Marco Casaglia 5759711 marco.casaglia@stud.unifi.it
03/07/2018

Monitoraggio Sensori in Smart Vehicles: Progetto MIPS A.A. 2017/2018

Il progetto consiste nello sviluppo di un algoritmo implementato in codice assembly MIPS di una unità di monitoraggio che determini la correttezza dei sensori di una automobile. Al tempo t l'unità di monitoraggio riceve da tre sensori un valore, che deve essere verificato al fine di determinare il corretto funzionamento del sensore stesso. Inoltre dev'essere determinato il corretto funzionamento dei sensori nel complesso secondo le seguenti politiche di aggregazione:

- P3(t) : Almeno uno dei tre sensori funziona correttamente al tempo t.
- P2(t) : Almeno due dei tre sensori funziona correttamente al tempo t.
- P1(t) : Tutti i sensori funzionano correttamente al tempo t.

Sia l'input fornito dai tre sensori che i sei output, che indicano il corretto funzionamento per ognuno dei tre sensori e il corretto funzionamento del sistema secondo le varie politiche di aggregazione, sono file di testo che devono trovarsi nella stessa cartella dell'eseguibile dell'assemblatore. Le operazioni dovranno eseguite per ogni t con (t) totale uguale 100.

Descrizione della soluzione adottata

L'unità di monitoraggio per prima cosa riceve i 100 input da ognuno dei 3 sensori e li carica su 3 diversi buffer, il cui indirizzo viene caricato in tre diversi registri. Carica inoltre in altri 6 registri i buffer di uscita dei 6 output. A quel punto inizia un ciclo che si ripete 100 volte, per il quale il registro $t0 agisce da contatore, in cui 4 procedure diverse vengono chiamate, e per le quali i registri temporanei vengono salvati nello stack, in modo che non siano modificati dalle prime 3 procedure, mentre le quarta procedura li utilizza, perciò ha accesso ai valori da modificare. Le prime tre procedure sono rispettivamente la procedura che restituisce la correttezza del sensore di pendenza, quella che restituisce la correttezza del sensore dello sterzo e quella che restituisce la correttezza del sensore di distanza da ostacoli. Al ritorno il risultato di ognuna delle procedure viene caricato in tre registri che vengono passati alla quarta procedura che si trova nel ciclo. Questa procedura, pone nei buffer di uscita appropriati i risultati della valutazione dei sensori, inoltre valuta il sistema secondo le 3 politiche di aggregazione e salva i risultati nei tre buffer appropriati. Questo viene ripetuto fino a che $t0 non raggiunge il valore 100 nel ciclo principale, a quel punto i buffer di uscita vengono utilizzati nel chiamare la procedura che li stampa nel file .txt che li pertiene.

Ecco le procedure nel dettaglio:

-Correttezza sensore di pendenza| Il sensore di pendenza non deve dare risultati per cui $|p(t)|<60$
perciò, per prima cosa in caso il valore sia negativo, il segno "-" viene ignorato. Dopodiché un ciclo decodifica i caratteri ASCII ottenuti dal buffer, assumendo un numero in base 10, perciò ad ogni ciclo il risultato viene moltiplicato per 10 e sommato al nuovo carattere letto, che per essere trasformato da ASCII a intero riceve una sottrazione di 48 unità. Adesso che è stato ottenuto un intero senza segno, è sufficiente verificare che sia minore di 60 perché la correttezza del sensore sia verificata.

-Correttezza sensore di sterzo| Il sensore di sterzo non deve dare risultati per cui $|s(t) – s(t -1)|>10$
perciò, come prima entro in un ciclo che decodifica i caratteri ASCII e restituisce il numero intero. A questo punto viene caricato dalla memoria il valore s(t-1) viene fatta la sottrazione con s(t) e il risultato, dopo aver ottenuto il valore assoluto della sottrazione viene verificato, in caso sia maggiore di 10 il sensore viene considerato non funzionante. In ogni caso prima di tornare al ciclo principale, s(t) viene salvato in memoria per essere utilizzato nella verifica di s(t+1).

-Correttezza del sensore di distanza da ostacoli| Il sensore di distanza da ostacoli per funzionare correttamente deve rispettare 3 direttive. La distanza non dev'essere mai zero oppure superiore a 50, inoltre in caso si tratti di un ostacolo mobile, non deve trovarsi alla stessa distanza per tre o più rilevazioni di seguito. Per prima cosa viene verificato se si tratta o meno di un ostacolo mobile. In caso non lo sia, viene

posta una flag interna alla procedura, inoltre viene resettata la flag che indica che due ostacoli mobili sono stati rilevati alla stessa distanza precedentemente. In seguito, c'è un ciclo che decodifica i caratteri ASCII, analogo a quelli precedenti, ma differente poiché i sensori forniscono dati in codifica esadecimale, per cui la moltiplicazione per 10 viene sostituita da una moltiplicazione per 16, inoltre dato che la codifica dei caratteri dell'alfabeto maiuscoli si trova 7 posizioni più in altro rispetto ad i numeri nella tabella ASCII, in caso si trovi uno di questi caratteri, viene effettuata una correzione sottraendo 7 a tali numeri. A quel punto la correttezza viene verificata in caso che il valore ottenuto sia diverso da zero, non maggiore di 50 e che, controllata la flag interna alla procedura ($t4), si tratti di un ostacolo immobile. Se invece si tratta di un ostacolo mobile, si carica dalla memoria il valore di distanza del precedente ostacolo mobile (zero in caso il precedente ostacolo fosse un ostacolo fisso), si salva il corrente valore di distanza in memoria e si fa un confronto tra i due valori. In caso sia lo stesso, si controlla la flag caricata dalla memoria. Se la flag è uguale a 1, i precedenti valori erano uguali, perciò questa sarebbe la terza istanza e la procedura porterebbe esito negativo. In caso la flag sia negativa invece viene caricato in memoria per settarla come positiva, e il risultato della procedura sarebbe positivo. Se i due valori di distanza sono diversi invece, la flag viene resettata a zero e la procedura ha esito positivo.

Utilizzo della memoria

Viene utilizzato lo stack per il salvataggio dei registri utili nel ciclo principale. Per i vari buffer e per la conservazione di flag in memoria è utilizzata memoria statica.
Il buffer buffer1_in, ha dimensione 1200 byte, per poter accomodare 100 misure di dimensione 12 caratteri. Infatti, maxInt32 sia unsigned che signed, è lungo 10 caratteri, a quei 10 dobbiamo aggiungere un eventuale segno '-' per numeri negativi, e ogni volta uno spazio che separa le misure.
Gli altri due buffer, buffer2_in e buffer3_in, hanno una formattazione più limitata, infatti ogni misura non può superare i 4 caratteri, perciò 400 byte sono sufficienti.
I buffer di uscita sono tutti formattati allo stesso modo. Un numero '1' oppure '0' seguito da uno spazio, perciò bastano 200 byte.

Motivazione delle scelte implementative

Il sistema del cycle_scheduler è stato implementato per simulare una valutazione dei vari sensori allo stesso tempo t, nonostante gli input da file siano immediati e contemporanei. I buffer di uscita hanno dimensione prestabilita e consociuta dalla formattazione del testo, perciò l'utilizzo della memoria dinamica sarebbe stato non necessario. I buffer di entrata potrebbero avere dimensione variabile, ma utlizzare una dimensione fissa mi permette di utilizzare un'unica procedura per la lettura di file, senza dover cambiare la dimensione del buffer letto.

Dimostrazione di corretto funzionamento

Le prime tre righe della tabella indicano gli input, le seguenti sono i tre output di correttezza dei sensori seguiti dai tre output P1,P2,P3.

Allegati nella cartella si trovano tutti e nove i file .txt di cui questa tabella rappresenta un estratto.

p(t)	-16	23	-21	25	-60	35	12	26	80
s(t)	50*	57	71	65	46	12	3	15	24
d(t)	A21	B21	B21	B21	B00	A31	A00	A33	B31
$corr_p(t)$	1	1	1	1	0	1	1	1	0
$corr_s(t)$	0	1	0	1	0	0	1	0	1
$corr_d(t)$	1	1	1	0	0	1	0	0	1
P1(t)	0	1	0	0	0	0	0	0	0
P2(t)	1	1	1	1	0	1	1	0	1
P3(t)	1	1	1	1	0	1	1	1	1

*s(t-1)=39

Discussione sui requisiti di memoria e sui tempi di esecuzione

First memory address:0x10010000

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x20656854	0x656c6966	0x73617720	0x746f6e20	0x756f6620	0x203a646e	0x646e6570	0x617a6e65
0x10010020	0x742e4e49	0x73007478	0x7a726574	0x2e4e496f	0x00747874	0x74736964	0x617a6e61	0x742e4e49
0x10010040	0x63007478	0x6572726f	0x7a657474	0x6550617a	0x6e65646e	0x554f617a	0x78742e54	0x6f630074
0x10010060	0x74657272	0x7a7a6574	0x65745361	0x4f6f7a72	0x742e5455	0x63007478	0x6572726f	0x7a657474
0x10010080	0x6944617a	0x6e617473	0x554f617a	0x78742e54	0x6f630074	0x74657272	0x7a7a6574	0x2e315061
0x100100a0	0x00747874	0x72726f63	0x65747465	0x50617a7a	0x78742e32	0x6f630074	0x74657272	0x7a7a6574
0x100100c0	0x2e335061	0x00747874	0x31203533	0x36322032	0x20303820	0x2d203133	0x32203631	0x322d2033
0x100100e0	0x35322031	0x30362d20	0x20353320	0x32203231	0x30382036	0x20313320	0x2036312d	0x2d203332
0x10010100	0x32203132	0x362d2035	0x35332030	0x20323120	0x38203632	0x31332030	0x36312d20	0x20333220
0x10010120	0x2031322d	0x2d203532	0x33203036	0x32312035	0x20363220	0x33203038	0x312d2031	0x33322036
0x10010140	0x31322d20	0x20353220	0x2030362d	0x31203533	0x36322032	0x20303820	0x2d203133	0x32203631
0x10010160	0x322d2033	0x35322031	0x30362d20	0x20353320	0x32203231	0x30382036	0x20313320	0x2036312d
0x10010180	0x2d203332	0x32203132	0x362d2035	0x35332030	0x20323120	0x38203632	0x31332030	0x36312d20
0x100101a0	0x20333220	0x2031322d	0x2d203532	0x33203036	0x32312035	0x20363220	0x33203038	0x312d2031

Last memory address:0x10010d4c

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010d00	0x20312031	0x20312031	0x20302031	0x20312031	0x20312031	0x20312031	0x20312031	0x20312031
0x10010d20	0x20312031	0x20312031	0x20312031	0x20312031	0x20302031	0x20312031	0x20312031	0x20312031
0x10010d40	0x20312031	0x20312031	0x0000005d	0x00000016	0x00000000	0x00000000	0x00000000	0x00000000
0x10010d60	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010d80	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010da0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010dc0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010de0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010e00	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010e20	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010e40	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010e60	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010e80	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010ea0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

First text. address:0x00400000

Address	Code
0x00400000	0x23bdffe0
0x00400004	0xafbf0000
0x00400008	0xafb00004
0x0040000c	0xafb10008
0x00400010	0xafb2000c
0x00400014	0xafb30010
0x00400018	0xafb40014
0x0040001c	0xafb50018
0x00400020	0xafb6001c
0x00400024	0x3c011001
0x00400028	0x34240018
0x0040002c	0x3c011001
0x00400030	0x342500c8
0x00400034	0x240604b0
0x00400038	0x0c10012d
0x0040003c	0x3c011001
0x00400040	0x343000c8

Last text. address:0x00400570

Address	Code
0x00400530	0x0000000c
0x00400534	0x03e00008
0x00400538	0x24020004
0x0040053c	0x3c011001
0x00400540	0x34240000
0x00400544	0x0000000c
0x00400548	0xafbf0000
0x0040054c	0xafb00004
0x00400550	0xafb10008
0x00400554	0xafb2000c
0x00400558	0xafb30010
0x0040055c	0xafb40014
0x00400560	0xafb50018
0x00400564	0xafb6001c
0x00400568	0x23bd0020
0x0040056c	0x2402000a
0x00400570	0x0000000c

Static data block:3392 bytes
Stack maximum allocated space: 24 bytes
text. block:1392 bytes

Total memory:3780bytes

Tempo di esecuzione:
-istruzioni che si eseguono ad ogni ciclo, perciò vanno ripetute 100 volte
lw:x20
lb:x4*
sw:x22
sb:x12
istruzioni generiche:252

-istruzioni non ripetute ogni ciclo
sw:x8
lw:x8
istruzioni generiche:96

*(2 di queste però verranno eseguite circa due volte per ciclo, trovandosi nei cicli di conversione da ASCII)
perciò in totale,
((20+4(+2))*100+8)*800ps+
((22+12)*100+8)*700ps+
(252*100)+96*500ps
=(2608*800+3408*700+25296*500)ps
Tempo di esecuzione totale stimato:17120000ps=0,017120000ms

I requisiti di memoria potrebbero essere diminuiti di molto se ogni sensore fosse analizzato consecutivamente, parte dei buffer per la lettura non sarebbero più necessari, e se facessimo anche immediatamente la stampa su file, sebbene non per le politiche di aggregazione, altri 600 byte per tre buffer di uscita non sarebbero più necessari. Altrimenti possiamo utilizzare memoria dinamica per allocare la memoria dei buffer di entrata, la riduzione dipenderebbe dalla formattazione dei file di input in ogni caso, la riduzione sul sensore degli ostacoli non si avrebbe, mentre sullo sterzo potrebbe essere ridotto fino a circa 3/5 della dimensione attuale. La riduzione sul buffer del sensore di pendenza sarebbe sicuramente di molto ridotta, probabilmente a circa un terzo della dimensione attuale

Codice:

```
#Marco Casaglia 5759711 marco.casaglia@stud.unifi.it

.data
fnf:        .ascii  "The file was not found: "
file1:      .asciiz     "pendenzaIN.txt"
file2:      .asciiz     "sterzoIN.txt"
file3:      .asciiz     "distanzaIN.txt"
out1:       .asciiz     "correttezzaPendenzaOUT.txt"
out2:       .asciiz     "correttezzaSterzoOUT.txt"
out3:       .asciiz     "correttezzaDistanzaOUT.txt"
outP1:      .asciiz     "correttezzaP1.txt"
outP2:      .asciiz     "correttezzaP2.txt"
outP3:      .asciiz     "correttezzaP3.txt"

buffer1_in: .space 1200             #max Int32 is 10 characters long,to that I add a possible "-" and necessary " "
                                    #which means the maximum buffer needed is 100 times the space needed for 12
characters.
buffer2_in: .space 400              #the maximum here is a 2 digit number , plus a " " , possibly preceeded by a "-" so 100
times 4
buffer3_in: .space 400              #same as before but here is a 2 digit number, preceeded by an uppercase character, plus a
                                    #" " so 100 times 4
buffer1_out: .space 200             #the outputs are the character "1" or "0" followed by a " " so 100 times 2 character for all
outputs.
buffer2_out: .space 200
buffer3_out: .space 200
bufferP1_out: .space 200
bufferP2_out: .space 200
bufferP3_out: .space 200
prev_steer: .space 4
prev_obs: .space 4
obs_flag: .space 4

.text
.globl main

main:

addi $sp, $sp, -32
            sw $ra, 0($sp)
            sw $s0, 4($sp)
            sw $s1, 8($sp)
            sw $s2, 12($sp)
            sw $s3, 16($sp)
            sw $s4, 20($sp)
            sw $s5, 24($sp)
            sw $s6, 28($sp)


slope_check_init:                   #reads input txt's files and makes $s0 the index
            la $a0, file1                       #the slope input buffer
            la $a1, buffer1_in
```

```
        li $a2, 1200
        jal read_file
        la $s0, buffer1_in

steer_check_init:               #reads the second input and makes $s1 the index
        la $a0, file2
        la $a1, buffer2_in
        li $a2, 400
        jal read_file
        la $s1, buffer2_in

obs_check_init:                 #same as before, for the obstacle check $s2 is the
        la $a0, file3           #index of the buffer
        la $a1, buffer3_in
        li $a2, 400
        jal read_file
        la $s2, buffer3_in

init:                                   #s0 serves as a counter, $s1 to $s6 are used as
        move $t0,$s0
        move $t1,$s1
        move $t2,$s2

        li $s0,0                #indexes for the output buffers
        la $s1,buffer1_out
        la $s2,buffer2_out
        la $s3,buffer3_out
        la $s4,bufferP1_out
        la $s5,bufferP2_out
        la $s6,bufferP3_out

cycle_scheduler:                #this cycle makes sure that every operation gets done

        addi $sp, $sp, -12      #at the same time (t=$s0+1)
        sw $t0, 0($sp)                  #since there are many procedure calls, saving
        sw $t1, 4($sp)                  #registers in the stack is needed
        sw $t2, 8($sp)

        addi $sp, $sp, -12
        move $a0,$t0
        jal slope_cycle_in      #calls the slope check procedure
        sw $v0,0($sp)                   #and saves the value for the correctness
        sw $v1,12($sp)                  #also every time I have to modify the saved index for the buffer
        lw $a0,16($sp)
        move $a1,$s0
        jal steer_cycle_in      #calls the steer check procedure
        sw $v0,4($sp)                   #and saves the value for the correctness as well
        sw $v1,16($sp)
        lw $a0,20($sp)
        jal obs_cycle_in        #same goes for the obstacle sensor check
        sw $v0,8($sp)
        sw $v1,20($sp)

        lw $a0, 0($sp)                  #now the saved registers are needed for filling the
        lw $a1, 4($sp)                  #output buffers and evaluating the correctness of
        lw $a2, 8($sp)                  #system for the different politics of aggregation
        addi $sp, $sp, 12       #and loaded as arguments

        move $a3,$s1
                                        #I have 6 outputs buffers, so
                                        #the stack is needed for the other arguments

        addi $sp, $sp, -20
        sw $s2, 0($sp)
```

```
        sw $s3, 4($sp)
        sw $s4, 8($sp)
        sw $s5, 12($sp)
        sw $s6, 16($sp)

        jal buffer_fill

        move $s1,$v0
        move $s2,$v1

        lw $s3, 0($sp)
        lw $s4, 4($sp)
        lw $s5, 8($sp)
        lw $s6, 12($sp)

        addi $sp, $sp, 16

        addi $s0,$s0,1
        lw $t0, 0($sp)
        lw $t1, 4($sp)
        lw $t2, 8($sp)
        addi $sp, $sp, 12
        li $t7,100                      #after 100 cycles, all sensors data was read
        beq $t7,$s0,write_to_txt        #so the procedure for printing on text is called.
        j cycle_scheduler

write_to_txt:                           #prints every buffer to the right output .txt file,
        la $a0,     out1
        la $a1, buffer1_out
        jal write_file

        la $a0,     out2
        la $a1, buffer2_out
        jal write_file

        la $a0,     out3
        la $a1, buffer3_out
        jal write_file

        la $a0, outP1
        la $a1, bufferP1_out
        jal write_file

        la $a0,     outP2
        la $a1, bufferP2_out
        jal write_file

        la $a0, outP3
        la $a1, bufferP3_out
        jal write_file

        j end                           #then the program gets closed

slope_cycle_in:
        move $t0,$a0                    #first step of the slope check procedure, since
        lb $t1,($t0)                    #we only care about the absoulte value of the
        li $t2, 0                       #sensor value, it skips '-' if it finds one.
        bne $t1, '-', slope_cycle
        addi $t0, $t0, 1

slope_cycle:                            #this cycle reads the buffer one byte by one
        lb $t1,($t0)                    #until it finds a space or the end of the string
        beq $t1,' ',slope_check
        beq $t1,$zero,slope_check
```

```
        addi $t1, $t1, -48          #and translates it from ASCII value to an integer
        mul $t2, $t2, 10            #considering it's supposed to read a base 10 number
        add $t2, $t2, $t1           #the final value will end up in $t2
        addi $t0, $t0, 1
        j slope_cycle

slope_check:                        #now that we have an integer in $t2, we check
        li $v0,1                    #if the value of the integer is less than 60,
        li $t3,60                   #if it is, the sensor works correctly so C(t)=$v0=1
        blt $t2,$t3,slope_end
        li $v0,0                    #otherwise $v0=0

slope_end:                          #before returning to the main cycle, it skips the
        addi $t0, $t0, 1            #' ' that interrupted the cycle.
        move $v1,$t0
        jr $ra

steer_cycle_in:
        move $t0,$a0                #sets up the registers for the conversion and loads
        li $t2, 0                   #the value of the sensor at t-1
        la $t3, prev_steer
        move $t7, $a1
        beq $t7,$zero,steer_cycle   #if it's the very first cycle, there will not be any
        lw $t5,($t3)                # value in $t3

steer_cycle:                        #this cycle is exactly the same as the slope one,
        lb $t1,($t0)                #converts the characters from ASCII to integers assuming
        beq $t1,' ',steer_check     #a decimal value, values separated with spaces, and null
        beq $t1,$zero,steer_check   #terminated string
        addi $t1, $t1, -48
        mul $t2, $t2, 10
        add $t2, $t2, $t1
        addi $t0, $t0, 1
        j steer_cycle

steer_check:                        #checks the integer value obtained, finds the absolute
        subu $t4,$t5,$t2            #value of the difference between the value at t and
        abs $t4,$t4                 #the value at t-1
        sw $t2,0($t3)               #saves the current value, because it will be the previous
        li $v0,1                    #value in the next cycle
        blt $t4,11,steer_end        #if the difference is 10 or less the sensor works correctly
        beq $t7,$zero,steer_end     #if t=0, then there's no meaning in the difference and we
        li $v0,0                    #assume the sensor is working correctly

steer_end:                          #exactly the same as the slope procedure end
        addi $t0, $t0, 1
        move $v1,$t0
        jr $ra

obs_cycle_in:
        move $t0,$a0                #Some registers get loaded with service variables
        lb $t1,($t0)                #we check the first character, which might be 'A' or 'B'
        la $t6, prev_obs            #distance of the previous moving obstacle
        la $t3, obs_flag            #flag that tells if we already found 2 obstcles at the same
        li $t4, 1                   #distance.
        li $t2, 0                   #$t4 is a local 'A' or 'B' flag, $t2 will contain the value
        li $t7, 0                   #$t7 now just contains 0, to reset the obs_flag, will be
        li $t8, 58                  #useful later.
        beq $t1, 'B', obs_cycle     #This checks if it starts with 'A' or 'B'
        li $t4, 0                   #in case of 'A', resets the flags.
        sw $t7,0($t3)

obs_cycle:                          #almost the same as the previous 2 values, but now
        addi $t0, $t0, 1            #it assumes a base 16 value so it's changed accordingly
```

```
        lb $t1,($t0)
        beq $t1,' ',obs_check
        beq $t1,$zero,obs_check
        blt $t1,$t8,ASCII_correct
        addi $t1,$t1,-7
ASCII_correct:
        addi $t1, $t1, -48
        mul $t2, $t2, 16
        add $t2, $t2, $t1
        j obs_cycle

obs_check:                              #Checks are done pace by pace
        li $t5,50                       #loads the value 50 to check if the sensor is past
        li $v0,0                        #maximum distance or equal to zero, in that case
        beq $t2,$zero,obs_end           #the procedure ends with $v0=0
        blt $t5,$t2,obs_end
        li $v0,1                        #now the sensor works correctly
        beq $t4,0,obs_end               #as long as it is a static obstacle, otherwise
        lw $t7,0($t6)                   #the previous obstacle distance gets loaded
        sw $t2,0($t6)                   #and the current one gets saved
        bne $t2,$t7,reset_flag          #in case they are not the same, the flag gets reset
        lw $t5,($t3)                    #otherwise it gets loaded
        li $v0,0                        #and if it's positive, $v0=0 and the procedure ends
        beq $t5,1,obs_end               #otherwise, we set the flag to 1, which means 2
        li $t5,1                        #consecutive same distance moving obstacles found
        li $v0,1                        #and the sensor is working correctly
        sw $t5,($t3)                    #the value of the flag gets saved in memory
        j obs_end

reset_flag:                             #makes the flag equal zero
        li $t5,0
        sw $t5,($t3)

obs_end:
        addi $t0, $t0, 1
        move $v1,$t0
        jr $ra

buffer_fill:                            #saves in the buffer the results from the sensor checks first
        move $t1,$a3
        lw $t2, 0($sp)
        lw $t3, 4($sp)
        lw $t4, 8($sp)
        lw $t5, 12($sp)
        lw $t6, 16($sp)
        addi $sp, $sp, 20

        addi $t7,$a0,48                 #converted to ASCII values
        sb $t7,($t1)
        addi $t1,$t1,1
        li $t7,' '                      #and separated with ' '
        sb $t7,($t1)
        addi $t1,$t1,1
        addi $t7,$a1,48
        sb $t7,($t2)
        addi $t2,$t2,1
        li $t7,' '
        sb $t7,($t2)
        addi $t2,$t2,1
        addi $t7,$a2,48
        sb $t7,($t3)
        addi $t3,$t3,1
        li $t7,' '
        sb $t7,($t3)
```

```
	addi $t3,$t3,1

	add $t7,$a0,$a1			#then sums the values, obtaining how many sensors are working
	add $t7,$t7,$a2			#correctly in $t7.

	beq $t7,$zero,end_buffer_fill_0	#if $t7 equals zero, no sensor is working, all politics buffer
	li $t8,'1'				#get a '0', otherwise, the P3 buffer gets a '1' and a ' '
	sb $t8,($t6)
	addi $t6,$t6,1
	li $t9,' '
	sb $t9,($t6)
	addi $t6,$t6,1

	beq $t7,1,end_buffer_fill_P1 #if $t7 equals 1, the remaning buffers will get a '0',
	sb $t8,($t5)				#otherwise at least 2 sensors are working so P2 gets the
	addi $t5,$t5,1				#correctness value as well
	sb $t9,($t5)
	addi $t5,$t5,1

	beq $t7,2,end_buffer_fill_P2		#if $t7 equals 2, only 2 sensor works at the current time so
	sb $t8,($t4)				#P1 gets a 0, otherwise all sensors are working and all the
buffers
	addi $t4,$t4,1				#get a '1'
	sb $t9,($t4)
	addi $t4,$t4,1

	j end_buffer_fill

end_buffer_fill_0:			#these procedures fill the appropriate buffers with '0' and ' '
	li $t8,'0'				#following the previous procedures.
	sb $t8,($t6)
	addi $t6,$t6,1
	li $t9,' '
	sb $t9,($t6)
	addi $t6,$t6,1

end_buffer_fill_P1:
	li $t8,'0'
	sb $t8,($t5)
	addi $t5,$t5,1
	sb $t9,($t5)
	addi $t5,$t5,1

end_buffer_fill_P2:
	li $t8,'0'
	sb $t8,($t4)
	addi $t4,$t4,1
	sb $t9,($t4)
	addi $t4,$t4,1

end_buffer_fill:			#in case the buffer_fill procedure never branched, now it only needs
	move $v0,$t1
	move $v1,$t2

	addi $sp, $sp, -16
	sw $t3, 0($sp)
	sw $t4, 4($sp)
	sw $t5, 8($sp)
	sw $t6, 12($sp)

	jr $ra				#return to the main cycle

read_file:				#open and read file with specified name in $a0 and buffer address in
$a1
```

```
        move $t0,$a2
        move    $t7,$a1
        li      $v0, 13                                 # Open File Syscall
        li      $a1, 0                                  # Read-only Flag
        li      $a2, 0                                  # (ignored)
        syscall
        move    $t8, $v0                # Save File Descriptor
        blt     $v0, 0, err                             #Error message in case the file isn't found

        li      $v0, 14                                 # Read File Syscall
        move    $a0, $t8                # Load File Descriptor
        move    $a1, $t7                # Load Buffer Address
        move    $a2, $t0                        # Buffer Size
        syscall
        j close                                         # close file

write_file:                                             #open or create file end write on it with specified name in $a0
and buffer address in $a1
        move    $t7,$a1
        li      $v0, 13                                 # Open File Syscall
        li      $a1, 1                                  # write and create Flag
        li      $a2, 0                                  # (ignored)
        syscall
        move    $t8, $v0                # Save File Descriptor
        blt     $v0, 0, err                             # Goto Error

        li      $v0, 15                                 # Write File Syscall
        move    $a0, $t8                # Load File Descriptor
        move    $a1, $t7                # Load Buffer Address
        li      $a2, 200  #                     Buffer Size
        syscall
        j close                                         # close file

close:
        li      $v0, 16                                 # Close File Syscall
        move    $a0, $t8                # Load File Descriptor
        syscall
        jr      $ra                                             # return

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

end:                                    #end of the program, gets called when all other procedures are finished

        lw $ra, 0($sp)
        lw $s0, 4($sp)
        lw $s1, 8($sp)
        lw $s2, 12($sp)
        lw $s3, 16($sp)
        lw $s4, 20($sp)
        lw $s5, 24($sp)
        lw $s6, 28($sp)
        addi $sp, $sp, 32

        li      $v0, 10                                 #quit program Syscall
        syscall
```