Monitoraggio di sensori in Smart Veichles

Progetto Assembly MIPS

per il corso di Architetture degli elaboratori

- A. A. 2017/2018 -

di Leonardo Bitto e Erica Giardina

mail: leonardo.bitto@stud.unifi.it

erica.giardina@stud.unifi.it

consegnato il 03/06/18

Indice

Introduzione

La specifica del progetto richiede che siano presi in esame dal codice dei file di testo con estensione .txt. Si pone quindi il problema dell'elaborazione dei dati contenuti nel file di testo, la gestione dei file viene delegata dal programma al sistema operativo , nel nostro caso a SPIM.

SPIM è un simulatore del processore MIPS che mette a disposizione un rudimentale sistema operativo che gestisce i file presenti nel file system. L'intero programma è stato scritto e testato con Qtspim.

Syscalls per gestire file di testo in MIPS

I file di testo vengono letti come serie di caratteri ASCII, ognuno dei quali richiede un byte di memoria. Quando vengono caricati in un buffer in memoria, i byte possono poi essere presi in esame una alla volta. Le syscalls richiedono dei valori necessari a specificare quale comando e con quali parametri viene chiamato il comando **syscall**.

Per poter caricare nel buffer la serie di caratteri prima è necessario specificare quale file debba essere aperto tramite la syscall *apri*, finito di usare il file può essere chiuso tramite *chiudi*:

Nome	**Syscall ($v0)**	**Inputs**	**Output**
apri	13	$a0 = nome del file	$v0 = descrittore del file
		$a1 = flag lettura	
		$a2 = modalità	
chiudi	16	$a0 = descrittore del file	$v0 = 0

Il nome del file viene specificato nel segmento dei dati in memoria e caricato nel registro tramite **la**. La syscall si aspetta il percorso completo dalla cartella home o C: fino al file che si intende caricare. Il descrittore del file è un numero che risulta utile per identificare il file aperto, una volta aperto può essere letto o scritto. La lettura richiede l'attivazione del flag $a1 durante l'apertura del file.

Nome	**Syscall ($v0)**	**Inputs**	**Output**
lettura	14	$a0 = descrittore del file	$v0 = byte letti
		$a1 = buffer	
		$a2 = numero di caratteri da leggere	
scrittura	15	$a0 = descrittore del file	$v0 = byte scritti
		$a1 = buffer	
		$a2 = numero di caratteri da scrivere	

Formato di input e output

Ciascuno dei sensori, misurando diversi aspetti del sistema ha specifiche di funzionamento diverse, dal testo del progetto:

> "Sensore di Pendenza: p(t) deve essere sempre compreso nell'intervallo -60 < p(t) < 60. Se questa condizione è verificata al tempo t allora il sensore è considerato come correttamente funzionante (valore di correttezza corrp(t)=1), altrimenti come malfunzionante (corrp(t)=0).
>
> Sensore di Sterzo: il valore di 0 < s(t) < 100 letto non può differire di più di 10 gradi rispetto al valore letto dal sensore all'istante precedente, ovvero |s(t) – s(t-1)| ≤ 10. Se questa condizione è verificata al tempo t allora il sensore è considerato come correttamente funzionante (valore di correttezza corrs(t)=1), altrimenti come malfunzionante (corrs(t)=0).
>
> Sensore di Distanza da Ostacoli: il sensore ha una portata massima di 50 metri. Il sensore sarà reputato come malfunzionante (quindi corrd(t)=0) in questi casi: o I valore di distanza rilevato al tempo t, ovvero il valore decimale di [d2(t)d3(t)]16, è superiore a 50 metri (es. d = B38, corrispondente a 56 metri di distanza), oppure o Al tempo t viene rilevato un ostacolo (di qualsiasi tipologia) con distanza zero (es. A00, corrispondente a 0 metri di distanza), oppure o Il sensore trasmette lo stesso valore di distanza da ostacoli mobili per più di due misurazioni consecutive (ad esempio se d(1)=B20, d(2)=B20 e d(3)=B20, allora corrd(1)=1, corrd(2)=1, e corrd(3)=0)."
>
> - *pag 2 della specifica del progetto*

Possiamo quindi definire quali caratteri ASCII in input aspettarci, in particolare:

- i numeri da 0 a 9, ogni sensore;
- lo spazio, ogni sensore;
- le lettere A, B, C, D, E, F, , solo per il sensore di Distanza ostacoli;
- il carattere meno (-), solo per il sensore di Pendenza;
- il carattere ASCII 0x0a, che indica la fine del file di input

Nel codice non vengono gestiti gli errori, il codice semplicemente passa al prossimo byte da esaminare. L'interpretazione dei caratteri si dovrà limitare a questa selezione dato che:

> Si può assumere che le sequenze dei valori dei sensori contenute nei files di input siano sintatticamente corrette (formato corretto).
>
> *-pag 5 della specifica del progetto*

Il formato dei file di testo è corretto se ha queste caratteristiche:

- Al massimo tre caratteri alternati a uno spazio, presente anche dopo l'ultimo valore;
- Ogni lettera presente deve essere maiuscola;
- La serie di valori in input deve terminare con un line feed.

I valori utilizzati in input sono stati generati casualmente da generatori di liste di numeri trovati su internet, eventualmente modificati per creare i casi di malfunzionamento necessari a testare i sensori.

Interpretatore ASCII-numero

Avendo copiato il contenuto del file di testo nel buffer possiamo passare all'interpretazione del suo contenuto byte per byte. Secondo il formato corretto del file di input di ogni sensore si ha che il primo carattere può essere il segno negativo, una lettera A o B oppure una cifra, i restanti due caratteri contengono altre cifre in base decimale o esadecimale, nel caso del sensore di distanza. La presenza di valori con basi differenti richiede una scelta della base su cui operare il calcolo del valore. Il nome della funzione è *converter,* di seguito il codice:

```
############ Code Segment ##############
.text
.globl converter

 converter:

        # controlla se è un simbolo negativo
        li $t3, 0x2d          # carica il valore esadecimale 0x2d in t3 (-)
        bne $a0, $t3, base    # se il byte è diverso dal segno negativo 0x2d
                              # allora calcola la base,
        li $a3, 1             # flag per numero negativo
        j AScontinue          # torna a loop

        # decidi con quale base operare
   base: beq $a2, $zero, decimal    # se il flag è spento allora decimale
          li $t5, 16                # imposta la base a sedici
          j lettera                 # controlla se è una lettera
decimal: li $t5, 10                 # metti la base a dieci
        addi $a0, $a0, -0x30    # sottrai al carattere ascii il valore 0x30
        j labelnum              # salta al calcolo del numero

        # SCEGLI ROUTINE NEL CASO SIA UN NUMERO O UNA LETTERA da convertire
 lettera: li $t3, 0x39
        bgt $a0, $t3, nodigit
        blt $a0, $zero, AScontinue
        sub $a0, $a0, 0x30  # 0x30 è un fattore necessario a convertire da ascii decimale
                            # solo nel caso dei numeri minori di 9
      # calcolo numero  $t8 viene usato per calcolare il numero
labelnum:
        mul $t8, $t8, $t5       # moltiplica per 10 $t8 = $t8 * 10
        beq $a3, 1, negativo    # se il flag negativo è attivo (1)
```

```
32.                     # allora esegui il calcolo come se fosse
33.                     # un numero negativo sennò
34.         add $t8, $t8, $a0       # $t8 = $t8 + digit
35.         j continue          # e vai a continue
36.  negativo: sub $t8, $t8, $a0       # $t8 = $t8 - digit
37.
38.  AScontinue:     move $v0, $t8          # sposta valore in registro out
39.                 jr $ra
40.   nodigit: sub $a0, $a0, 65   # valore decimale in codifica ASCII
41.         addi $a0, $a0, 10   # riporta al valore decimale la lettera in esadecimale
42.         j labelnum
```

La funzione che interpreta il codice ASCII prende un byte in entrata, che rappresenta una cifra e il suo funzionamento viene modificato da tre registri che funzionano da flag:

- $a0, byte da interpretare
- $a2, flag che indica su che base operare
- $a3, flag che indica che il numero è negativo

Il flag del numero negativo si imposta automaticamente quando viene letto il simbolo ASCII del segno negativo che in codifica esadecimale equivale a 0x2d, la funzione quindi torna al chiamante e si aspetta che i prossimi bit rappresentano un numero negativo (righe 8-12).
Il flag della base invece deve essere impostato prima della chiamata al convertitore, con il registro $a2 = 0 si opera in decimale, nel caso dei sensori di pendenza e sterzo, la base esadecimale viene usata nel caso del sensore di distanza. Nel calcolo della base su cui operare (righe 15-20), in caso di base decimale la funzione converte il numero da ASCII a cifra decimale e richiama la routine di calcolo del numero (righe 29-38) restituendolo in $v0. Nel caso sia richiesto il calcolo con base esadecimale la cifra in entrata deve essere esaminata per verificare se è una lettera o una cifra (righe 23-26), secondo la codifica ASCII presente nella tabella di seguito, possiamo vedere che i numeri sono rappresentati da i codici da 0x30 a 0x39 a cui è necessario togliere il valore 0x30 per ottenere la cifra, mentre nel caso sia una lettera (righe 42-44), possiamo verificare tramite la tabella che i codici delle lettere vanno da $65_{decimale}$ per A a $70_{decimale}$ per F, che indicano i valori da 10 a 15. Possiamo quindi sottrarre 65 e aggiungere 10. Con la lettera D, che ha codice ASCII decimale 68, sottraendo 65 si ottiene 3, aggiungendo 10 si ottiene 13.

0 1 2 3 4 5 6 7 8 9 10 11 12 **13** 14 15 valori decimali
0 1 2 3 4 5 6 7 8 9 A B C **D** E F valori esadecimali corrispondenti

Ottenuto la singola cifra si può passare al calcolo completo. La specifica richiede, per ogni sensore, di calcolare numeri che hanno al massimo due cifre espresse in 2 byte, in ordine la cifra più significativa seguita da quella meno significativa. SI utilizza la variabile $t8 per contenere il valore del primo byte, contenente la cifra più significativa, che viene moltiplicato per la base scelta. Al

registro $t8 viene sommato il valore del secondo byte contenente la seconda cifra(righe 34-37). Nel caso il flag in $a3 sia acceso si sottrae la cifra contenuta nel secondo byte(riga 37).
Quando viene letto un byte contenente un spazio, codice esadecimale 0x20, possiamo considerare il valore in input completamente letto, quindi il numero presente in $v0 può essere elaborato dalla condizione di funzionamento del sensore. La variabile $t8 deve essere azzerata ogni volta che viene incontrato uno spazio per evitare di sommare il precedente numero calcolato al successivo, $v0 non ha bisogno di questo trattamento in quanto viene riscritto dalle funzioni dei sensori.
Dato che che la chiamata della condizione viene scandita dallo spazio richiede che nei file di input l'ultimo valore da calcolare debba essere seguito da uno spazio per essere calcolato.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`	128	80	Ç	160	A0	á	192	C0	└	224	E0	α
1	01	Start of heading	33	21	!	65	41	A	97	61	a	129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
2	02	Start of text	34	22	"	66	42	B	98	62	b	130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
3	03	End of text	35	23	#	67	43	C	99	63	c	131	83	â	163	A3	ú	195	C3	├	227	E3	π
4	04	End of transmit	36	24	$	68	44	D	100	64	d	132	84	ä	164	A4	ñ	196	C4	─	228	E4	∑
5	05	Enquiry	37	25	%	69	45	E	101	65	e	133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
6	06	Acknowledge	38	26	&	70	46	F	102	66	f	134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
7	07	Audible bell	39	27	'	71	47	G	103	67	g	135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
8	08	Backspace	40	28	(	72	48	H	104	68	h	136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i	137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j	138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k	139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l	140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m	141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n	142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o	143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
16	10	Data link escape	48	30	0	80	50	P	112	70	p	144	90	É	176	B0	░	208	D0	╨	240	F0	≡
17	11	Device control 1	49	31	1	81	51	Q	113	71	q	145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
18	12	Device control 2	50	32	2	82	52	R	114	72	r	146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
19	13	Device control 3	51	33	3	83	53	S	115	73	s	147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
20	14	Device control 4	52	34	4	84	54	T	116	74	t	148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u	149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v	150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
23	17	End trans. block	55	37	7	87	57	W	119	77	w	151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
24	18	Cancel	56	38	8	88	58	X	120	78	x	152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
25	19	End of medium	57	39	9	89	59	Y	121	79	y	153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z	154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{	155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|	156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}	157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~	158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	⌂	159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Illustrazione 1: - Tabella Ascii

Segmento dati e stack

I valori in input per i sensori sono contenuti in file di testo che, per essere aperti, devono essere indirizzati tramite un indirizzo contenuto come stringa ASCII terminata, allocata tramite la direttiva **.asciiz**, è necessario specificare l'intero percorso. Il processo di ogni sensore si occupa di aprire e copiare il contenuto del proprio file nel bufferIN, questo spazio può essere condiviso in quanto le chiamate ai sensori sono sequenziali, appena il primo sensore ha finito di leggere e elaborare i valori in input lo spazio che li conteneva può essere riscritto per ospitare i dati del prossimo sensore.
Il buffer viene quindi dimensionato secondo il numero massimo di byte che è possibile avere in un file di testo input di ogni sensore. Nel segmento dati di ogni file delle politiche il buffer di input viene chiamato con l'etichetta **bufferIN**, ogni carattere ASCII viene identificato come un valore esadecimale a due cifre che viene contenuto in un byte. Sapendo come sono formattati i file di input possiamo calcolare l'area di memoria.
Il file di input deve avere 100 valori da esaminare e uno spazio tra ogni valore, più quello finale, saranno quindi necessari almeno 100 byte per gli spazi. Esaminando le specifiche date si può assumere che ogni valore sarà composto da al massimo tre caratteri, un segno negativo o una lettera e due o una cifra. Il dimensionamento deve coprire il caso peggiore ovvero quello in cui tutti i valori dei file di input abbiano il massimo numero di cifre possibili. Questo vuol dire che ogni valore in input è composto da 3 caratteri per ognuno dei 100 input avendo quindi 300 byte totali sommando quelli dello spazio e aggiungendo ulteriore spazio per ottenere un contorno separatore tra dati in memoria si ottiene il valore di 420 byte necessari per il bufferIN.

```
############ Data Segment ##############
.data
 bufferIN:  .space 420
 bufferOUT1: .space 250
 bufferOUT2: .space 250
 bufferOUT3: .space 250
 OUT1: .asciiz "/home/leonardo/Documenti/ASSEMBLY/Project_ADE_17_18_consegna/correttezzaP1.txt"
 OUT2: .asciiz "/home/leonardo/Documenti/ASSEMBLY/Project_ADE_17_18_consegna/correttezzaP2.txt"
 OUT3: .asciiz "/home/leonardo/Documenti/ASSEMBLY/Project_ADE_17_18_consegna/correttezzaP3.txt"

 INp:  .asciiz "/home/leonardo/Documenti/ASSEMBLY/Project_ADE_17_18_consegna/pendenzaIN.txt"
 OUTp: .asciiz "/home/leonardo/Documenti/ASSEMBLY/Project_ADE_17_18_consegna/correttezzaPendenzaOUT.txt"
 bufferOUTp: .space 250

 INs:  .asciiz "/home/leonardo/Documenti/ASSEMBLY/Project_ADE_17_18_consegna/sterzoIN.txt"
 OUTs: .asciiz "/home/leonardo/Documenti/ASSEMBLY/Project_ADE_17_18_consegna/correttezzaSterzoOUT.txt"
 bufferOUTs: .space 250

 INd:  .asciiz "/home/leonardo/Documenti/ASSEMBLY/Project_ADE_17_18_consegna/distanzaIN.txt"
 OUTd: .asciiz "/home/leonardo/Documenti/ASSEMBLY/Project_ADE_17_18_consegna/correttezzaDistanzaOUT.txt"
 bufferOUTd: .space 250
```

Nel caso del bufferOUT si rende necessario allocare uno spazio per ogni file output da scrivere. Possiamo seguire un ragionamento simile per il dimensionamento dei buffer di uscita di ogni file, non solo quello delle politiche. Ogni file ha un valore di uscita per ogni valore in entrata e essendo composto solo da un byte, seguito sempre da uno spazio abbiamo la necessità di 200 byte per i buffer di uscita. Aggiungiamo 50 byte per contorno e otteniamo 250 byte necessari.
Nel codice viene allocato lo spazio in memoria tramite la direttiva **.space** che riserva il numero di byte che gli viene passato di seguito i valori:

bufferIN, 420 byte equivalenti a 105 parole;

bufferOUT, 220 byte equivalenti a 55 parole;

Lo stack viene utilizzato per mantenere l'indirizzo di ritorno alla funzione delle politiche contenuto nel registro $ra. Il registro viene riscritto dalle funzioni che interpreta il byte e che verificano il funzionamento del sensore, viene quindi caricato sullo stack prima di entrare nel loop e viene caricato dallo stack prima di saltare al chiamante.
Nel segmento della memoria riservato ai dati sono necessarie 2 sezioni di memoria, Il buffer di input e il buffer di output.

Infine per ogni sensore e politica è necessario un indirizzo al file di output e per i sensori è necessario specificare il percorso al file di input. I percorsi vengono inseriti con la direttiva .**asciiz** e richiedono il percorso completo per il file di testo in input e output.

Pseudo-codice

La logica dietro l'intero programma è molto semplice, vengono chiamati i tre processi dei tre sensori e poi si va a unire i risultati nelle varie politiche riempiendo i gli appositi buffer e scrivendo nei file di testo i valori ottenuti in output.

```
Chiamata a pendenza
flush dei registri
Chiamata a sterzo
flush dei registri
Chiamata a distanza
flush dei registri
Politica 1
Politica 2
Politica 3
fine del programma
```

Sensori

I processi dei sensori vengono richiamati all'inizio del codice e si occupano della gestione dei file di testo e della lettura e scrittura del buffer, un altro processo gestisce i valori in input e applica le condizioni di funzionamento dei sensori. Le procedure vengono chiamate sequenzialmente nei file delle politiche, per questo è possibile usare i registri $s0-$s7 in tutti i file dei sensori senza creare dei conflitti, è possibile fare un ragionamento simile anche per le condizioni dei sensori e il convertitore ASCII-numero che utilizzano i registri $t0-$t9.

Codice dei sensori

Pseudo-codice

I file dei sensori si occupano di aprire e chiudere i file di input/output e di leggere/scrivere il loro contenuto negli appositi buffer. Tra le operazioni di Lettura/scrittura ogni sensore ha un ciclo addetto al caricamento di un byte che deve essere interpretato da ASCII a numero, il risultato viene passato alla funzione che applica le condizioni di funzionamento del sensore e scrive il suo risultato nel buffer di uscita.
Il buffer di input finisce con il valore 0x0a che corrisponde a un 'a capo' o line feeder. Il line feeder e lo spazio scandiscono le azioni che devono essere compiute.
Nel caso venga caricato uno spazio si può assumere che il valore sia già stato letto interamente quindi è possibile passarlo alla funzione che elabora il valore ottenendo un 1 o uno 0 che andrà scritto nel buffer di uscita.
Il line feeder indica la fine dei valori da interpretare e che il buffer di output può essere scritto nel file di uscita, per poi tornare alla funzione chiamante.
I codici hanno una struttura simile per ogni sensore, di seguito lo pseudocodice :

```
apri file input
leggi file input, carica in bufferIN
chiudi file input
ciclo :
        carica byte da bufferIN
                se viene caricato il line feeder si esce dal ciclo
                se viene caricato uno spazio passa alla condizione del sensore
        interpreta byte e torna al ciclo
        condizione funzionamento sensore
        scrivi in bufferOUT
apri file output
scrivi in file output, carica da bufferOUT
chiudi file output
salto al chiamante
```

Nonostante lo pseudo-codice dia l'idea di come funzionano questi pezzi di codice, ci sono delle differenze da notare in ogni versione del codice.

Sensore pendenza

```
############ Code Segment ##############
.text
.globl SenPendenza

 SenPendenza:
    # apri il file degli input (ottenere il descrittore del file in $v0)

    li $v0, 13       # codice del sistema operativo per aprire file txt
    la $a0, INp      # nome del file da caricare
    li $a1, 0        # flag per scrittura
    li $a2, 0        # flag per modalità
    syscall          # chiamata al sistema

    # leggi dal file appena aperto,

    move $a0, $v0    # metti nel parametro di entrata descrittore del file
                     # indirizzo del buffer in cui copiare il file di testo
    la $a1, bufferIN     # viene specificato dalla funzione chiamante come parametro di
                                                ingresso
    li $a2, 500      # specifica la quantità di caratteri nel buffer
    li $v0, 14       # codice di chiamata per lettura
    syscall          # chiamata al sistema

    # chiudi il file IN
    move $a0, $v0
    li $v0, 16
    syscall

    li $v0, 0        # azzeramento di v0 in quanto usato successivamente

    li $s1, 0        # inizializzazione contatore bufferIN
    li $s2, 0        # inizializzazione contatore bufferOUT
    li $s6, 0x0a     # carica valore di fine buffer
    li $s7, 0x20     # carica il valore esadecimale 0x20 in t9 (spazio)

    # ALLOCAZIONE NELLO STACK DELL'INDIRIZZO DI RITORNO

    addi $sp, $sp, -4
```

```
38.     sw $ra, 0($sp)
39.
40.      Ploop:
41.         add $a0, $s5, $s1
42.         lb $a0, ($a0)    # carica  il byte da esaminare
43.         beq $a0, $s6, PFINE       # se questo equivale al terminatore 0x0a
44.                             # allora si passa a FINE
45.
46.         # check se ha trovato uno spazio
47.         bne $a0, $s7, Pconv       # se il byte caricato in a0 non è 20 vai a conv
48.
49.         move $a0, $v0           # metti come argomento di entrata l'uscita del
50.                             # convertitore ASCII-numero
51.         jal pendenza           # chiamata al modulo condPendenza.s
52.
53.         # conversione in ASCII del risultato e
54.         # scrittura nel bufferOUT
55.         addi $v0, $v0, 48       # ritrasforma in codice ASCII i valori 1 && 0
56.         sb $v0, bufferOUTp($s2)        # metti il valore trovato nella posizione
57.                             # a cui siamo arrivati nel bufferOUT($s2)
58.         addiu $s2, $s2, 1       # aumenta di 1 il contatore del bufferOUT
59.         sb $s7, bufferOUTp($s2)        # aggiungi uno spazio, codice 20 in bufOUT($s2)
60.         addiu $s2, $s2, 1       # aumenta di 1 il contatore del bufferOUT
61.
62.         # azzeramento registri
63.         # una volta calcolato il valore tutti registri che hanno
64.         # preso parte al suo calcolo devono essere resettati a zero
65.         add $v0, $zero, $zero    # conteneva un numero non più necessario
66.         add $a3, $zero, $zero    # conteneva il segno negativo
67.         add $t2, $zero, $zero    # conteneva numero di condizione di pendenza
68.         add $t3, $zero, $zero    # conteneva numero di condizione di pendenza
69.         add $t8, $zero, $zero    # conteneva un numero usato dal convertitore
70.         j Pcontinue
71.
72.     Pconv: li $a2, 0
73.         jal converter
74.   Pcontinue: addiu $s1, $s1, 1       # aumenta di 1 il contatore di bufferIN
75.         j Ploop
76.
77.   PFINE:
78.     # apri il file OUT (ottenere il descrittore del file)
79.
80.     li $v0, 13            # codice del sistema operativo per aprire file txt
81.     la $a0, OUTp        # nome del file da caricare
```

```
82.     li $a1, 1           # flag per scrittura
83.     li $a2, 0           # flag per modalità
84.     syscall             # chiamata al sistema
85.
86.     # scrivi nel file appena aperto
87.
88.     move $a0, $v0         # descrittore file di OUT
89.     li $v0, 15            # codice syscall per scrittura
90.     la $a1, bufferOUTp    # indirizzo bufferOUT
91.     li $a2, 200           # quanti caratteri da scrivere
92.     syscall
93.
94.     # chiudi il file OUT in a0 è gia presente il descrittore del file OUT
95.     li $v0, 16
96.     syscall
    # RIPRISTINO DALLO STACK DELL'INDIRIZZO DI RITORNO E SALTO AL chiamante
97.     lw $ra, 0($sp)
98.     addi $sp, $sp, 4
99.    jr $ra
```

Il sensore addetto alla pendenza non presenta molte variazioni rispetto allo pseudo-codice ma possiamo notare come si rendono necessari degli azzeramenti dei registri prima dell'entrata nel loop e dopo la scrittura del numero nel buffer di uscita. Inoltre prima del loop vengono impostati i registri $s6 e $s7 (righe 32-33) ai valori dello spazio e del line feeder in codice ASCII esadecimale, che vengono usati per comparare il byte estratto dalla memoria e decidere quale azione compiere.

Sensore sterzo

```
1.  ############# Code Segment ##############
2.  .text
3.  .globl SenSterzo
4.
5.   SenSterzo:
6.
7.      # apri il file degli input (ottenere il descrittore del file in $v0)
8.
9.      li $v0, 13      # codice del sistema operativo per aprire file txt
10.     la $a0, INs     # nome del file da caricare
11.     li $a1, 0       # flag per scrittura
12.     li $a2, 0       # flag per modalità
13.     syscall         # chiamata al sistema
14.
15.     # leggi dal file appena aperto,
```

```
16.
17.     move $a0, $v0     # metti nel parametro di entrata descrittore del file
18.                       # indirizzo del buffer in cui copiare il file di testo
19.     la $a1, bufferIN
20.     li $a2, 500       # specifica la quantità di caratteri nel buffer
21.     li $v0, 14        # codice di chiamata per lettura
22.     syscall           # chiamata al sistema
23.
24.     # chiudi il file IN
25.     move $a0, $v0
26.     li $v0, 16
27.     syscall
28.
29.     li $v0, 0         # azzeramento di v0 in quanto usato successivamente
30.     li $a0, 0         # azzeramento di a0 in quanto usato successivamente contiene s(t)
31.     li $a1, 0         # azzeramento di a1 in quanto usato successivamente contiene s(t-1)
32.
33.     li $s2, 0         # inizializzazione contatore bufferIN
34.     li $s3, 0         # inizializzazione contatore bufferOUT
35.     li $s6, 0x0a      # terminatore bufferIN
36.     li $s7, 0x20      # carica il valore esadecimale 0x20 in s5 (spazio)
37.
38.     # ALLOCAZIONE NELLO STACK DELL'INDIRIZZO DI RITORNO
39.
40.     addi $sp, $sp, -4
41.     sw $ra, 0($sp)
42. # calcola indirizzo in memoria assoluto del buffer in e carica il byte
43.     add $a1, $s5, $s2
44.     lb $a1, ($a1)
45.     blt $a1, 48, Scontinue    # controlla se il byte è un numero
46.     bgt $a1, 57, Scontinue    # deve essere compreso tra 48 e 57
47.     addi $a1, $a1, -48        # converti il char ASCII in intero
48.     addi $s2, $s2, 1          # aggiungi uno al contatore di bufferIN
49.     add $t5, $s5, $s2
50.     lb $t5, ($t5)             # carica byte successivo
51.     beq $t5, $s5, Scontinue   # se il carattere è uguale a uno spazio vai a continue
52.     blt $t5, 48, Scontinue    # controlla se il byte è un numero
53.     bgt $t5, 57, Scontinue    # deve essere compreso tra 48 e 57
54.     addi $t5, $t5, -48        # converti il char ASCII in intero
55.     mul $a1, $a1, 10          # moltiplica per 10 $a1 = $a1 * 10
56.     add $a1, $a1, $t5         # $a1 = $a1 + digit
57.     addi $s2, $s2, 2          # aggiungi due al contatore di bufferIN
58.     add $t5, $zero, $zero     # $t5 = 0 ripulisce variabile
59.
```

```
60.     Sloop:
61.         add $a0, $s5, $s2
62.         lb $a0, ($a0)    # carica  il byte da esaminare
63.         beq $a0, $s6, SFINE       # se questo equivale al terminatore del bufferIN
64.                         # 0x0a allora si passa a FINE
65.
66.         # check se ha trovato uno spazio
67.         bne $a0, $s7, Sconv       # se il byte caricato in s0 non è 20 vai a next
68.
69.         move $a0, $v0       # sposta da v0 dove è stato messo l'ultimo
70.                     # numero calcolato, in a0 e chiama la funzione
71.                     # che implementa la condizione di funzionamento
72.                     # dell sterzo
73.         jal sterzo              # chiamata al modulo condSterzo.s
74.
75.         # conversione in ASCII del risultato e
76.         # scrittura nel bufferOUT
77.         addi $v0, $v0, 48       # ritrasforma in codice ASCII i valori 1 && 0
78.         sb $v0, bufferOUTs($s3)    # metti il valore trovato nella posizione
79.                         # a cui siamo arrivati nel bufferOUT
80.         addiu $s3, $s3, 1       # aumenta di 1 il contatore del bufferOUT
81.         sb $s7, bufferOUTs($s3)    # aggiungi uno spazio, codice 20 in bufOUT
82.         addiu $s3, $s3, 1       # aumenta di 1 il contatore del bufferOUT
83.
84.         # scorrimento dell'ultimo valore calcolato
85.         move $a1, $a0           # conteneva il vecchio numero esaminato
86.         add $a0, $zero, $zero     # imposta a zero $a0
87.         add $v0, $zero, $zero     # conteneva un numero non piu necessario
88.         add $t3, $zero, $zero     # il registro conteneva un numero usato temporaneamente
89.                         # dalla condizione di sterzo
90.         add $t8, $zero, $zero     # conteneva un numero di appoggio per il converter
91.
92.         #calcolo numero
93.         j Scontinue
94.     Sconv: li $a2, 0
95.         jal converter
96.   Scontinue: addiu $s2, $s2, 1       # aumenta di 1 il contatore di bufferIN
97.         j Sloop
98.
99.   SFINE:
100.
101.            # apri il file OUT (ottenere il descrittore del file)
102.
103.            li $v0, 13            # codice del sistema operativo per aprire file txt
```

```
104.            la $a0, OUTs            # nome del file da caricare
105.            li $a1, 1               # flag per scrittura
106.            li $a2, 0               # flag per modalità
107.            syscall                 # chiamata al sistema
108.
109.            # scrivi nel file appena aperto
110.
111.            move $a0, $v0           # descrittore file di OUT
112.            li $v0, 15              # codice syscall per scrittura
113.            la $a1, bufferOUTs      # indirizzo bufferOUT
114.            li $a2, 198             # quanti caratteri da scrivere
115.            syscall
116.
117.            # chiudi il file OUT in a0 è gia presente il descrittore del file OUT
118.            li $v0, 16
119.            syscall
120.
121.          # RIPRISTINO DALLO STACK DELL'INDIRIZZO DI RITORNO E SALTO AL chiamante
122.             lw $ra, 0($sp)
123.             addi $sp, $sp, 4
124.             jr $ra
```

Per verificare il funzionamento del sensore di sterzo viene richiesto di calcolare la differenza tra un valore e il suo precedente, è quindi necessario calcolare il primo numero, in modo tale da avere due valori prima di entrare nel loop. Il loop calcola i valori dal secondo in poi, calcolando per ogni coppia la differenza, se è minore di 10 allora la condizione del sensore deve risultare vera, in caso contrario falsa. Questa funzione imposta i valori in $a0 e $a1 ai valori da sottrarre prima di chiamare la condizione di funzionamento del sensore, una volta calcolato un valore viene messo in $a0 e l'altro viene fatto scorrere da $a0 a $a1 dopo aver scritto nel bufferOUT. Il primo valore viene considerato senza precedente quindi non ne viene calcolata la differenza con nessun numero, infatti il calcolo del primo numero viene fatto usando il registro $a1.

Sensore distanza

```
1.  ############# Code Segment ##############
2.  .text
3.  .globl SenDistanza
4.
5.   SenDistanza:
6.
7.      # apri il file degli input (otteni il descrittore del file in $v0)
8.      li $v0, 13      # codice del sistema operativo per aprire file txt
9.      la $a0, INd     # nome del file da caricare
10.     li $a1, 0       # flag per scrittura
```

```
    li $a2, 0       # flag per modalità
    syscall         # chiamata al sistema

    # leggi dal file appena aperto
    move $a0, $v0    # metti nel parametro di entrata descrittore del file
                              # indirizzo del buffer in cui copiare il file di testo
    la $a1, bufferIN
    li $a2, 500      # specifica la quantità di caratteri nel buffer
    li $v0, 14       # codice di chiamata per lettura
    syscall         # chiamata al sistema

    # chiudi il file IN
    move $a0, $v0
    li $v0, 16
    syscall

    li $v0, 0       # azzeramento di v0 in quanto usato successivamente
    li $a0, 0       # azzeramento di a0
    li $a1, 0       # azzeramento di a1
    li $a2, 0       # azzeramento di a2
    li $a3, 0       # azzeramento di a3
    li $s2, 0       # inizializzazione contatore bufferIN
    li $s3, 0       # inizializzazione contatore bufferOUT
    li $s5, 0x0a     # terminatore bufferIN
    li $s6, 0x20     # carica il valore esadecimale 0x20 in s5 (spazio)

    # ALLOCAZIONE NELLO STACK DELL'INDIRIZZO DI RITORNO

    addi $sp, $sp, -4
    sw $ra, 0($sp)

    Dloop:
        add $a1, $s5, $s2
        lb $a1, ($a1)    # carica  il byte da esaminare
        move $t9, $a1           # carica la lettera nel registro t9 (usata dalla condizione)
        beq $a1, $s5, DFINE       # se questo equivale al terminatore del bufferIN
                            # 0x0a allora si passa a FINE

        addiu $s2, $s2, 1       # aumenta di 1 il contatore di bufferIN

            # check se ha trovato uno spazio
        Dloop2:
            add $a0, $s5, $s2
            lb $a0, ($a0)    # carica  il byte da esaminare
```

```
55.
56.             beq $a0, $s5, DFINE       # se questo equivale al terminatore del bufferIN
57.                                       # 0x0a allora si passa a FINE
58.             bne $a0, $s6, Dconv       # se il byte caricato in s0 non è 20 allora
59.                                       # è un numero hex quindi vai a conv
60.
61.             # sposta i corretti valori da calcolare prima di chiamare la funzione distanza
62.             move $a0, $v0    # per controllare se si è verificata tre volte la stessa sequenza
63.             add $a1, $a1,$a0 # fai la somma lettera valore.
64.
65.             jal distanza            # chiamata al modulo condDistanza.s
66.
67.             # conversione in ASCII del risultato e
68.             # scrittura nel bufferOUT
69.             addi $v0, $v0, 48       # ritrasforma in codice ASCII i valori 1 && 0
70.             sb $v0, bufferOUTd($s3)   # metti il valore trovato nella posizione
71.                                       # a cui siamo arrivati nel bufferOUT
72.             addiu $s3, $s3, 1       # aumenta di 1 il contatore del bufferOUT
73.             sb $s6, bufferOUTd($s3)   # aggiungi uno spazio, codice 20 in bufOUT
74.             addiu $s3, $s3, 1       # aumenta di 1 il contatore del bufferOUT
75.
76.
77.             add $t2, $zero, $zero    # conteneva un numero di cond distanza
78.             add $t3, $zero, $zero    # conteneva un numero di cond distanza
79.             add $t4, $zero, $zero    # conteneva un numero di cond distanza
80.             add $t5, $zero, $zero    # conteneva un numero di cond distanza
81.             add $t6, $zero, $zero    # conteneva un numero di cond distanza
82.             add $t7, $zero, $zero    # conteneva un numero di cond distanza
83.             add $t8, $zero, $zero    # conteneva un numero di intrprt ASCII
84.             j Dcontinue
85.
86.             #calcolo numero
87.         Dconv: li $a3, 0    # spegni flag di segno negativo
88.             li $a2, 1    # accendi flag base esadecimale
89.             jal converter
90.             addiu $s2, $s2, 1       # aumenta di 1 il contatore di bufferIN
91.             j  Dloop2
92.
93.  Dcontinue: addiu $s2, $s2, 1       # aumenta di 1 il contatore di bufferIN
94.          j Dloop
95.  DFINE:
96.
97.     # apri il file OUT (ottenere il descrittore del file)
98.
```

```
99.     li $v0, 13          # codice del sistema operativo per aprire file txt
100.        la $a0, OUTd        # nome del file da caricare
101.        li $a1, 1           # flag per scrittura
102.        li $a2, 0           # flag per modalità
103.        syscall             # chiamata al sistema
104.
105.        # scrivi nel file appena aperto
106.
107.        move $a0, $v0       # descrittore file di OUT
108.        li $v0, 15          # codice syscall per scrittura
109.        la $a1, bufferOUTd      # indirizzo bufferOUT
110.        li $a2, 198         # quanti caratteri da scrivere
111.        syscall
112.
113.        # chiudi il file OUT in a0 è gia presente il descrittore del file OUT
114.
115.        li $v0, 16
116.        syscall
117.
118.      # RIPRISTINO DALLO STACK DELL'INDIRIZZO DI RITORNO E SALTO AL chiamante
119.        lw $ra, 0($sp)
120.        addi $sp, $sp, 4
121.        jr $ra
```

Il formato dell’input di questo sensore ha portato a scrivere due loop, il primo si occupa della lettera che indica la tipologia di ostacolo e viene usata per verificare che non venga misurata la stessa distanza da un ostacolo mobile per più di tre volte; mentre il secondo loop calcola il numero esadecimale che segue la lettera, si noti nella riga 88 che viene attivato il flag per la base esadecimale. Viene usata una operazione di somma per unire il valore della distanza dal tipologia di ostacolo, diminuendo cosi lo spazio necessario a comparare i valori che via via i presentano, infatti il cambio della lettera o di numero creerebbero un valore diverso rendendo diverso il valore negli $a, che rappresentano, il primo $a0, il valore numerico e i restanti $a1, $a2, $a3, la somma lettera+valore.
Prima che sia possibile fare la somma è necessario ottenere il valore numerico, ovvero prima della chiamata alla condizione.
L’ operazione di somma grava meno di un'operazione di lw e sw per questo è stata usata questa strategia.

Condizioni funzionamento sensori

Un’altra funzione viene addetta all’elaborazione del numero convertitore dall’ASCII prendendolo in input e restituendo in output un valore 0 o 1.

Condizione Pendenza

Il sensore di pendenza deve risultare in ogni istante in un intervallo compreso tra -60 e +60. Per dimostrare il funzionamento nel caso di valori sbagliati, questi sono stati inseriti nel file di input, assumendo che questi non potessero superare i -70 e +70 .

La procedura prende in input in $a0 un numero, positivo o negativo, e verifica che appartenga all’intervallo [-60 , +60], se il valore appartiene all’intervallo restituisce in $v0 il valore 1, in caso contrario il valore 0.

```
1.  ############# Code Segment ##############
2.  .text
3.  .globl pendenza
4.
5.          # PARTE DI CONTROLLO SE IL NUMERO CORRISPONDE
6.          # ALL'INTERVALLO +60 -60
7.          # viene elaborato da una funzione esterna
8.          # in a0 viene messo l'input in v0 l'output
9.          pendenza:
10.           li $t0, 60                  # imposta $t0 al limite massimo
11.           li $t1, -60                 # imposta $t1 al limite minimo
12.
13.           slt $t2, $t1, $a0       # imposta a 1 $t2 se $t1 < $v0
14.           slt $t3, $a0, $t0       # imposta a 1 $t3 se $v0 < $t0
15.
16.           and $v0, $t2, $t3       # and logico tra i due registri $t3 & $t4
17.                                       # imposta $t2 a 1 se $t2 < $v1 < $t1
18.
19.           j $ra
```

Nel codice vengono usate 4 variabili temporanee. $t0 e $t1 permettono di settare il limite massimo e minimo mentre $t2 e $t4 contengono due valori di passaggio che permettono di verificare che le due condizioni siano entrambe vere allo stesso tempo, nelle righe 13-14 viene usato il comando **set less than** per impostarli a 1 nel caso il valore sia maggiore del limite minimo e minore del limite massimo. I due risultati sono poi passati tramite l’operatore **and** che restituisce uno solo se entrambi i valori sono uno, ovvero entrambe le condizioni sono vere nello stesso momento. il valore di uscita viene messo in $v0 e si ritorna al programma chiamante.

In & Out sensore pendenza

La sequenza di valori usati in input e restituiti in output sono di seguito, i valori in nero sono gli input, i valori in rosso sottostanti a ogni numero sono il relativo output :

70 -32 -61 73 39 29 -67 40 -74 47 6 -51 -58 56 74 4 -16 3 32 7 59 20 -19 2 22 -29 57 69
0 1 0 0 1 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0
-59 -45 -40 -46 -48 -15 5 72 26 -17 -66 -52 -37 27 66 34 31 10 -63 41 -4 46 30 -72 70
1 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
-62 8 23 14 37 55 45 35 65 -65 16 -54 58 33 -71 71 -36 1 -10 -64 -35 -55 13 36 11
0 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 0 1 1 1 1 1
17 -9 -23 -49 67 62 64 75 53 12 38 -28 -50 -30 -13 -8 -68 15 -20 -25 54 -26
1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1

Condizione Sterzo

Il sensore dello sterzo richiede che la differenza tra un valore in entrata e quello precedente sia minore di 10, in questo caso si ha un output di 1, in caso contrario 0.

La funzione accetta due valori in input, in $a0 il valore appena interpretato, in $a1 il valore precedentemente letto verificando che la differenza tra ogni coppia sia minore di 10:

```
1.  ############# Code Segment ##############
2.  .text
3.  .globl sterzo
4.
5.          # PARTE DI CONTROLLO SE LA DIFFERENZA
6.          # È MINORE DI 10 IN QUEL CASO RESTITUIRE 1 IN V0
7.          # viene elaborato da una funzione esterna
8.          # in a0 viene messo
9.      sterzo:
10.             addi $t0, $zero, 10     # imposta $t0 a 10
11.             addi $t1, $zero, -1     # imposta $t1 a -1
12.             sub $t3, $a0, $a1       # $a0 = $a0 - $a1
13.             bgt $t3, $t1, sfine      # se $a0 > 0 allora fine
14.             nor $t3, $t3, $zero     # sennò inverti i bit del NUMERO
15.             addi $t3, $t3, 1        # e aggiungi uno, per il complemento a due questo
                                          numero è ora positivo
16.  sfine:   slt $v0, $t3, $t0       # imposta $v0 a 1 se $a0 < 10
17.             j $ra
```

Il codice di controllo del sensore inizia con l'allocazione di due variabili 10 in $t0 e -1 in $t1 inoltre viene usata il registro temporaneo $t3 per contenere il valore della differenza.
La differenza, prima di essere esaminata come maggiore/minore di 10, deve essere portata a modulo.

Nella riga 13 il branch controlla se il valore della differenza è negativo, in quel caso provvede a farne il modulo, utilizzando il meccanismo che il complemento a due ci permette(righe 14-15), vengono invertiti i valori dei bit tramite un nor del registro temporaneo con il registro che contiene zero e viene sommato uno al registro temporaneo ottenendo il modulo del numero.
Si procede infine con l'elaborazione dell'output, il registro $v0 viene impostato a uno se il registro $t3 è minore di 10, 0 altrimenti. Infine si ritorna al programma chiamante.

In & Out sensore sterzo

Dato che la specifica prevede di dare un valore di funzionamento ogni due valori in questa rappresentazione dell'input e del output si ha il valore di ritorno, in rosso, tra due valori in entrata, in nero. Il primo valore non ha un precedente, dunque non viene calcolata la differenza con nessun numero

79 85 44 56 28 23 36 61 43 46 33 78 91 34 85 65 19 3 89 70 83 12 32
1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 13 20 9 71 62 8 29 76 88 24 80 4 22 15 27 39 14 41 90 11 82 58
0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
50 99 75 47 54 38 94 66 37 53 31 17 18 67 7 52 42 95 92 10 77 69 74
1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0
63 55 23 49 68 73 45 2 5 84 98 86 9 60 72 51 30 6 26 87 96 16 40
1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
97 21 35 64 59 25 48 57
0 0 0 1 0 0 1

Condizione Distanza

Il sensore di distanza funziona quando il valore numerico esadecimale espresso dal valore di input appartiene all'intervallo [0,50] e che non vengano misurate tre volte di fila distanze uguali da ostacoli mobili, identificati con la lettera B.
Possiamo quindi dividere l'input di questo sensore in due parti, la parte che identifica l'ostacolo che viene processata in un primo loop e viene messa nel registro $a1 e nel registro $t9.
Il primo registro in cui viene messo il byte rappresentante la lettera viene sommato al valore che lo segue una volta calcolato. La somma valore numerico+lettera viene usata per usare un solo registro per contenere i valori chee vengono poi passati dal registro $a1 al registro $s0 per poi andare nel registro $s1, lo "scorrimento" dei registri viene usato per tenere traccia dei due valori precedenti. I registri $s0 e $s1 sono stati usati al posto di $a2 e $a3 per evitare conflitti con la funzione di interpretazione del codice ASCII.
Il secondo registro $t9 contiene solo la lettera necessaria alla verifica della riga 20.
Analizzando il codice possiamo vedere che vengono inizialmente usate due registri $t0 e $t1 come costanti per contenere il valore numerico massimo del sensore e la lettera B in ASCII (righe 6-7), non è necessario usare un registro per contenere il valore numerico minimo esistendo il registro $zero.
Il codice prosegue alla verifica del numero che viene passato in $a0 se è contenuto nei limiti massimo e minimo usando due registri di appoggio, $t2 e $t3 per verificare che le due condizione siano verificate simultaneamente, il registro $t4 viene usato per controllare che siano vere entrambe

(righe11-14). Nel caso il valore in $t4 sia zero, la distanza non rientra nel range e possiamo uscire dalla funzione restituendo zero in $v0 dato che il sensore è già ritenuto non funzionante.

```
1.  ############ Code Segment ##############
2.  .text
3.  .globl distanza
4.        distanza:
5.
6.          li $t0, 50            # imposta $t1 a 50 decimale
7.          li $t1, 0x42           # valore di B in codifica ASCII esadecimale
8.
9.          # controllo se valore di distanza in $a1 è compreso tra
10.         # 0 e 50
11.         slt $t2, $zero, $a0      # imposta a 1 $t3 se 0 < $a0
12.         slt $t3, $a0, $t0       # imposta a 1 $t4 se $a0 < $t0
13.
14.         and $t4, $t2, $t3       # and logico tra i due registri $t2 & $t3
15.         beq $t4, $zero dfine      # imposta $t4 a 1 se 0 < $a0 < 50
16.
17.         # controllo misure precedenti
18.         add $t7, $zero, 1       # supponi che non i tre valori siano diversi
19.
20.         bne $t9, $t1, dfine       # se la lettera corrente è diversa da B
21.                                    # puoi saltare a fine
22.    bne $a1, $s0, dfine   # se la lettera in input non è una B allora vai a fine
23.         li  $t5, 0                  # sennò carica in $t5 il valore 1
24.    bne $a1, $s1, dfine # se la lettera precedente non era una B allora vai fine
25.         li  $t6, 0                  # sennò carica in $t6 il valore 1
26.         and $t7, $t5, $t6       # verifica che non sia stato letto tre volte
27.                              # t5 e t6 sono rispettivamente la seconda e terza volta
28.     dfine:move $s1, $s0
29.         move $s0, $a1
30.
31.   and $v0, $t4, $t7    # metti il risultato 1 in v0 se entrambe le condizioni sono
      rispettare
32.
33. j $ra
```

In caso si ottenga un 1 in $t4 possiamo procedere verificando che non si siano presentati tre volte lo stesso valore per un sensore mobile. Il codice suppone vera la condizione che non si siano presentati tre volte lo stesso valore fino a prova contraria(riga 18). Dato che la condizione di malfunzionamento specifica che si deve tenere conto della tipologia di ostacolo dobbiamo valutare che tipo di ostacolo abbiamo in input, viene usato il registro temporaneo in $t9 che contiene la lettera corrente equiparato al registro $t1, contenente la lettera B (riga 20). Se i due registri sono diversi abbiamo un ostacolo fisso che non presenta problemi e possiamo passare alla fine, in caso contrario dobbiamo verificare se il valore+lettera che stiamo esaminando non sia uguale a coppie valore+lettera avute precedentemente (righe 22-26), la somma valore+lettera qui risulta utile per distinguere i casi in cui si ha lo stesso valore numerico ma diversa lettera , senza la somma non si terrebbe traccia di che lettera sia passata, si consideri l'esempio:

B20 A20 B20

In questo caso usando la somma si avrebbe la sequenza corretta 111, ma senza la somma non si potrebbe tenere traccia del fatto che nella seconda coppia lettera.valore la lettera è una A, valutando solo il valore numerico si avrebbe la sequenza errata 110 avendo letto la terza lettera come una B e tre volte lo stesso valore. La somma quindi risulta una semplice ed efficace soluzione per memorizzare valori e lettere nello stesso registro.

Infine la funzione provvede a scorrere i registri e verificare che entrambe le condizioni siano verificate mettendo il risultato in $v0 e ritornando alla funzione chiamante.

In & Out sensore distanza

A2D B00 A32 A1A B27 B27 B27 A02 A2F AE2 B72 A3D AC1 A3A BE3 B1A B8E B1C B20 BBC

1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 1 0 1 1 0

A01 BB0 BF8 AFF B16 AAB AEE AAA BE9 A0D AB4 A8A BCB A12 AFC A42 BB5 B76 BC0

1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0

A65 B5C ACF BEC BF6 A11 B6A B56 BFB B63 B60 A82 B51 AD2 B08 A03 BC0 A97 BC1 B17

0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1

BD4 B6B AF4 AF4 AAC AD8 AF1 A82 B73 A65 B49 AB9 B72 BAF BCB AF4 B32 A38 AFB

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

B30 BFA B83 BD2 AE0 A41 A70 AE5 B9E BA2 B5A B85 BCF BB1 BB2 A1C A1C A62 ACB

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0

BC5 B5F A92

0 0 0

Politiche di correttezza

Le politiche di correttezza sono usate per aggregare i risultati di ogni sensore secondo tre modalità. Dalla specifica del progetto si legge:

> "Ad ogni istante t, una volta che dei valori di correttezza corrp(t), corrs(t), corrd(t) sono stati stabiliti per ciascuno dei sensori, si deve stabilire il valore di correttezza dell'intero sistema corrsis(t) aggregando tali indici singoli secondo le seguenti politiche:
> P1. Il sistema viene reputato funzionante all'istante t (corrsis(t)=1) se tutti e 3 i sensori sono ritenuti funzionanti allo stesso istante (ovvero corrp(t)=1 AND corrs(t)=1 AND corrd(t)=1). Altrimenti il sistema viene reputato come malfunzionante (corrsis(t)=0).
> P2. Il sistema viene reputato funzionante all'istante t (corrsis(t)=1) se almeno 2 sensori sono ritenuti funzionanti allo stesso istante. Altrimenti il sistema viene reputato come malfunzionante (corrsis(t)=0).
> P3. Il sistema viene reputato funzionante all'istante t (corrsis(t)=1) se almeno 1 sensore è ritenuto funzionante allo stesso istante. Altrimenti il sistema viene reputato come malfunzionante (corrsis(t)=0)"

-pag 2 della specifica del progetto

Sono quindi richieste tre procedure che collegano i valori risultati dai sensori nelle tre modalità richieste.

Le procedure delle politiche si concludono con la scrittura nel file di testo output e dopo aver applicato la terza politica viene chiuso il simulatore SPIM.

Pseudo-codice

La logica di ogni politica è identica se non per il modo con cui vengono aggregati i vari valori dei buffer di uscita. Appena tutte le funzioni dei sensori sono state chiamate si passa al loop addetto al caricamento dei caratteri ASCII dai buffer di uscita dei sensori in registri temporanei i cui valori devono essere riportati a numeri. Ottenuti i valori numerici possiamo verificarli secondo quanto richiesto dalla politica. Il codice esce dal loop quando uno dei tre bit è uno zero, ovvero i valori dei buffer sono finiti, per questo fine il contorno separatore si rende utile. Una volta che i buffer in uscita delle politiche sono stati scritti si passa a copiarne il contenuto nei file di testo in uscita della politica. Di seguito uno pseudo-codice:

```
ciclo :
        carica  i byte dai bufferOUT dei sensori
                se viene caricato zero si esce dal ciclo
        interpreta byte da ASCII a numero
        politica di aggregazione risultati dei sensori

        scrivi in bufferOUT

apri file output
scrivi in file output, carica da bufferOUT
chiudi file output
metti in $v0 l'indirizzo del bufferOUT
salto al chiamante
```

Codice politiche

Dato la somiglianza dei codici per le varie politiche viene riportato l'intero codice solo per la prima politica, mentre per le altre politiche possiamo limitarci alla discussione della parte di codice in cui viene implementato il controllo dei tre valori dei sensori

Politica 1

```
1.  #  ESEGUI PRIMA POLITICA
2.   # POLITICA DI AGGREGAZIONE E SCRITTURA IN FILE DI TESTO OUT
3.   # carica byte in $t0, $t1, $t2
4.   la $s2, bufferOUTd      # indirizzo bufferOUT DISTANZA
5.   la $s1, bufferOUTs      # indirizzo bufferOUT STERZO
6.   la $s0, bufferOUTp      # indirizzo bufferOUT PENDENZA
7.
8.   li $s3, 0           # inizializzazione del contatore del bufferIN
9.   li $s4, 0           # inizializzazione del contatore del bufferOUT
10.  li $s5, 0x20          # codice ASCII per SPAZIO
11.
12. P1loop:
13.
14.  lb $t0, ($s0)           # carica byte dal buffer pendenza
15.  beq $t0, $zero, P1scrivi     # se il byte estratto è zero allora vai a fine
16.                  # funziona perchè i numeri sono in codice ascii nei buffers
17.                  # e questi terminano con zero
18.  addi $s0, $s0, 2         # incremento di 2 dell'indirizzo base cosi da
19.                  # non considerare lo spazio
20.
21.  lb $t1, ($s1)           # carica byte dal buffer sterzo
22.  beq $t1, $zero, P1scrivi     # se il byte estratto è zero allora vai a fine
23.  addi $s1, $s1, 2         # incremento di 2 dell'indirizzo base cosi da
24.                  # non considerare lo spazio
25.
26.  lb $t2, ($s2)           # carica byte dal buffer distanza
27.  beq $t2, $zero, P1scrivi     # se il byte estratto è zero allora vai a fine
28.  addi $s2, $s2, 2         # incremento di 2 dell'indirizzo base cosi da
29.                  # non considerare lo spazio
30.
31.  addi $t0, $t0, -48     # riporta in decimale i caratteri letti nel buffer (0 o 1)
32.  addi $t1, $t1, -48     # riporta in decimale i caratteri letti nel buffer (0 o 1)
33.  addi $t2, $t2, -48     # riporta in decimale i caratteri letti nel buffer (0 o 1)
34.
35.    # controllo se 3 sensori sono funzionanti
36.    and $t0, $t0, $t1      # controllo se $t0 e $t1 funzionano
37.    and $t0, $t0, $t2      # controllo se anche $t2 funziona
38.
39.  # scrittura nel bufferOUT
40.  addi $t0, $t0, 48       # ritrasforma in codice ASCII i valori 1 && 0
41.  sb $t0, bufferOUT1($s4)   # metti il valore trovato nella posizione
42.                  # a cui siamo arrivati nel bufferOUT
```

```
43. addiu $s4, $s4, 1       # aumenta di 1 il contatore del bufferOUT
44. sb $s5, bufferOUT1($s4)   # aggiungi uno spazio, codice 20 in bufOUT
45. addiu $s4, $s4, 1       # aumenta di 1 il contatore del bufferOUT
46.
47.
48. add $t0, $zero, $zero    # conteneva un numero di cond
49. add $t1, $zero, $zero    # conteneva un numero di cond
50. add $t2, $zero, $zero    # conteneva un numero di cond
51.
52. #incremento contatore
53. addi $s3, $s3, 2      # incremento contatore di due, salta lo spazio tra i valori
54.
55. j P1loop
56.
57. P1scrivi:
58. # apri il file OUT (ottenere il descrittore del file)
59. li $v0, 13            # codice del sistema operativo per aprire file txt
60. la $a0, OUT1           # nome del file da caricare
61. li $a1, 1             # flag per scrittura
62. li $a2, 0             # flag per modalità
63. syscall               # chiamata al sistema
64.
65. # scrivi nel file appena aperto
66. move $a0, $v0          # descrittore file di OUT
67. li $v0, 15            # codice syscall per scrittura
68. la $a1, bufferOUT1       # indirizzo bufferOUT
69. li $a2, 200           # quanti caratteri da scrivere
70. syscall
71.
72. # chiudi il file OUT in a0 è gia presente il descrittore del file OUT
73.
74. li $v0, 16
75. syscall
```

Possiamo notare che gli indirizzi vengono gestiti in modo particolare in questo caso, viene caricato l'indirizzo base del buffer in tre registri $s0, $s1, $s2, ai registri viene poi sommato l'incremento. Ottenuti gli indirizzi possiamo passare a estrarre i byte. Gli indirizzi vengono incrementati dopo aver caricato il byte di due, in modo tale da indirizzare il prossimo valore non considerando lo spazio.
I byte estratti vengono convertiti da ASCII a numero e valutati (righe 31-33), in questo caso si ha che tutti e tre i sensori devono essere funzionanti nello stesso momento per rendere funzionante il sistema, per ottenere questo risultato i tre valori vengono aggregati con l'operatore **and**.

In e Out della politica 1

I risultati ottenuti nei buffer di uscita delle funzioni dei sensori vengono rappresentati in verde per il sensore di pendenza, in blu il sensore di sterzo e in nero il sensore di distanza. In rosso i risultati della politica, in questo caso abbiamo bisogno di tutti e tre i sensori funzionanti per poter dichiarare il sistema funzionante.

0 1 0 0 1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0
1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1
0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1
1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Politica 2

```
1.  # controllo se almeno 2 sensori sono funzionanti
2.  # ps+pd+sd
3.
4.          and $t3, $t0, $t1   # usa tre registri temporanei per mantenere il risultato
5.          and $t4, $t0, $t2   # degli and tra ps, pd, sd
6.          and $t5, $t1, $t2   #
7.          or $t6, $t3, $t4    # il risultato del primo or tra ps e pd viene messo in t6
8.          or $t0, $t6, $t5    # e il risultato finale tra (ps+pd) or sd in t0
```

La seconda politica di aggregazione è la più complessa in quanto richiede che almeno due sensori in un dato momento siano attivi per poter dichiarare il sistema funzionante. Per ottenere questo risultato è necessario possiamo passare da una tabella di verità :

pendenza	sterzo	distanza	2 sensori
0	0	0	0
0	0	1	0
0	1	0	0
0	*1*	*1*	*1*
1	0	0	0
1	*0*	*1*	*1*
1	*1*	*0*	*1*
1	*1*	*1*	*1*

Estraendo la prima forma canonica dalla tabella otteniamo:

p’sd+ps’d+psd’+psd =

Manipolando l’espressione possiamo minimizzarla:

(p’sd+psd)+(ps’d+psd)+(psd’+psd) =
(p’+p)(sd)+(s’+s)(pd)+(d’+d)(sp) =
ps+pd+sd

Sono quindi necessari tre registri temporanei $t3, $t4, $t5, che contengono i risultati degli **and** tra ps, pd, sd, (righe 4-6) questi risultati vengono poi uniti tramite degli **or** utilizzando un ulteriore registro di appoggio, $t6 infine il risultato viene messo in $t0 (righe 7-8).

In e Out della politica 2

I risultati ottenuti nei buffer di uscita delle funzioni dei sensori vengono rappresentati in verde per il sensore di pendenza, in blu il sensore di sterzo e in nero il sensore di distanza. In rosso i risultati della politica, in questo caso abbiamo bisogno di almeno due sensori funzionanti per poter dichiarare il sistema funzionante.

Politica 3

```
1.          # controllo se almeno uno dei 3 sensori sono funzionanti
2.      or $t0, $t0, $t1    # controllo se $t0 o $t1 funzionano
3.      or $t0, $t0, $t2    # controllo se $t2 funziona
```

La terza politica di aggregazione permette di dichiarare il sistema funzionante quando almeno uno dei tre sensori è funzionante in un dato momento. Per implementare questa funzione vengono usati due operatori **or.**

In e Out della politica 3

I risultati ottenuti nei buffer di uscita delle funzioni dei sensori vengono rappresentati in verde per il sensore di pendenza, in blu il sensore di sterzo e in nero il sensore di distanza. In rosso i risultati della politica, in questo caso abbiamo bisogno di almeno uno dei sensori funzionanti per poter dichiarare il sistema funzionante.

Conclusioni

Segmento testo

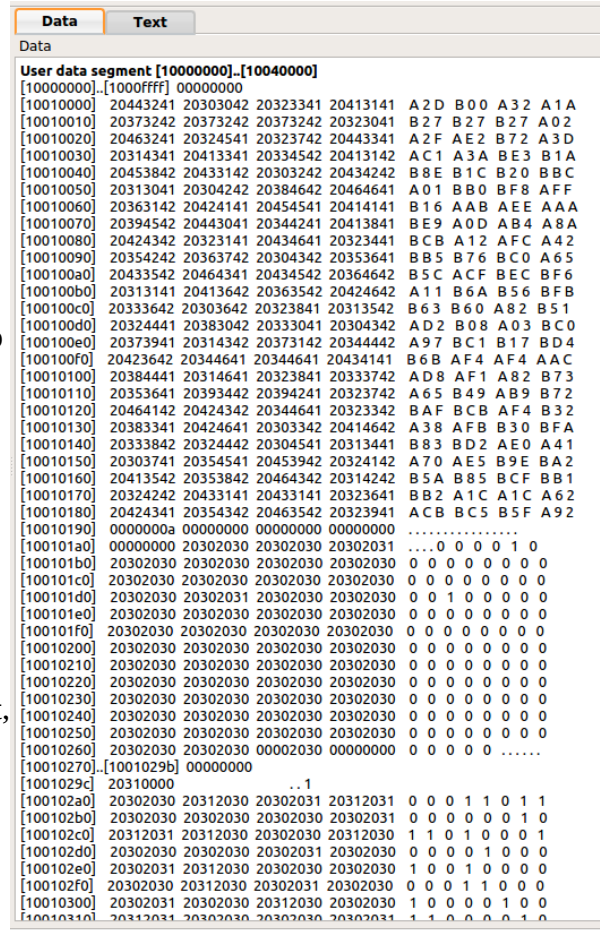
Lo spazio occupato dal segmento di testo contenente le istruzioni del programma può essere calcolato tramite gli indirizzi che QtSpim ci fornisce. Quando viene caricato il programma su QtSpim, questo numera ogni istruzione partendo dal valore esadecimale 0x400014, le prime righe sono necessarie al compilatore, quel valore indica il primo byte contenente l'istruzione, dato che MIPS usa l'indirizzamento al byte per le istruzioni.
Una volta caricato il programma, scorrendo fino alla fine della finestra che mostra il segmento di testo arriviamo all'ultimo indirizzo in cui viene posta l'ultima istruzione del programma. La numerazione in QtSpim è differente dai programmi di testo con cui viene scritto il programma in quanto QtSpim elabora le eventuali pseudoistruzioni trasformandole in istruzioni native di MIPS. In questo caso quindi si ha il valore esadecimale 0x400848 che equivale al valore decimale 4196424 che sottratto al valore iniziale 0x400014 (in decimale 4194324) si ottiene 2100. Questo valore viene moltiplicato per 4 in quanto le istruzioni MIPS sono sempre lunghe 32 bit o 4 byte. La moltiplicazione per 4 permette di passare da numero di istruzioni(word) a byte.
0x400848 – 0x400014 =
4196424 – 4194324 = 2100 istruzioni * 4 = 8400 byte
Questo valore, 8400 byte, è lo spazio necessario per contenere le istruzioni del programma.

Segmento dati

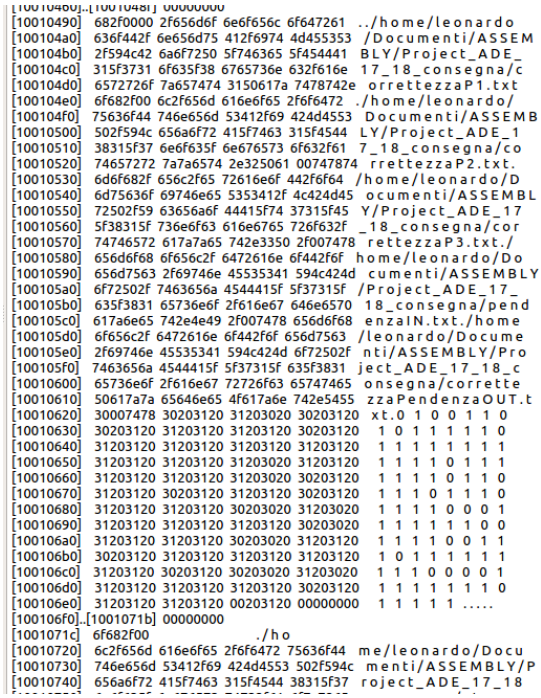
Il segmento dati viene riempito con gli indirizzi ai file da leggere e scrivere e i buffers contenenti i valori calcolati, di seguito viene mostrato uno screenshot in cui viene mostrato come QtSpim mette i dati nel segmento.

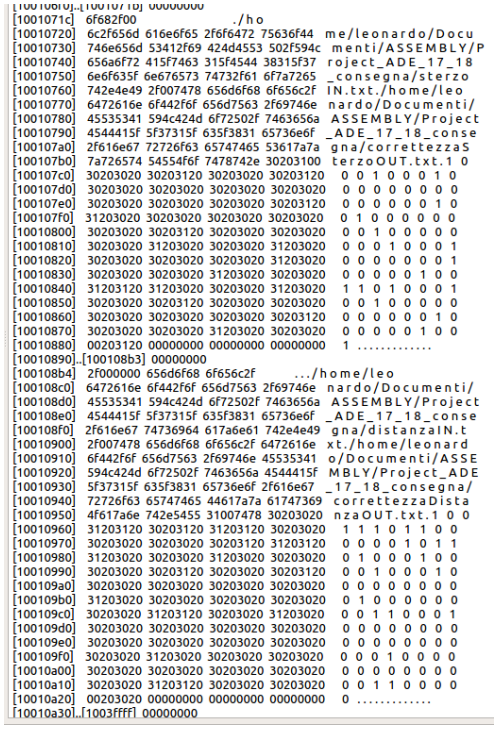
Nello screenshot di fianco possiamo notare come lo spazio del bufferIN riservato dal file della politica, viene riscritto da ogni sensore (nell'immagine possiamo vedere che sono presenti solo i dati del sensore di distanza, l'ultimo che viene chiamato), questo riduce l'uso della memoria per quanto riguarda i valori di input del sensore.

Al di sotto del bufferIN possiamo vedere i bufferOUT delle politica. Negli altri due screenshot, nella pagina seguente, mostrano il resto del segmento dati, seguono l'indirizzo del file di input



e di output di ogni sensore con il relativo buffer di uscita.





Per calcolare lo spazio necessario a contenere tutti i dati nel segmento dati, possiamo sommare i valori che abbiamo impostato per ogni buffer e indirizzo.

I buffer sono in totale 7:

- bufferIN, viene dichiarato dalla politica, è condiviso dai sensori, è grande 420 byte;
- bufferOUT1, contiene i dati output della politica 1 è grande 250 byte;
- bufferOUT2, contiene i dati output della politica 2 è grande 250 byte;
- bufferOUT3, contiene i dati output della politica 3 è grande 250 byte;
- bufferOUTp, contiene i dati output del sensore di pendenza, è grande 250 byte;
- bufferOUTs, contiene i dati output del sensore di sterzo, è grande 250 byte;
- bufferOUTd, contiene i dati output del sensore di distanza, è grande 250 byte;

In totale i buffer necessitano di 1920 byte per poter contenere i dati processati da una politica.

Gli indirizzi invece vengono scritti come percorsi completi scritti in ASCII e terminati con un line feeder. Gli indirizzi sono 9 in totale, 2 per ogni sensore (per IN e OUT) e 3 per i file di output della politica. Purtroppo non è possibile usare indirizzi relativi e quindi siamo costretti a usare i più lunghi indirizzi assoluti che devono essere cambiati una volta che il codice viene eseguito in un altro sistema. Ma sapendo esattamente dove si trovano i file da eseguire possiamo scrivere il percorso e indirizzare il file interessato, i primi 61 caratteri degli indirizzi sono uguali per ogni indirizzo */home/leonardo/Documenti/ASSEMBLY/Project_ADE_17_18_consegna/* poi viene scritto

il nome del file che ha una lunghezza massima di 22 caratteri, per finire abbiamo l'estensione del file, altri 4 caratteri, e il carattere del line feeder. In totale quindi per ogni indirizzo ai file di testo vengono usati 88 caratteri ASCII, ognuno di questi occupa un byte. Si hanno 88 byte per indirizzo e 9 indirizzi che in totale occupano 792 byte.

Stack

Durante l'esecuzione del codice lo stack viene usato in modo molto limitato. I processi che gestiscono i sensori usano una parola all'inizio della loro spezzone di codice e la elimina dallo stack prima di tornare al chiamante. La capacità massima necessaria dello stack per questo eseguibile è di 1 parola, in quanto vengono caricati dei registri di ritorno, uno alla volta. Una parola equivale a 4 byte.

Possiamo quindi definire lo spazio in memoria necessario a contenere i dati del segmento dati, le istruzioni del codice e lo stack, sommando tutti i valori ottenuti finora otteniamo :

8400 + 1920 + 792 + 4 = 11116 byte

Tempo complessivo di esecuzione dell'intero programma

Leggendo il codice è possibile notare che le istruzioni utilizzate nel codice appartengono tutte a un piccolo elenco, il numero associato è il numero di volte che l'istruzione appare nel codice e questa è stata calcolata tramite l'aiuto delle funzioni di ricerca dell'editor di testo usato per scrivere il codice. Le istruzioni sono state raggruppate secondo il criterio della specifica:

> Determinare il tempo complessivo di esecuzione dell'intero programma, considerando la somma dei tempi di esecuzione delle singole istruzioni con le seguenti assunzioni:
>
> Ciascuna istruzione di lettura dati da memoria (lw, lh, lb) viene eseguita in 800ps (800x10-12sec);
>
> Ciascuna istruzione di scrittura dati su memoria (sw, sh, sb) viene eseguita in 700ps (700x10-12sec);
>
> Ciascuna delle istruzioni rimanenti viene eseguita in 500ps (500x10-12sec);
>
> Escludere dal calcolo il tempo necessario per leggere/scrivere da/su file (syscall per la read/write).
>
> *-pag 4 della specifica del progetto*

Contando le istruzioni quindi non si sono considerate le syscall.

- istruzioni di load, totale 139:
  - lw, 3
  - lb, 15
  - la, 27(l'assemblatore traduce questa pseudo-istruzione nella coppia di istruzioni **lui** e **ori,** quindi viene contata come istruzione di load)
- istruzioni di store, totale 15:
  - sw, 3
  - sb, 12

- altre istruzioni, totale 271:
  - li, 94 (l'assemblatore traduce questa pseudo-istruzione nell'istruzione **ori**)
  - add e istruzioni derivate (addi, addiu), 158
  - move, 20
  - sub, 4
  - mul, 2
  - jal e j, 32
  - jr, 4
  - slt, 5
  - blt, 3
  - bgt, 4
  - beq, 17
  - bne, 7
  - and, 9
  - or, 5

Per ottenere il totale delle istruzioni è necessario sommare l'unica istruzione che non è stata contata, la syscall, nel codice questa istruzione viene chiamata 25 volte. Otteniamo 139 + 15 + 271 + 25 = 450 istruzioni totali di cui 139, le istruzioni di load necessitano di $800*10^{-12}$ s per essere eseguite, 15, le istruzioni di store sono risolte in $700*10^{-12}$ s e il resto delle istruzioni hanno bisogno di $500*10^{-12}$ s. Abbiamo quindi che:

$139 * 800*10^{-12}\,s + 15 * 700*10^{-12}\,s + 271 * 500*10^{-12}\,s = 2.572*10^{-7}\,s$

Il calcolo del tempo necessario all'esecuzione viene calcolato in modo approssimativo a causa dell'uso di pseudo-istruzioni usate nel codice, queste vengono tradotte dall'assemblatore in una serie di istruzioni proprie dell'ISA.

Discussione sull'ottimizzazione del codice

Data la modalità con cui viene calcolato il tempo di esecuzione, la somma dei tempi di ogni istruzione, la minimizzazione del codice usato è una forma di ottimizzazione. Il codice ha alcuni punti in cui potrebbe essere minimizzato in quanto presenta pezzi di codice ripetuti. Ad esempio l'azzeramento dei registri dopo la chiamata al processo dei sensori, potrebbe essere raccolto in una funzione opportunamente chiamata al momento in cui nessun valore del registro è più necessario, questa soluzione pur rendendo più compatto e leggibile il codice, non permette di salvare eventualmente dei valori se non caricandoli sullo stack. Esistono poi le scritture dei buffer nei file di testo di ogni sensore, anche qui sarebbe possibile scrivere un processo che prende in input l'indirizzo del file di testo da scrivere e l'indirizzo del buffer da copiare. Queste operazioni diminuirebbero leggermente il numero delle istruzioni. Per l'ottimizzazione del codice tramite la pipeline ci si affida completamente all'assemblatore.

Con gli accorgimenti discussi sopra si riuscirebbe gia ad avere degli sconti in termini di memoria in quanto il numero delle istruzioni si abbasserebbe, di conseguenza lo spazio richiesto dal segmento testo diminuirebbe.

Per usare meno memoria è possibile anche usare un ragionamento simile a quello usato per il bufferIN per i buffer delle politiche. Invece di usare tre buffer delle politiche sarebbe possibile usarne uno solo che viene riscritto da ogni politica dopo che il precedente ha riempito e copiato il buffer nel suo file di output. Nel codice, per via del minimo uso di risorse, non viene implementata questa soluzione, è stato scelto invece di usar tre porzioni di memoria che porta però ad avere tutti i risultati ottenuti nel segmento dati in modo chiaro. In questo caso è possibile sapere quanta memoria viene risparmiata in quanto sappiamo che la soluzione porta all'eliminazione di due buffer da 250 byte ciascuno, ottenendo 500 byte in meno richiesti dal codice per funzionare.

Altre migliorie che possono essere apportate al codice sono ad esempio l'uso di una jump table per implementare il costrutto case-switch che permette di innescare una serie di azioni a secondo dell'input ricevuto, nel nostro caso sarebbe utile per creare una versione più sofisticata del processo che permette di interpretare un byte in ASCII. Nel caso di questo esercizio non è stata usata la jump table in quanto avrebbe reso più complicata la stesura del codice, la cui logica è abbastanza elementare da non richiedere miglioramenti in termini semplicità del codice.