indirizzo dell'array di correttezza.
Carica sui registri \$t1, \$t2,\$t3 i byte memorizzati rispettivamente in prima, seconda e terza posizione dell'array di correttezza.
Chiama la procedura per la creazione del file di output per P1.
- **loop_P2**: Alloca 108 byte su SF (84 per i sensori di lettura, 12 per P1 e 12 da questa procedura), ripristina da SF i registri precedentemente utilizzati per la scrittura su file. Carica sui registri \$t1, \$t2,\$t3 i byte memorizzati rispettivamente in prima, seconda e terza posizione dell'array di correttezza.
- **P2_ex** : procedura che realizza la seconda politica di aggregazione per cui il sistema risulta funzionante in caso di corretto funzionamento di almeno due sensori su 3, attraverso la seguente espressione : $ab + c * (ab' + a'b)$, dove a, b e c sono i valori di verità dei tre sensori.
Memorizza il byte contenuto in \$t5 a partire dall'indirizzo del buffer di uscita

\$s1 e passa il controllo alla stampa su file, incrementando opportunamente l'indirizzo del buffer

Effettua controllo su \$a3, che è stato passato come parametro.

Qualora il contenuto di esso sia 0, significa che è stato scritto il valore di correttezza dell'intero sistema relativo all'ultimo input, per tutti i sensori, pertanto è possibile terminare le operazioni su questa policy.

In caso contrario stampa anche spazio, salva su SF i registri utilizzati e dealloca lo stack, ritornando il controllo al chiamante.

- **P3:** Alloca 120 byte su SF (84 per i sensori di lettura, 12 per P1, 12 per P2 e 12 da questa procedura), memorizza indirizzo di ritorno su SF, carica su \$s3 contenuto ricevuto come parametro attraverso \$a1, ovvero indirizzo dell'array di correttezza.
Carica sui registri \$t1, \$t2, \$t3 i byte memorizzati rispettivamente in prima, seconda e terza posizione dell'array di correttezza.
Chiama la procedura per la creazione del file di output per P1.
- **loop_P3:** Alloca 108 byte su SF (84 per i sensori di lettura, 12 per P1, 12 per P2 e 12 da questa procedura), ripristina da SF i registri precedentemente utilizzati per la scrittura su file. Carica sui registri \$t1, \$t2, \$t3 i byte memorizzati rispettivamente in prima, seconda e terza posizione dell'array di correttezza.
- **P3_ex :** procedura che realizza la seconda politica di aggregazione per cui il sistema risulta funzionante in caso di corretto funzionamento di almeno un sensore su 3, attraverso l'operazione di OR bitwise dei primi due elementi, il cui risultato viene combinato con l' OR bitwise del terzo elemento.
Memorizza il byte contenuto in \$t5 a partire dall'indirizzo del buffer di uscita \$s1 e passa il controllo alla stampa su file, incrementando opportunamente l'indirizzo del buffer
Effettua controllo su \$a3, che è stato passato come parametro.
Qualora il contenuto di esso sia 0, significa che è stato scritto il valore di correttezza dell'intero sistema relativo all'ultimo input, per tutti i sensori, pertanto è possibile terminare le operazioni su questa policy.
In caso contrario stampa anche spazio, salva su SF i registri utilizzati e dealloca lo stack, ritornando il controllo al chiamante.

Descrizione dei registri utilizzati

- **Slope**

- **\$t3** numero di valori residui da leggere in input
- **\$t2** file_OUT_Descriptor
- **\$s0** file_IN_Descriptor
- **\$s1** buffer_IN_address
- **\$s7** buffer_OUT_address
- **\$t1** correctness_array address
- **\$s2** byte contenuto su buffer_IN_address
- **\$t4** valore booleano (0,1) codificato in ASCII
- **\$t5** valore spazio codificato in ASCII

- **Steer**

- **\$t3** numero di valori residui da leggere in input
- **\$t2** file_OUT_Descriptor
- **\$s0** file_IN_Descriptor
- **\$s1** buffer_IN_address
- **\$s5** valore lettura istante (t-1)
- **\$s7** buffer_OUT_address
- **\$t1** correctness_array address
- **\$s2** byte contenuto su buffer_IN_address
- **\$t6** flag di prima esecuzione, inizializzato a 1 e poi resettato a 0.
- **\$t4** usato per operazioni di verifica correttezza input tramite operazioni slti.
- **\$t5** inizialmente usato per conservare somma parziale della conversione, poi inizializzato a valore spazio codificato in ASCII.
- **\$a3** valore booleano (0,1) codificato in ASCII
- **\$t7** usato per operazioni di verifica che la differenza $s(t)-s(t-1)$ fosse positiva o negativa (tramite slt).

- **Dist**

- **\$t3** numero di valori residui da leggere in input
- **\$t2** file_OUT_Descriptor
- **\$s0** file_IN_Descriptor
- **\$s1** buffer_IN_address
- **\$s7** buffer_OUT_address
- **\$t1** correctness_array address
- **\$s2** byte contenuto su buffer_IN_address
- **\$s3** registro per somma parziale conversione
- **\$s4** contatore occorrenze residue ostacoli mobili a stessa distanza

- **\$s5** registro che conserva $d(t-1)$ per ostacoli mobili
 - **\$t4** inizialmente usato per operazioni di verifica correttezza input tramite operazioni slti, poi valore booleano (0,1) codificato in ASCII
 - **\$t5** valore spazio codificato in ASCII
 - **\$t6** usato per indirizzare la lettera iniziale di ogni input ($\$s1 - 2$)
 - **\$t7** usato per operazioni di verifica che la differenza $s(t)-s(t-1)$ fosse positiva o negativa (tramite slt).
- **P1**
 - **\$t1** elemento in prima posizione di correctness_array
 - **\$t2** elemento in seconda posizione di correctness_array
 - **\$t3** elemento in terza posizione di correctness_array
 - **\$t4** registro ausiliario per AND bitwise
 - **\$t5** valore spazio codificato in ASCII
 - **\$s0** file_OUT_Descriptor
 - **\$s1** buffer_OUT_address
 - **\$s3** indirizzo di correctness array passato come argomento
 - **\$a3** flag di fine lettura
- **P2**
 - **\$t1** elemento in prima posizione di correctness_array
 - **\$t2** elemento in seconda posizione di correctness_array
 - **\$t3** elemento in terza posizione di correctness_array
 - **\$t5** usato per calcolo ' $P1 \text{ AND } bP2$ ', poi per ' $(P1 \text{ AND } P2) \text{ OR } (P3 \text{ AND } (P1 \text{ XOR } P2))$ ', infine per valore spazio codificato in ASCII
 - **\$t6** usato per calcolo ' $P1 \text{ XOR } P2$ '
 - **\$t7** usato per calcolo ' $P3 \text{ AND } (P1 \text{ XOR } P2)$ '
 - **\$s0** file_OUT_Descriptor
 - **\$s1** buffer_OUT_address
 - **\$s3** indirizzo di correctness array passato come argomento
 - **\$a3** flag di fine lettura
- **P3**
 - **\$t1** elemento in prima posizione di correctness_array
 - **\$t2** elemento in seconda posizione di correctness_array
 - **\$t3** elemento in terza posizione di correctness_array
 - **\$t4** registro ausiliario per ' $P1 \text{ OR } P2$ ' bitwise
 - **\$t5** per ' $P1 \text{ OR } P2$ ' OR P3 bitwise, poi valore spazio codificato in ASCII
 - **\$s0** file_OUT_Descriptor
 - **\$s1** buffer_OUT_address
 - **\$s3** indirizzo di correctness array passato come argomento
 - **\$a3** flag di fine lettura

Test di corretto funzionamento

Input Sensore di pendenza

-20 -12 -10 50 90 110 25 -13 65 21

Correttezza pendenza

1 **1** **1** 1 0 0 1 1 0 1

Input Sensore di sterzo

45 50 60 80 14 58 11 85 91 17

Correttezza sterzo

1 **1** **1** 0 0 0 0 0 1 0

Input Sensore di distanza

B1A B1A B1A B1A A22 A1D A00 A1A B00 B2A

Correttezza distanza

1 **1** **0** 0 1 1 0 1 0 1

Policy 1

1 **1** **0** 0 0 0 0 0 0 0

Policy 2

1 **1** **1** 0 0 0 0 1 0 1

Policy 3

1 **1** **1** 1 1 1 1 1 1 1

Considerazioni sui requisiti di memoria

Il codice prevede il seguente numero di istruzioni:

- | | |
|------------------|-----------------|
| - 42 addi | - 23 j |
| - 1 addu | - 17 jr |
| - 1 add | - 11 beq |
| - 24 sub | - 6 bne |
| - 2 mul | - 3 blez |
| - 43 la | - 4 bgt |
| - 36 lb | - 2 blt |
| - 15 sb | - 1 bgtz |
| - 54 lw | - 4 slti |
| - 41 sw | - 1 ble |
| - 43 move | - 3 or |
| - 39 li | - 1 xor |
| - 51 jal | - 1 nor |
| - 15 beqz | - 4 and |

Sono state utilizzate pseudo-istruzioni quali **li**, **la**, **move**, **mul**, **beqz**, **bne**, **blez**, **bgt**, **bgtz**, **ble**.

In questo caso il numero totale di istruzioni è di 604, per uno spazio totale occupato di 2.4 KB, considerando che per le operazioni di trasferimento dati (la, li) , confronto (beqz, bne,..) e elaborazione aritmetica (mul) vengono richieste due istruzioni per ciascuna pseudo-istruzione considerata.

Per quanto riguarda l'occupazione del segmento dati, contenente i dati di elaborazione, visto che non è stata utilizzata allocazione dinamica della memoria, contribuiscono solamente i dati statici, ovvero i dati già presenti in memoria al momento della compilazione, e che ci rimangono per tutto il tempo di esecuzione del programma.

In questo caso l'occupazione massima del segmento dati sarà di 2300 Byte, cioè 2.3 KB, dato dalla somma dei byte riservati per i buffer di lettura e scrittura su file.

Non è stata considerata ai fini dello studio la memoria occupata dalle stringhe di stampa dei messaggi su console e delle stringhe dei percorsi dei file di input in quanto variabili a seconda della locazione stessa del file.

Lo spazio massimo allocato sullo stack è pari a 120 Byte.

Il tempo complessivo del programma è stato calcolato rimuovendo dal numero di istruzioni sopra riportate le istruzioni per la lettura/scrittura. Non sono state prese in considerazione le operazioni del tipo **la** e **jal** per la stampa di messaggi su console. Sono state considerate anche le pseudo-istruzioni.

Il numero di istruzioni complessivo sarà:

- | | |
|------------------|-----------------|
| - 42 addi | - 23 j |
| - 1 addu | - 17 jr |
| - 1 add | - 11 beq |
| - 24 sub | - 6 bne |
| - 2 mul | - 3 blez |
| - 31 la | - 4 bgt |
| - 36 lb | - 2 blt |
| - 15 sb | - 1 bgtz |
| - 54 lw | - 4 slti |
| - 41 sw | - 1 ble |
| - 43 move | - 3 or |
| - 39 li | - 1 xor |
| - 28 jal | - 1 nor |
| - 15 beqz | - 4 and |

Istruzioni di caricamento **da memoria**: $90 * 800 = 72.000$ ps

Istruzioni di memorizzazione **in memoria**: $56 * 700 = 39.200$ ps

Altre istruzioni : $423 * 500 = 211.500$ ps

Tempo di esecuzione totale : $(72.000 + 39.200 + 211.500)$ ps = $0,3227$ μ s

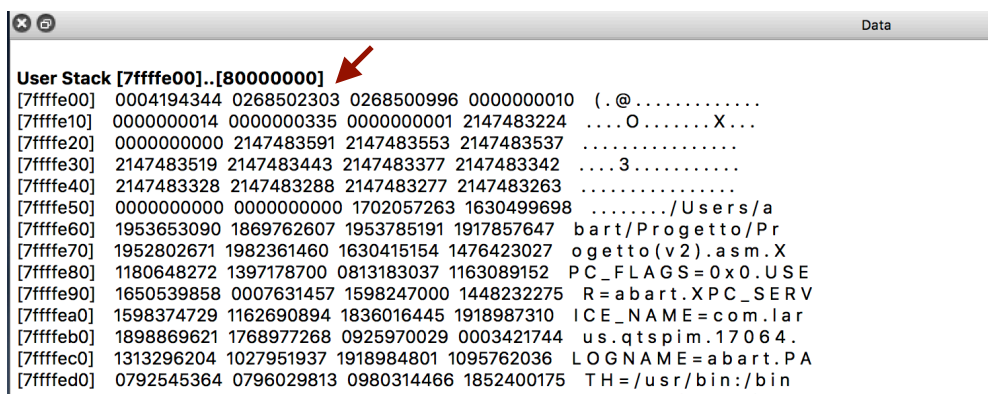
Rispetto all'implementazione attuale è possibile diminuire i requisiti di memoria utilizzando un buffer di lettura di dimensione arbitraria, leggendo il file un numero arbitrario di byte alla volta, con lo svantaggio tuttavia di dover chiamare ogni volta la procedura di apertura del file da leggere, incrementando dunque il tempo di esecuzione del programma.

Potrebbe essere migliorata inoltre l'allocazione di memoria in maniera da statica a dinamica sullo heap attraverso la chiamata di sistema `sbrk()`.

In base al design di progettazione la soluzione adottata risulta poco ottimizzabile dal punto di vista del tempo di esecuzione, se non attraverso procedura di riduzione del numero di registri utilizzati, e, di conseguenza, del numero di istruzioni.

Screenshot

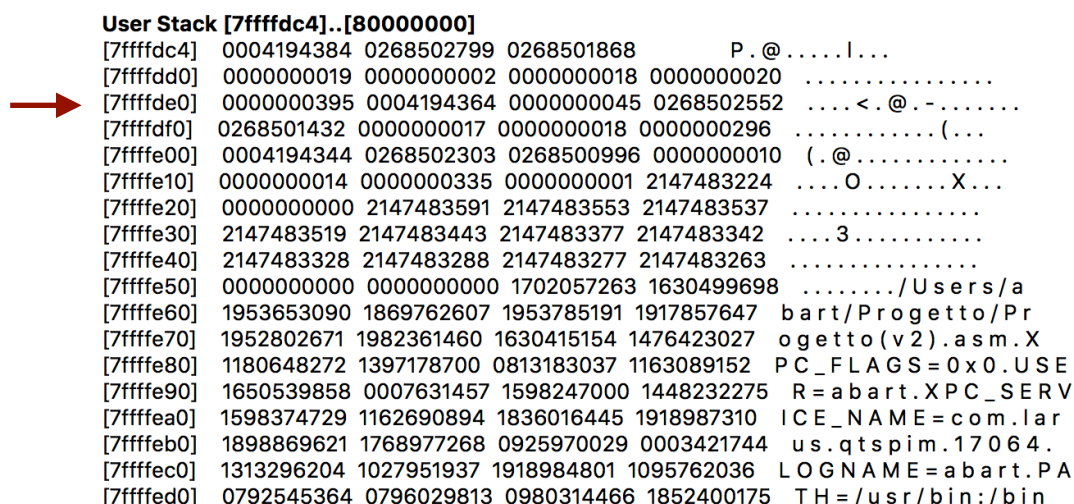
Screenshot dello stack al momento del salvataggio del registro \$s7 contenente buffer_OUT_index nella posizione 4 partire da \$sp, all'interno della procedura to_next del sensore di pendenza.



```

User Stack [7ffffe00]..[80000000]
[7ffffe00] 0004194344 0268502303 0268500996 0000000010 (.@.....
[7ffffe10] 0000000014 0000000335 0000000001 2147483224 ....O.....X...
[7ffffe20] 0000000000 2147483591 2147483553 2147483537 .....
[7ffffe30] 2147483519 2147483443 2147483377 2147483342 ....3.....
[7ffffe40] 2147483328 2147483288 2147483277 2147483263 .....
[7ffffe50] 0000000000 0000000000 1702057263 1630499698 ...../Users/a
[7ffffe60] 1953653090 1869762607 1953785191 1917857647 bart/Progetto/Pr
[7ffffe70] 1952802671 1982361460 1630415154 1476423027 ogetto(v2).asm.X
[7ffffe80] 1180648272 1397178700 0813183037 1163089152 PC_FLAGS=0x0.USE
[7ffffe90] 1650539858 0007631457 1598247000 1448232275 R=abart.XPC_SERV
[7ffffea0] 1598374729 1162690894 1836016445 1918987310 ICE_NAME=com.lar
[7ffffeb0] 1898869621 1768977268 0925970029 0003421744 us.qts pim.17064.
[7ffffec0] 1313296204 1027951937 1918984801 1095762036 LOGNAME=abart.PA
[7ffffed0] 0792545364 0796029813 0980314466 1852400175 TH=/usr/bin:/bin
  
```

Screenshot dello stack al momento del salvataggio del registro \$t3 contenente numero di valori residui da leggere in input per il sensore di distanza nella posizione 28 a partire da \$sp, all'interno della procedura nxt_val del sensore di distanza.



```

User Stack [7ffffdc4]..[80000000]
[7ffffdc4] 0004194384 0268502799 0268501868 P.@....l...
[7ffffdd0] 0000000019 0000000002 0000000018 0000000020 .....
[7ffffde0] 0000000395 0004194364 0000000045 0268502552 ....<.@.-.....
[7ffffdf0] 0268501432 0000000017 0000000018 0000000296 .....(....
[7ffffe00] 0004194344 0268502303 0268500996 0000000010 (.@.....
[7ffffe10] 0000000014 0000000335 0000000001 2147483224 ....O.....X...
[7ffffe20] 0000000000 2147483591 2147483553 2147483537 .....
[7ffffe30] 2147483519 2147483443 2147483377 2147483342 ....3.....
[7ffffe40] 2147483328 2147483288 2147483277 2147483263 .....
[7ffffe50] 0000000000 0000000000 1702057263 1630499698 ...../Users/a
[7ffffe60] 1953653090 1869762607 1953785191 1917857647 bart/Progetto/Pr
[7ffffe70] 1952802671 1982361460 1630415154 1476423027 ogetto(v2).asm.X
[7ffffe80] 1180648272 1397178700 0813183037 1163089152 PC_FLAGS=0x0.USE
[7ffffe90] 1650539858 0007631457 1598247000 1448232275 R=abart.XPC_SERV
[7ffffea0] 1598374729 1162690894 1836016445 1918987310 ICE_NAME=com.lar
[7ffffeb0] 1898869621 1768977268 0925970029 0003421744 us.qts pim.17064.
[7ffffec0] 1313296204 1027951937 1918984801 1095762036 LOGNAME=abart.PA
[7ffffed0] 0792545364 0796029813 0980314466 1852400175 TH=/usr/bin:/bin
  
```

Screenshot del segmento dati a seguito di esecuzione dell'elaborazione del sensore di pendenza : è possibile notare come siano stati caricati in memoria tutti gli elementi presenti nel file di testo relativo a tale sensore.

```

User data segment [10000000]..[10040000]
[10000000]..[1000ffff] 00000000
[10010000] 0540029485 0540160301 0540029229 0958410805 -20 -12 -10 50 9
[10010010] 0825303088 0892477488 0858860832 0540358176 0 110 25 -13 65
[10010020] 0757084466 0924856373 0859054128 0540356896 21 -50 70 43 15
[10010030] 0908079673 0825040946 0942481456 0876027952 92 62 -10 -80 74
[10010040] 0540226080 0540029485 0540160301 0540029229 23 -20 -12 -10
[10010050] 0958410805 0825303088 0892477488 0858860832 50 90 110 25 -13
[10010060] 0540358176 0757084466 0924856373 0859054128 65 21 -50 70 43
[10010070] 0540356896 0908079673 0825040946 0942481456 15 92 62 -10 -8
[10010080] 0876027952 0540226080 0540029485 0540160301 0 74 23 -20 -12
[10010090] 0540029229 0958410805 0825303088 0892477488 -10 50 90 110 25
[100100a0] 0858860832 0540358176 0757084466 0924856373 -13 65 21 -50 7
[100100b0] 0859054128 0540356896 0908079673 0825040946 0 43 15 92 62 -1
[100100c0] 0942481456 0876027952 0540226080 0540029485 0 -80 74 23 -20
[100100d0] 0540160301 0540029229 0958410805 0825303088 -12 -10 50 90 11
[100100e0] 0892477488 0858860832 0540358176 0757084466 0 25 -13 65 21 -
[100100f0] 0924856373 0859054128 0540356896 0908079673 50 70 43 15 92 6
[10010100] 0825040946 0942481456 0876027952 0540226080 2 -10 -80 74 23
[10010110] 0540029485 0540160301 0540029229 0958410805 -20 -12 -10 50 9
[10010120] 0825303088 0892477488 0858860832 0540358176 0 110 25 -13 65
[10010130] 0757084466 0924856373 0859054128 0540356896 21 -50 70 43 15
[10010140] 0908079673 0825040946 0942481456 0876027952 92 62 -10 -80 74
[10010150] 0003355168 0000000000 0000000000 0000000000 23.....
[10010160]..[1001018f] 00000000
[10010190] 1702057263 1630499698 1953653090 1869762607 /Users/abart/Pro
[100101a0] 1953785191 1701850991 1852138606 1313431930 getto/pendenzaIN
[100101b0] 1954051118 0000000000 0000000000 0000000000 .txt.....
[100101c0]..[10010343] 00000000
[10010344] 1934962432 0796095077 1918984801 ./Users/abar
[10010350] 1917857652 1952802671 1932488564 2054317428 t/Progetto/sterz
[10010360] 0776882543 0007633012 0000000000 0000000000 oIN.txt.....

```

Screenshot dello stack al momento del salvataggio del registro \$s1 contenente file_OUT_Descriptor nella posizione 8 a partire da \$sp, all'interno della procedura P3_ex della Politica 3.

```

User Stack [7ffffda0]..[80000000]
[7ffffda0] 0004194444 0000000023 0268503528 0004194424 ..@.....x.@.
[7ffffdb0] 0000000022 0268503288 0004194404 0000000021 .....d.@.....
[7ffffdc0] 0268503048 0004194384 0268502799 0268501868 ....P.@.....l...
[7ffffdd0] 0000000019 0000000002 0000000018 0000000020 .....
[7ffffde0] 0000000395 0004194364 0000000045 0268502552 ....<.@.-.....
[7ffffdf0] 0268501432 0000000017 0000000018 0000000296 .....(....
[7ffffe00] 0004194344 0268502303 0268500996 0000000010 (.@.....
[7ffffe10] 0000000014 0000000335 0000000001 2147483224 ....O.....X...
[7ffffe20] 0000000000 2147483591 2147483553 2147483537 .....
[7ffffe30] 2147483519 2147483443 2147483377 2147483342 ....3.....
[7ffffe40] 2147483328 2147483288 2147483277 2147483263 .....
[7ffffe50] 0000000000 0000000000 1702057263 1630499698 ...../Users/a
[7ffffe60] 1953653090 1869762607 1953785191 1917857647 bart/Progetto/Pr
[7ffffe70] 1952802671 1982361460 1630415154 1476423027 ogetto(v2).asm.X
[7ffffe80] 1180648272 1397178700 0813183037 1163089152 PC_FLAGS=0x0.USE
[7ffffe90] 1650539858 0007631457 1598247000 1448232275 R=abart.XPC_SERV
[7ffffea0] 1598374729 1162690894 1836016445 1918987310 ICE_NAME=com.lar
[7ffffeb0] 1898869621 1768977268 0925970029 0003421744 us.qtspim.17064.
[7ffffec0] 1313296204 1027951937 1918984801 1095762036 LOGNAME=abart.PA
[7ffffed0] 0792545364 0796029813 0980314466 1852400175 TH=/usr/bin:/bin
[7ffffee0] 1937059642 1651715954 0792358505 1852400243 :/usr/sbin:/sbin

```

Codice

.data

```
slope_IN_buffer : .space 400 #Riservo 4B * 100 (=N rilevamenti + 100 spazi ) per il sensore di
pendenza.
slope_IN_path: .asciiz "/Users/abart/Documents/Universita/AE/Progetto2018/pendenzaIN.txt"
steer_IN_buffer : .space 300 #Riservo 3B * 100 (=N rilevamenti + 100 spazi) per il sensore di
sterzata.
steer_IN_path: .asciiz "/Users/abart/Documents/Universita/AE/Progetto2018/sterzoIN.txt"
dist_IN_buffer : .space 400 #Riservo 4B * 100 (=N rilevamenti + 100 spazi) per il sensore di
distanza.
dist_IN_path: .asciiz "/Users/abart/Documents/Universita/AE/Progetto2018/distanzaIN.txt"

slope_OUT_buffer : .space 200 #Riservo 2B * 100 (= N rilevamenti + 100 spazi) per il sensore di
pendenza.
slope_OUT_path: .asciiz "/Users/abart/Documents/Universita/AE/Progetto2018/
correttezzaPendenzaOUT.txt"
steer_OUT_buffer : .space 200 #Riservo 2B * 100 (= N rilevamenti + 100 spazi) per il sensore di
sterzata.
steer_OUT_path: .asciiz "/Users/abart/Documents/Universita/AE/Progetto2018/
correttezzaSterzoOUT.txt"
dist_OUT_buffer : .space 200 #Riservo 2B * 100 (= N rilevamenti + 100 spazi) per il sensore di
distanza.
dist_OUT_path: .asciiz "/Users/abart/Documents/Universita/AE/Progetto2018/
correttezzaDistanzaOUT.txt"

pol_1_buffer : .space 200 #Riservo 2B * 100 (= N rilevamenti + 100 spazi) per la politica 1.
pol_1_path: "/Users/abart/Documents/Universita/AE/Progetto2018/correttezzaP1.txt"
pol_2_buffer : .space 200 #Riservo 2B * 100 (= N rilevamenti + 100 spazi) per la politica 2.
pol_2_path: "/Users/abart/Documents/Universita/AE/Progetto2018/correttezzaP2.txt"
pol_3_buffer : .space 200 #Riservo 2B * 100 (= N rilevamenti + spazi ) per la politica 3.
pol_3_path: "/Users/abart/Documents/Universita/AE/Progetto2018/correttezzaP3.txt"

welcome: .asciiz "Welcome to the control unit of the car. \n"
unknown_char : .asciiz "ALERT : Unknown value read from the sensor. \n"
start_S: .asciiz "Starting analyzis of Slope sensor... \n"
start_St: .asciiz "Starting analyzis of Steer sensor... \n"
start_D: .asciiz "Starting analyzis of Distance sensor... \n"
start_P1: .asciiz "Started computation of Policy 1. \n"
start_P2: .asciiz "Started computation of Policy 2. \n"
start_P3: .asciiz "Started analysis of of Policy 3. \n"
completed_S: .asciiz "Completed Slope sensor computation test. \n"
completed_St: .asciiz "Completed Steer sensor computation test. \n"
completed_D: .asciiz "Completed Distance sensor computation test. \n"
complete: .asciiz "Program computation completed. Terminated execution."
correctness_array: .space 12 # riservo 3 word per i valori di correttezza
```

.text
.globl main

main:	la \$a0, welcome jal printf	# Stampa messaggio di benvenuto
	la \$t9, correctness_array	# \$t9 contiene l'indirizzo di partenza dell'array da # passare ai sensori
turnOn:	la \$a0, start_S jal printf	# Stampa messaggio di inizio elaborazione sensore di pendenza
	la \$a1, (\$t9) jal slope_S	# assegna a \$a1 indirizzo di correctness_array da # passare come argomento # salta a istruzione di avvio sensore pendenza
	la \$a0, start_St jal printf	# Stampa messaggio inizio elaborazione sterzo
	la \$a1, 4(\$t9)	# \$a1 contiene l'indirizzo di partenza dell'array da # passare ai sensori incrementato di 4
	jal steer_S	# salta a istruzione di avvio sensore di sterzo
	la \$a0, start_D jal printf	# Stampa messaggio di inizio elaborazione distanza
	la \$a1, 8(\$t9)	# \$a1 contiene l'indirizzo di partenza dell'array da # passare ai sensori incrementato di 8
	jal dist_S	# salta a istruzione di avvio sensore di distanza
	la \$a0, start_P1 jal printf	# Stampa messaggio di inizio elaborazione Policy 1
	la \$a1, (\$t9)	# \$a1 contiene l'indirizzo di partenza dell'array da # passare alla Policy 1
	jal P1	# salta a istruzione di avvio Policy 1
	la \$a0, start_P2 jal printf	# Stampa messaggio di inizio elaborazione Policy 2
	la \$a1, (\$t9)	# \$a1 contiene l'indirizzo di partenza dell'array da # passare alla Policy 2
	jal P2	# salta a istruzione di avvio Policy 2

	la \$a0, start_P3 jal printf	# Stampa messaggio di inizio elaborazione Policy 3
	la \$a1, (\$t9) jal P3	# \$a1 contiene l'indirizzo di partenza dell'array da # passare alla Policy 3 # salta a istruzione di avvio Policy 3
engine_ON:	la \$a1, (\$t9) jal loop_SI	# salta a istruzione di recupero sensore di pendenza
	la \$a1, 4(\$t9) jal loop_St	# salta a istruzione di recupero sensore di sterzo
	la \$a1, 8(\$t9) jal loop_D	# salta a istruzione di avvio sensore di distanza
	move \$a3, \$v0	# salva su \$a3 0-flag di terminazione della lettura # restituito come risultato dal sensore di distanza
	la \$a1, (\$t9) jal loop_P1 move \$a3, \$v0	# salta a istruzione di avvio Policy 1 # salva su \$a3 0-flag di uscita da Policy 1 restituito # come risultato
	la \$a1, (\$t9) jal loop_P2 move \$a3, \$v0	# salta a istruzione di avvio Policy 2 # salva su \$a3 0-flag di uscita da Policy 2 restituito # come risultato
	la \$a1, (\$t9) jal loop_P3 beqz \$v0, exit	# salta a istruzione di avvio Policy 3 # se il risultato restituito attraverso \$v0 e' 0, tutte le # policy sono chiuse e chiudo il programma
	j engine_ON	# loop
slope_S:	addi \$sp, \$sp, -24 sw \$ra, 0(\$sp)	# Alloco 24 byte su Stack # salvo su SF indirizzo di ritorno
	move \$t1, \$a1	# salvo su \$t1 indirizzo di partenza array di # correttezza passato come argomento
	la \$a0, slope_IN_path jal open_file_r	# filepath_IN per lettura da file # salta a procedura di apertura di file da leggere
	move \$s0, \$v0	# \$s0 contiene file_IN_Descriptor
	la \$a1, slope_IN_buffer jal read_from_file	# indirizzo di partenza del buffer di lettura # salta a procedura di lettura da file

	move \$t3, \$v0	\$t3 contiene numero di valori da leggere in input
	move \$s1, \$a1	# carico su \$s1 l'indirizzo di partenza del buffer da leggere
	lb \$s2, (\$s1)	# carico su \$s2 il primo byte indirizzato dal buffer di lettura
	la \$a0, slope_OUT_path	# filepath_OUT per scrittura su file
	jal open_file_w	# salta a procedura di scrittura su file
	move \$t2, \$v0	# \$t2 contiene file_OUT_Descriptor
	la \$s7, slope_OUT_buffer	# carico su \$s7 indirizzo di partenza del buffer_OUT
	j sign_check	
loop_Sl:	addi \$sp, \$sp, -24	# Alloco 24 byte su stack per ripristinare dati locali # precedentemente memorizzati
	sw \$ra, 0(\$sp)	# salvo su SF indirizzo di ritorno
	lw \$t3, 20(\$sp)	# ripristino da SF numero di valori residui da leggere
	lw \$t2, 16(\$sp)	# ripristino da SF file_OUT_Descriptor
	lw \$s0, 12(\$sp)	# ripristino da SF file_IN_Descriptor
	lw \$s1, 8(\$sp)	# ripristino da SF buffer_IN_index
	lw \$s7, 4(\$sp)	# ripristino da SF buffer_OUT_index
	move \$t1, \$a1	# salvo su \$t1 indirizzo di partenza array di # correttezza passato come argomento
	blez \$t3, close_slope	# se il numero di valori residui da leggere e' minore o pari a zero termino l'esecuzione sul sensore
	lb \$s2, (\$s1)	# carico su \$s2 il primo byte indirizzato dal buffer da leggere
sign_check:	bne \$s2, 45, num_check	# se avanza, allora è il simbolo ' - ' (= 45 in ASCII)
	addi \$s1, \$s1, 1	# incremento l'indirizzo del buffer di lettura di una # posizione
	sub \$t3, \$t3, 1	# decremento numero di valori da leggere
	lb \$s2, (\$s1)	# carico su \$s2 il byte successivo
	beq \$s2, 45, unknown_char	# se avanza OK, altrimenti e' nuovamente il simbolo ' - ' ERRORE
num_check:	blt \$s2, 48, unknown_char	# se il contenuto di \$s2 < 0 (= 48 in ASCII) # non appartiene al dizionario di interesse
	bgt \$s2, 57, unknown_char	# se il contenuto si \$s2 > 9 (= 57 in ASCII) # non appartiene al dizionario di interesse
first_val:	bgt \$s2, 53, sec_val	# se il contenuto di \$s2 < 5 (= 53 in ASCII) # passo al controllo dell'elemento successivo
	addi \$s1, \$s1, 2	# incremento di due posizioni l'indirizzo del buffer di lettura

```

sub $t3, $t3, 2      # decremento numero di valori da leggere

lb $s2, ($s1)        # carico su $s2 2 byte indirizzato da buffer incrementato di
                    # due posizioni (3 cifra)

bne $s2, 32, slope_NW  # se $s2 non contiene uno spazio ( = 32 in ASCII ) ho
                    # una lettura di un dato espresso (almeno) in
                    # centinaia =ERRORE
j slope_W

sec_val:      addi $s1, $s1, 1      # incremento buffer_IN_index di una posizione

sub $t3, $t3, 1      # decremento numero di valori da leggere
lb $s2, ($s1)        # carico su $s2 il byte indirizzato da buffer incrementato di
                    # una posizione

beqz $s2, slope_W    # se $s2 contiene zero il file e' terminato, il sensore funziona

bne $s2, 32, slope_NW  # se $s2 non contiene uno spazio ho una lettura di un
                    # input > 60 = ERRORE
j slope_W

slope_NW:      li $t4, 48          # $t4 contiene il valore 0 in ASCII
                    j write_slope

slope_W:      li $t4, 49          # $t4 contiene il valore 1 in ASCII

write_slope:  sb $t4, ($t1)        # memorizzo il byte contenuto su $t4 nella prima locazione
                    # di memoria dell'array
sb $t4, ($s7)        # memorizzo il byte contenuto su $t4 nella locazione di
                    # memoria puntata dal buffer di output

jal write_sensor     # salta a procedura di stampa del valore booleano
addi $s7, $s7, 1     # incremento buffer di uscita di una posizione

beqz $t3, close_slope  # se il numero di valori da leggere e' zero, chiude
                    # l'operazione sul sensore

li $t5, 32           # $t4 contiene lo spazio in codifica ASCII
sb $t5, ($s7)        # memorizzo il byte contenuto su $t4 nella locazione di
                    # memoria puntata dal buffer di# output

jal write_sensor     # salta a procedura di stampa del carattere spazio

addi $s7, $s7, 1     # incremento di una posizione il buffer di uscita

```

to_next:	addi \$s1, \$s1, 1 sub \$t3, \$t3, 1 lb \$s2, (\$s1) beq \$s2, 32, to_next sw \$t3, 20(\$sp) sw \$t2, 16(\$sp) sw \$s0, 12(\$sp) sw \$s1, 8(\$sp) sw \$s7, 4(\$sp) lw \$ra, 0(\$sp) addi \$sp, \$sp, 24 jr \$ra	# incremento di una posizione buffer_IN_index # decremento numero di valori da leggere # carico su \$s2 il byte indirizzato da buffer incrementato di # una posizione # se \$s2 contiene uno spazio, scorro ancora, # altrimenti salvo i registri e torno a main # salvo su SF numero di valori residui da leggere in input # salvo su SF file_OUT_Descriptor # salvo su SF file_IN_Descriptor # salvo su SF buffer_IN_index # salvo su SF buffer_OUT_index # ripristino da SF indirizzo di ritorno del chiamante # dealloco 24 byte da Stack # salto a istruzione contenuta in \$ra
steer_S:	sub \$sp, \$sp, 52 sw \$ra, 0(\$sp) move \$t1, \$a1 la \$a0, steer_IN_path jal open_file_r move \$s0, \$v0 sw \$s0, 16(\$sp) la \$a1, steer_IN_buffer jal read_from_file move \$t3, \$v0 move \$s1, \$a1 lb \$s2, (\$s1) la \$a0, steer_OUT_path jal open_file_w move \$t2, \$v0 sw \$t2, 20(\$sp) la \$s7, steer_OUT_buffer	# Alloco 52 byte su Stack # salvo su SF indirizzo di ritorno # salvo su \$t1 indirizzo dell'array di correttezza passato # come argomento # filepath_IN per lettura da file # salta a procedura di apertura file da leggere # \$s0 contiene file_IN_Descriptor # salvo su SF contenuto di \$s0 nella locazione # di memoria opportunamente assegnata # indirizzo di partenza del buffer di lettura # salta a procedura di lettura da file # memorizzo in \$t3 numero di valori da leggere # carico su \$s1 l'indirizzo di partenza del buffer # carico su \$s2 il primo byte indirizzato dal buffer di lettura # filepath_OUT per scrittura su file # salta a procedura di scrittura su file # \$t2 contiene file_OUT_Descriptor # salvo su SF contenuto di \$t2 nella locazione di # memoria opportunamente assegnata # carico su \$s7 indirizzo di partenza del buffer_OUT
	j range_check	


```

loop_St:    sub $sp,$sp, 52      # Alloco 52 byte su Stack per ripristinare dati locali
                                     # precedentemente memorizzati
            sw $ra, 0($sp)        # salvo su SF indirizzo di ritorno
            lw $t3,24($sp)        # ripristino da SF numero di valori residui da leggere in input
            lw $t2,20($sp)        # ripristino da SF file_OUT_Descriptor
            lw $s0,16($sp)        # ripristino da SF file_IN_Descriptor
            lw $s1,12($sp)        # ripristino da SF buffer_IN_index
            lw $s7,8($sp)         # ripristino da SF buffer_OUT_index
            lw $s5,4($sp)         # ripristino da SF s(t-1)

            move $t1, $a1         # salvo su $t1 indirizzo di partenza array di correttezza
                                     # passato come argomento

            blez $t3, close_steer    # se il numero di valori residui da leggere e' minore o
                                     # pari a zero, termino l'esecuzione sul sensore

            lb $s2, ($s1)          # carico su $s2 il primo byte indirizzato dal buffer di lettura

            li $t6, 1              # $t6 ha il compito di flag per la prima accensione, in cui il
                                     # sensore risulta funzionante se sono verificate tutte le
                                     # specifiche, ad eccezione di quella riguardante la differenza
                                     # con l'istante precedente

range_check: slti $t4, $s2, 48
                bgtz $t4, unknown_char    # se il contenuto di $s2 < 0 ( = 48 in ASCII) allora si
                                     # tratta di un elemento che non appartiene all'insieme
                                     # numerico di interesse

                slti $t4, $s2, 58
                beqz $t4, unknown_char    # se il contenuto di $s2 < 10 ( = 58 in ASCII) allora si
                                     # tratta di un numero del dizionario

sec_num:    addi $s1, $s1, 1        # incremento di una posizione buffer_IN_index
                sub $t3, $t3, 1        # decremento numero di valori da leggere

            lb $s2,($s1)              # carico su $s2 il byte indirizzato dal buffer di lettura
                                     # incrementato di una posizione

            bne $s2, 32, third_num    # se il contenuto di $s2 non e' uno spazio analizzo l'elemento
                                     # successivo, altrimenti il sensore risulta funzionante
            j steer_W

third_num: addi $s1, $s1, 1        # incremento di una posizione buffer_IN_index
                sub $t3, $t3, 1        # decremento numero di valori da leggere

            lb $s2,($s1)              # carico su $s2 il byte indirizzato dal buffer di lettura
                                     # incrementato di una posizione

            bne $s2, 32, steer_NW    # se la terza cifra dell'input non e' uno spazio allora
                                     # l'input e' dell'ordine delle centinaia = ERRORE

```

	sub \$s1, \$s1, 2	# resetto t0 alla prima posizione del numero
	addi \$t3, \$t3, 2	# ripristino numero di valori da leggere
str_to_int:	lb \$s2, (\$s1)	# carico su \$s2 il byte indirizzato dal buffer di lettura alla # posizione precedente ai controlli (primo elemento)
	addi \$s2, \$s2, -48	# converte da ASCII a rappresentazione decimale il primo # elemento dell'input
	mul \$s2, \$s2, 10	# moltiplico il primo elemento per 10 (decine)
	move \$t5, \$s2	# sposto contenuto di \$s2 somma parziale della # conversione, in \$t5
	addi \$s1, \$s1, 1	# incremento di una posizione buffer_IN_index
	sub \$t3, \$t3, 1	# decremento numero di valori da leggere
	lb \$s2, (\$s1)	# carico su \$s2 il byte indirizzato dal buffer di lettura # incrementato di una posizione
	addi \$s2, \$s2, -48	# converte da ASCII a rappresentazione decimale il secondo # elemento dell'input
	addu \$s6, \$t5, \$s2	# somma su \$s6 i due numeri convertiti in rappresentazione # decimale
	beqz \$t6, steer_W	# se e' la prima esecuzione, il contenuto di \$t6 risulta zero e il # sensore funzionante
prev_check:	sub \$t6, \$s6, \$s5	# sottrazione di $s(t) - s(t-1)$ caricata su \$t6
	slt \$t7, \$t6, \$zero	# $t7 = (t6 < 0 ? 1 : 0)$
	bnez \$t7, neg_val	# se \$t7 e' zero si tratta di un numero negativo
	j pos_val	
neg_val:	nor \$t6, \$t6, \$zero	# complemento a uno di \$t6, contenente numero negativo
	addi \$t6, \$t6, 1	# $t6 + 1$ per ottenere valore assoluto della differenza
pos_val:	ble \$t6, 10, steer_W	# se $ s(t) - s(t-1) \leq 10$ il sensore rispetta le # specifiche e funziona
steer_NW:	li \$a3, 48	# \$a3 contiene il valore 0 in ASCII
	j write_steel	
steer_W:	li \$a3, 49	# \$a3 contiene il valore 1 in ASCII
write_steel:	sb \$a3, (\$t1)	# memorizzo il byte contenuto su \$a3 nella seconda # locazione di memoria dell'array

```

sb $a3, ($s7)      # memorizzo il byte contenuto su $a4 all'indirizzo puntato
                   # dal buffer di output

jal write_sensor    # salta a procedura di stampa del valore booleano nel file di

addi $s7, $s7, 1    # incremento di una posizione il buffer di uscita
beqz $t3, close_steer # se il numero di valori da leggere e' zero, chiude
                   # l'operazione sul sensore

li $t5, 32          # $t5 contiene lo spazio in codifica ASCII
sb $t5, ($s7)       # memorizzo il byte contenuto su $t5 nella locazione di
                   # memoria puntata dal buffer di output

jal write_sensor    # procedura per stampa del carattere spazio nel file di output
addi $s7, $s7, 1    # incremento di una posizione il buffer di uscita

move $s5, $s6       # sposto su $s5 contenuto di $s6, contenente s(t) che
                   # diventera' s(t-1) alla prossima lettura

```

```

to_nxt:      addi $s1, $s1, 1      # incremento di una posizione il buffer di uscita
                sub $t3, $t3, 1      # decremento numero di valori da leggere
                lb $s2, ($s1)        # carico su $s2 il byte indirizzato dal buffer di lettura
                beq $s2, 32, to_nxt  # se $s2 contiene uno spazio, scorro ancora,
                                   # altrimenti salvo i registri e torno a main

```

```

sw $s5, 4($sp)     # salvo su SF valore sensore istante precedente
sw $s7, 8($sp)     # salvo su SF buffer_OUT_index
sw $s1, 12($sp)    # salvo su SF buffer_IN_index
sw $t3, 24($sp)    # salvo su SF numero di valori residui da leggere in
                   # input per Steer

```

```

lw $ra, 0($sp)     # ripristino da SF indirizzo di ritorno
addi $sp, $sp, 52  # dealloco 52 elementi da stack

jr $ra             # salto a istruzione contenuta in $ra

```

```

dist_S:      sub $sp, $sp, 84     # alloco stack di 84 byte
                sw $ra, 0($sp)      # salvo su SF indirizzo di ritorno
                move $t1, $a1       # salvo su $t1 indirizzo dell'array di correttezza
                                   # passato come argomento

li $s3, 0          # registro per somma parziale conversione inizializzato a 0

li $s4, 2          # registro contatore del numero massimo di occorrenze di
                   # ostacoli mobili a stessa distanza accettabili

li $s5, 0          # registro che conserva d(t-1) per ostacoli mobili

sw $s5, 20($sp)    # $s5 contiene inizialmente 0, poi valore sensore ostacolo
                   # mobile istante (t-1)

```

```

la $a0, dist_IN_path      # filepath_IN per lettura da file
jal open_file_r           # salta a procedura di apertura di file da leggere
move $s0, $v0             # salvo in s0 il descrittore del file di IN
sw $s0, 12($sp)           # salvo su SF descrittore file IN di Dist

la $a1, dist_IN_buffer    # indirizzo di partenza del buffer di lettura
jal read_from_file        # salta a procedura di lettura da file
move $t3, $v0             # $t3 contiene numero di valori da leggere

move $s1, $a1             # carico su $s1 l'indirizzo di partenza del buffer da leggere
lb $s2, ($s1)             # carico su $s2 il primo byte indirizzato dal buffer di
                          # lettura

la $a0, dist_OUT_path     # filepath_OUT per scrittura su file
jal open_file_w           # salta a procedura di scrittura su file
move $t2, $v0             # $t2 contiene file_OUT_Descriptor
sw $t2, 24($sp)           # salvo su SF file_OUT_Descriptor di Dist

la $s7, dist_OUT_buffer   # assegno a $s7 indirizzo di partenza del buffer da
                          # cui scrivere il file di output

```

j lett_check

```

loop_D:    sub $sp, $sp, 84      # Alloco 84 byte su stack per ripristinare dati locali
                          # precedentemente memorizzati
sw $ra, 0($sp)           # salvo su SF indirizzo di ritorno

lw $t3, 28($sp)          # ripristino da SF numero di valori residui da leggere
                          # in input per sensore attuale
lw $t2, 24($sp)          # ripristino da SF file_OUT_Descriptor
lw $s5, 20($sp)          # ripristino da SF valore s(t-1)
lw $s4, 16($sp)          # ripristino da SF flag ostacoli mobili a stessa
                          # distanza residui
lw $s0, 12($sp)          # ripristino da SF file_OUT_Descriptor
lw $s1, 8($sp)           # ripristino da SF buffer_IN_index
lw $s7, 4($sp)           # ripristino da SF buffer_OUT_index

move $t1, $a1            # salvo su $t1 indirizzo di partenza array di correttezza passato
                          # come argomento

beq $t3, $zero, close_dist # se il numero di valori residui e' pari a zero, termino
                          # l'esecuzione sul sensore
lb $s2, ($s1)            # carico su $s2 il byte indirizzato dal buffer da leggere

li $s3, 0                # inizializzo registro per somma parziale della conversione

```

```

lett_check:  blt $s2, 65, unknown_char    # se il contenuto di $s2 < A ( = 65 in ASCII )
                                          # non appartiene al dizionario di interesse

```

bgt \$s2, 66, unknown_char # se il contenuto di \$s2 > B (= 66 in ASCII)
 # non appartiene al dizionario di interesse

char_check: addi \$s1, \$s1, 1 # incremento l'indirizzo del buffer di lettura di una posizione
 sub \$t3, \$t3, 1 # decremento numero di valori da leggere
 lb \$s2, (\$s1) # carico su \$s2 il byte indirizzato dal buffer da leggere
 # incrementato di una posizione

beq \$s2, 32, conversion_end # se \$s2 contiene uno spazio la conversione
 # e' terminata, altrimenti itero
 beqz \$s2, conversion_end # se \$s2 contiene il valore zero, fine lettura da
 # input

slti \$t4, \$s2, 65 # se il contenuto di \$s2 < A (= 65 in ASCII), il
 # secondo elemento dell'input non e' una lettera
 beqz \$t4, is_letter
 addi \$s2, \$s2, -48 # converte da ASCII (in rappresentazione
 # esadecimale) a intero (in rappresentazione
 # decimale) l'elemento successivo (numero)
 j normalize_in

is_letter: slti \$t4, \$s2, 71 # se \$s2 < 71, \$t4 = 1, altrimenti \$t4 = 0
 beqz \$t4, unknown_char # se il contenuto di \$s2 >= G (= 71 in ASCII),
 # il secondo elemento dell'input non e' una
 # lettera del dizionario di interesse

 addi \$s2, \$s2, -55 # converte da ASCII (in rappresentazione
 # esadecimale) a intero (in rappresentazione
 # decimale) l'elemento successivo (lettera)

normalize_in: mul \$s3, \$s3, 16 # moltiplica per 16 per conversione in
 # base esadecimale
 add \$s3, \$s3, \$s2 # somma parziale della conversione
 j char_check

conversion_end: beqz \$s3, dist_NW # se il contenuto di \$s3, cioè d2(t)d3(t) = 0
 # ERRORE

 bgt \$s3, 50, dist_NW # se il contenuto di \$s3, cioè d2(t)d3(t) > 50
 # ERRORE

 move \$t6, \$s1 # copio contenuto di \$s1 buffer_IN_index, in \$t5
 sub \$t6, \$t6, 3 # decremento di due posizioni l'indirizzo del buffer di
 # lettura per tornare al primo elemento di ogni input
 # (= lettera 'A' o 'B')

 lb \$s2, (\$t6) # carico su \$s2 il byte indirizzato dal buffer \$t6 da
 # leggere

	beq \$s2, 66, save_mov_obj	# se il contenuto di \$s2 e' 'B' passo al # controllo di verifica per ostacoli mobili
	li \$s4, 2	# resetta numero di occorrenze consecutive di # ostacoli mobili a stessa distanza lecite
	j dist_W	
save_mov_obj:	beq \$s3, \$s5, dist_check	# se il contenuto di \$s3 equivale alla distanza # letta del sensore all'istante precedente, # salvato solamente in caso di ostacolo # mobile, controllo che ci siano ancora # occorrenze ostacoli mobili consecutivi
	sw \$s3, 20(\$sp)	# salvo il contenuto di \$s3, cioè d2(t)d3(t) su # SF alla locazione di memoria assegnata per # la memorizzazione del valore del sensore # all'istante precedente
	li \$s4, 2	# resetta numero di occorrenze consecutive di ostacoli mobili a stessa distanza lecite
	j dist_W	
dist_check:	sub \$s4, \$s4, 1	# decremento di un elemento registro contatore delle # occorrenze consecutive per ostacoli mobili
	blez \$s4, dist_error	# se il contenuto del registro \$s4 e' pari a zero, # allora non sono permesse nuove occorrenze # per ostacoli mobili a stessa distanza
	sw \$s3, 20(\$sp)	# salvo il contenuto di \$s3, cioè d2(t)d3(t) su # SF alla locazione di memoria assegnata per # la memorizzazione del valore del sensore # all'istante precedente
	j dist_W	
dist_error:	sw \$s3, 20(\$sp)	# salvo il contenuto di \$s3, cioè d2(t)d3(t) su SF alla # locazione di memoria assegnata per la memorizzazione # del sensore all'istante precedente
dist_NW:	li \$t4, 48	# \$t4 contiene il valore 0 in ASCII
	j write_dist	
dist_W:	li \$t4, 49	# \$t4 contiene il valore 1 in ASCII
write_dist:	sb \$t4, (\$t1)	# memorizzo il valore della variabile booleana nella terza # locazione di memoria dell'array di correttezza
	sb \$t4, (\$s7)	# memorizzo il byte contenuto su \$t4 nella locazione di # memoria puntata dal buffer di output

```

jal write_sensor      # salta a procedura di stampa del valore booleano

addi $s7, $s7, 1      # incremento buffer di uscita di una posizione

beqz $t3, close_dist   # se il numero di valori da leggere e' zero, chiude
                      # l'operazione sul sensore

```

```

li $t5, 32            # $t4 contiene lo spazio in codifica ASCII
sb $t5, ($s7)          # memorizzo il byte contenuto su $t4 nella locazione di
                      # memoria puntata dal buffer di output
jal write_sensor      # salta a procedura di stampa del carattere spazio
addi $s7, $s7, 1      # incremento posizione del buffer di output

```

```

nxt_val:    addi $s1, $s1, 1      # incremento di una posizione buffer_IN_index
                sub $t3, $t3, 1      # decremento numero di valori da leggere
                lb $s2, ($s1)        # carico su $s2 il byte indirizzato da buffer incrementato di
                                # una posizione
                beq $s2, 32, nxt_val  # se $s2 contiene uno spazio, scorro ancora,
                                # altrimenti salvo i registri e torno a main
                beqz $s2, close_dist  # se $s2 contiene 0, termino esecuzione sul sensore

                sw $s7, 4($sp)        # salvo su SF buffer_OUT_index
                sw $s1, 8($sp)        # salvo su SF buffer_IN_index
                sw $s4, 16($sp)       # salvo su SF numero occorrenze residue ostacoli
                                # mobili permessi
                sw $t3, 28($sp)       # salvo su SF numero di valori residui da leggere in
                                # input per Dist

                lw $ra, 0($sp)        # ripristino da SF registro di ritorno

                addi $sp, $sp, 84     # dealloco 84 byte da stack

                jr $ra                # salto a istruzione contenuta in $ra

```

```

P1:    sub $sp, $sp, 96          # Alloco 96 byte su stack
                sw $ra, 0($sp)        # salvo su SF indirizzo di ritorno

                move $s3, $a1         # salvo su $t1 indirizzo dell'array di correttezza passato
                                # come argomento

                lb $t1, ($s3)         # carico su $t1 primo valore array correttezza
                lb $t2, 4($s3)        # carico su $t2 secondo valore array correttezza
                lb $t3, 8($s3)        # carico su $t3 terzo valore array correttezza

                la $a0, pol_1_path    # filepath_OUT per scrittura su file
                jal open_file_w       # salta a procedura di scrittura su file
                move $s0, $v0         # $s0 contiene file_OUT_Descriptor

```

sw \$s0,4(\$sp) # salvo su SF file_OUT_Descriptor per P1

la \$s1, pol_1_buffer # carico su \$s1 indirizzo di partenza del buffer_OUT per P1

j P1_ex

loop_P1: sub \$sp,\$sp, 96 # Alloco 96 byte su stack per ripristinare dati locali
precedentemente memorizzati

sw \$ra, 0(\$sp) # salvo su SF indirizzo di ritorno
lw \$s0, 4(\$sp) # carico da SF file_OUT_Descriptor per P1
lw \$s1, 8(\$sp) # carico da SF buffer_OUT_index per P1

move \$s3, \$a1 # salvo su \$s3 indirizzo dell'array di correttezza
passato come argomento

lb \$t1, (\$s3) # carico su \$t1 primo valore array correttezza
lb \$t2, 4(\$s3) # carico su \$t2 secondo valore array correttezza
lb \$t3, 8(\$s3) # carico su \$t3 terzo valore array correttezza

P1_ex: and \$t4, \$t1, \$t2 # AND bitwise tra primo e secondo valore correttezza
and \$t4, \$t4, \$t3 # AND bitwise tra risultato precedente e terzo valore
correttezza

sb \$t4, (\$s1) # memorizzo il byte contenuto su \$t4 nella locazione
di memoria puntata dal buffer di output
jal write_policy # salta a procedura di stampa del valore booleano nel
file di output
addi \$s1, \$s1, 1 # incremento buffer di uscita di una posizione

beqz \$a3, exit_p1 # se il contenuto del registro \$a3, passato come
argomento e con funzione di 0-flag per identificare
chiusura dell'ultimo sensore e' zero, chiudo la P1

li \$t5, 32 # \$t4 contiene lo spazio in codifica ASCII
sb \$t5, (\$s1) # memorizzo il byte contenuto su \$t5 nella locazione
di memoria puntata dal buffer di output

jal write_policy # salta a procedura di stampa del carattere spazio
addi \$s1, \$s1, 1 # incremento buffer di uscita di una posizione

sw \$s1,8(\$sp) # salvo su SF buffer_OUT_index per P1
lw \$ra, 0(\$sp) # ripristino da SF indirizzo di ritorno del chiamante
addi \$sp,\$sp, 96 # dealloco 96 byte da stack
jr \$ra # salto a istruzione contenuta in \$ra

P2: sub \$sp,\$sp, 108 # Alloco 108 byte su stack
sw \$ra, 0(\$sp) # salvo su SF indirizzo di ritorno

move \$s3, \$a1	# salvo su \$s3 indirizzo dell'array di correttezza # passato come argomento
lb \$t1, (\$s3)	# carico su \$t1 primo valore array correttezza
lb \$t2, 4(\$s3)	# carico su \$t2 secondo valore array correttezza
lb \$t3, 8(\$s3)	# carico su \$t3 terzo valore array correttezza
la \$a0, pol_2_path	# filepath_OUT per scrittura su file
jal open_file_w	# salta a procedura di scrittura su file
move \$s0, \$v0	# \$s0 contiene file_OUT_Descriptor
sw \$s0, 4(\$sp)	# salvo su SF file_OUT_Descriptor per P2
la \$s1, pol_2_buffer	# carico su \$s1 indirizzo di partenza del buffer_OUT per P2
sw \$s1, 8(\$sp)	# salvo su buffer_OUT_per P2
j P2_ex	

loop_P2:	sub \$sp, \$sp, 108	# Alloco 108 byte su stack per ripristinare dati locali # precedentemente memorizzati
	sw \$ra, 0(\$sp)	# salvo su SF indirizzo di ritorno
	lw \$s0, 4(\$sp)	# carico da SF file_OUT_Descriptor per P2
	lw \$s1, 8(\$sp)	# carico da SF buffer_OUT_index per P2
	move \$s3, \$a1	# salvo su \$s3 indirizzo dell'array di correttezza # passato come argomento
	lb \$t1, (\$s3)	# carico su \$t1 primo valore array correttezza
	lb \$t2, 4(\$s3)	# carico su \$t2 secondo valore array correttezza
	lb \$t3, 8(\$s3)	# carico su \$t3 terzo valore array correttezza

P2_ex:	and \$t5, \$t1, \$t2	# corr_P1 AND corr_P2
	xor \$t6, \$t1, \$t2	# corr_P1 XOR corr_P2 = P1P2' + P1'P2
	and \$t7, \$t3, \$t6	# corr_P3 AND (P1P2' + P1'P2)
	or \$t5, \$t5, \$t7	# (corr_P1 AND corr_P2) OR (corr_P3 AND (P1P2' + P1'P2))
	sb \$t5, (\$s1)	# memorizzo il byte contenuto su \$t5 nella locazione di # memoria puntata dal buffer di output
	jal write_policy	# salta a procedura di stampa del valore booleano
	addi \$s1, \$s1, 1	# incremento buffer di uscita di una posizione
	beqz \$a3, exit_p2	# se il contenuto del registro \$a3, passato come argomento # e con funzione di 0-flag per identificare chiusura # dell'ultimo sensore e' zero, chiudo P2
	li \$t5, 32	# \$t5 contiene lo spazio in codifica ASCII
	sb \$t5, (\$s1)	# memorizzo il byte contenuto su \$t5 nella locazione di # memoria puntata dal buffer di output

jal write_policy	# salta a procedura di stampa del carattere spazio
addi \$s1, \$s1, 1	# incremento buffer di uscita di una posizione
sw \$s1, 8(\$sp)	# salvo su SF file_OUT_Descriptor per P2
lw \$ra, 0(\$sp)	# ripristino da SF indirizzo di ritorno
addi \$sp, \$sp, 108	# dealloco 108 byte da stack
jr \$ra	# salto a istruzione contenuta in \$ra

P3:

sub \$sp, \$sp, 120	# Alloco 120 byte su stack
sw \$ra, 0(\$sp)	# salvo su SF indirizzo di ritorno
move \$s3, \$a1	# salvo su \$t1 indirizzo dell'array di correttezza passato # come argomento
lb \$t1, (\$s3)	# carico su \$t1 primo valore array correttezza
lb \$t2, 4(\$s3)	# carico su \$t2 secondo valore array correttezza
lb \$t3, 8(\$s3)	# carico su \$t3 terzo valore array correttezza
la \$a0, pol_3_path	# filepath_OUT per scrittura su file
jal open_file_w	# salta a procedura di scrittura su file
move \$s0, \$v0	# \$s0 contiene file_OUT_Descriptor
sw \$s0, 4(\$sp)	# salvo su SF file_OUT_Descriptor per P3
la \$s1, pol_3_buffer	# carico su \$s1 indirizzo di partenza del buffer_OUT per P3
sw \$s1, 8(\$sp)	# salvo su SF buffer_OUT_index per P3
j P3_ex	

loop_P3:

sub \$sp, \$sp, 120	# Alloco 108 byte su stack per ripristinare dati # locali precedentemente memorizzati
sw \$ra, 0(\$sp)	# salvo su SF indirizzo di ritorno
lw \$s0, 4(\$sp)	# carico da SF file_OUT_Descriptor per P3
lw \$s1, 8(\$sp)	# carico da SF buffer_OUT_index per P3
move \$s3, \$a1	# salvo su \$t1 indirizzo dell'array di # correttezza passato come argomento
lb \$t1, (\$s3)	# carico su \$t1 primo valore array correttezza
lb \$t2, 4(\$s3)	# carico su \$t2 secondo valore array correttezza
lb \$t3, 8(\$s3)	# carico su \$t3 terzo valore array correttezza

P3_ex:

or \$t4, \$t2, \$t3	# OR bitwise tra primo e secondo valore correttezza
or \$t5, \$t4, \$t1	# OR bitwise tra risultato precedente e terzo valore correttezza

sb \$t5, (\$s1)	# memorizzo il byte contenuto su \$t5 nella locazione # di memoria puntata dal buffer di output
jal write_policy	# salta a procedura di stampa del valore booleano nel file di output
addi \$s1, \$s1, 1	# incremento buffer di uscita di una posizione
beqz \$a3, exit_p3	# se il contenuto del registro \$a3, passato come argomento # e con funzione di 0-flag per identificare chiusura # dell'ultimo sensore e' zero, chiudo P3
li \$t5, 32	# \$t4 contiene lo spazio in codifica ASCII
sb \$t5, (\$s1)	# memorizzo il byte contenuto su \$t5 nella locazione di # memoria puntata dal buffer di output
jal write_policy	# salta a procedura di stampa del carattere spazio
addi \$s1, \$s1, 1	# incremento buffer di uscita di una posizione
sw \$s1, 8(\$sp)	# salvo su SF file_OUT_Descriptor per P3
lw \$ra, 0(\$sp)	# ripristino da SF indirizzo di ritorno
addi \$sp, \$sp, 120	# dealloco 96 byte da stack
jr \$ra	# salto a istruzione contenuta in \$ra

Funzioni di supporto

printf:	li \$v0, 4	# system call per stampa stringa
	syscall	
	jr \$ra	

unknown_char:	la \$a0, unknwn_char
	jal printf

open_file_r:	li \$v0, 13	# system call per apertura del file
	li \$a1, 0	# flag per lettura
	li \$a2, 0	# modalita e' ignorata
	syscall	# chiamata di sistema
	jr \$ra	

open_file_w:	li \$v0, 13	# system call per apertura del file di output
	li \$a1, 1	# flag per scrittura
	li \$a2, 0	# modalita ignorata
	syscall	# chiamata di sistema
	jr \$ra	

read_from_file:	li \$v0, 14 move \$a0, \$s0 li \$a2, 300 syscall jr \$ra	# system call per lettura del file # descrittore del file # numero massimo di valori da leggere # chiamata di sistema
write_policy:	li \$v0, 15 move \$a0, \$s0 move \$a1, \$s1 li \$a2, 1 syscall jr \$ra	# system call per scrittura su file # descrittore del file # indirizzo a partire dal quale scrivere # numero di caratteri da scrivere # syscall per scrittura su file
write_sensor:	li \$v0, 15 move \$a0, \$t2 move \$a1, \$s7 li \$a2, 1 syscall jr \$ra	# system call per scrittura su file # descrittore del file OUT # indirizzo a partire dal quale scrivere # dimensione buffer # syscall per scrittura su file
close_slope:	lw \$t2, 16(\$sp) lw \$s0, 12(\$sp) jal close_sensor la \$a0, completed_S jal printf lw \$t3, 20(\$sp) lw \$s1, 8(\$sp) lw \$s7, 4(\$sp) lw \$ra, 0(\$sp) addi \$sp, \$sp, 24 jr \$ra	# ripristino da SF descrittore file OUT # ripristino da SF descrittore file IN # salta a procedura di chiusura operazioni I/O # Stampa messaggio di completamento # ripristino da SF numero di valori da leggere # ripristino da SF buffer_IN_index # ripristino da SF buffer_OUT_index # ripristino da SF indirizzo di ritorno # Dealloca lo stack utilizzato
close_steer:	lw \$t2, 20(\$sp) lw \$s0, 16(\$sp) jal close_sensor la \$a0, completed_St jal printf lw \$t3, 24(\$sp) lw \$s1, 12(\$sp) lw \$s7, 8(\$sp) lw \$s5, 4(\$sp) lw \$ra, 0(\$sp)	# ripristino da SF descrittore file OUT # ripristino da SF descrittore file IN # salta a procedura di chiusura operazioni I/O # Stampa messaggio di completamento # ripristino da SF numero di valori da leggere # ripristino da SF buffer_IN_index # ripristino da SF buffer_OUT_index # ripristino da SF valore sensore istante precedente # ripristino da SF indirizzo di ritorno per chiamante

	addi \$sp,\$sp, 52	# Dealloca lo stack utilizzato
	jr \$ra	
close_dist:	lw \$t2, 24(\$sp)	# ripristino da SF descrittore file OUT
	lw \$s0, 12(\$sp)	# ripristino da SF descrittore file IN
	jal close_sensor	# salta a procedura di chiusura operazioni I/O
	la \$a0, completed_D	
	jal printf	
	lw \$t3, 28(\$sp)	# ripristino da SF numero di valori da leggere
	lw \$s5, 20(\$sp)	# ripristino da SF valore s(t-1)
	lw \$s4, 16(\$sp)	# ripristino da SF numero occorrenze residue
		# ostacoli mobili permessi
	lw \$s1, 8(\$sp)	# ripristino da SF buffer_IN_index
	lw \$s7, 4(\$sp)	# ripristino da SF buffer_OUT_index
	lw \$ra, 0(\$sp)	# ripristino da SF indirizzo di ritorno per chiamante
	addi \$sp,\$sp, 84	# Dealloca lo stack utilizzato
	move \$v0, \$zero	# restituisce zero come risultato per indicare
		# terminazione operazioni di I/O sui sensori
	jr \$ra	
close_sensor:	li \$v0, 16	# system call per chiusura del file di output
	move \$a0, \$t2	# descrittore del file di OUTPUT
	syscall	
	li \$v0, 16	# system call per chiusura del file di input
	move \$a0, \$s0	# descrittore del file di INPUT
	syscall	
	jr \$ra	
exit_p1:	li \$t8, 1	
	j exit_p	
exit_p2:	li \$t8, 2	
	j exit_p	
exit_p3:	li \$t8, 3	
exit_p:	lw \$s0, 4(\$sp)	# dealloco da SF file_OUT_Descriptor di P1/P2/P3...
	lw \$s1, 8(\$sp)	# dealloco da SF buffer_OUT_index di P1/P2/P3...
	li \$v0, 16	# system call per chiusura del file di output
	move \$a0, \$s0	# descrittore del file di OUTPUT

syscall

lw \$ra, 0(\$sp) # ripristino da SF indirizzo di ritorno

beq \$t8, 1, incr_P1 # opportuno incremento per corretta deallocazione P1

beq \$t8, 2, incr_P2 # opportuno incremento per corretta deallocazione P2

beq \$t8, 3, incr_P3 # opportuno incremento per corretta deallocazione P3

incr_P1: addi \$sp, \$sp, 96 # dealloco stack di 96 byte
 j end_exitP

incr_P2: addi \$sp, \$sp, 108 # dealloco stack di 108 byte
 j end_exitP

incr_P3: addi \$sp, \$sp, 120 # dealloco stack di 120 byte

end_exitP: move \$v0, \$zero # imposto il contenuto di \$v0 a zero
 jr \$ra # salto a istruzione contenuta in \$ra

exit: la \$a0, complete # stampa stringa di terminazione programma
 jal printf
 li \$v0, 10 # Chiude il programma
 syscall