UNIVERSITÀ DEGLI STUDI DI VERONA

Elaborato ASM
Architettura degli Elaboratori - Laboratorio

Corso di Laurea in Informatica

Jonathan Fin (VR471297) e Gabriel Adami (VR472400)

30/06/2022

Sistema di telemetria del videogame F1

Indice

Specifiche del progetto

Descrizione

Si descriva un programma che simuli il sistema di telemetria del videogame F1.

Il sistema fornisce in input i dati grezzi di giri motore (rpm), temperatura motore e velocità di tutti i piloti presenti in gara per ogni istante di tempo.

Ogni campo è diviso da una virgola usata come separatore.

Ogni riga del file di input è così composta:

`<tempo>,<id_pilota>,<velocità>,<rpm>,<temperatura>`

`id_pilota` rappresenta un valore numerico che identifica univocamente un pilota.

L'associazione tra id e nome del pilota è il seguente:

ID	NOME
0	Pierre Gasly
1	Charles Leclerc
2	Max Verstappen
3	Lando Norris
4	Sebastian Vettel
5	Daniel Ricciardo
6	Lance Stroll
7	Carlos Sainz
8	Antonio Giovinazzi
9	Kevin Magnussen
10	Alexander Albon
11	Nicholas Latifi
12	Lewis Hamilton
13	Romain Grosjean
14	George Russell
15	Sergio Perez
16	Daniil Kvyat
17	Kimi Raikkonen
18	Esteban Ocon
19	Valtteri Bottas

Obiettivo

Si scriva un programma in assembly che restituisca i dati relativi al solo pilota indicato nella prima riga del file, in base a delle soglie indicate.

Vengono definite tre soglie per tutti i dati monitorati: LOW, MEDIUM, HIGH.

Il file di output dovrà riportare queste soglie per tutti gli istanti di tempo in cui il pilota è monitorato.

Le righe del file di output saranno strutturate nel seguente modo e ordine:

`<tempo>,<livello rpm>,<livello temperatura>,<livello velocità>`

Inoltre, viene richiesto di aggiungere alla fine del file di output una riga aggiuntiva che contenga, nel seguente ordine: il numero di giri massimi rilevati, la temperatura massima rilevata, la velocità di picco e infine la velocità media.

La struttura dell'ultima riga sarà quindi la seguente:

`<rpm max>,<temp max>,<velocità max>,<velocità media>`

Le soglie per i dati monitorati sono così definite:

- Giri Motore
    - LOW: rpm <= 5000
    - MEDIUM: 5000 < rpm <=10000
    - HIGH: rpm > 10000
- Temperatura
    - LOW: temp <= 90
    - MEDIUM: 90 < temp <= 110
    - HIGH: temp > 110
- Velocità
    - LOW: speed <= 100
    - MEDIUM: 100 < speed <=250
    - HIGH: speed > 250

La velocità media viene calcolata come quoziente intero della divisione.

Se il nome del pilota inserito non è valido il programma deve restituire la stringa scritta esattamente nel seguente modo: `Invalid`.

Variabili utilizzate e loro scopo

File **telemetry.s**:

- `invalid_pilot_str` → stringa che viene scritta nel file di output quando il nome del pilota nella prima riga del file di input non è valido;
- `nome_pilota` → stringa che conterrà la prima riga del file di input, ovvero il nome del pilota;
- `tempo_str` → stringa che conterrà il tempo (da copiare sul file di output se l'ID corrisponde);
- `id_str` e `id` → stringa che conterrà l'ID e suo rispettivo intero (1 byte perché arriva massimo a 19);
- `vel_str` → stringa che conterrà la velocità;
- `vel_max` e `vel_sum` → variabili intere che servono a salvare la velocità massima e la somma di tutte le velocità (per la velocità media);
- `num_data` → numero di misurazioni effettuate per il pilota richiesto;
- `temperatura_str` e `temperatura_max` → vedi sopra;
- `rpm_str` e `rpm_max` → vedi sopra;
- `frase_errore_numero` e `frase_errore_numero_len` → stringa da stampare a video in caso di errori nell'input e sua rispettiva lunghezza;
- `spiazzamento_output` → intero che conterrà lo spiazzamento del file di output (per salvare a quale punto si è arrivati a scrivere).

File **stoid.s**:

- `pilot_<i>_str` (con <i> ∈ [0, 19]) → stringhe che contengono i nomi validi dei piloti del videogame F1. <i> è uguale all'ID corrispondente al pilota.

File **itoa.s**:

- `num_str` → stringa che conterrà il numero convertito (massimo 5 cifre).

Specifiche delle funzioni

Funzione **main.c**:

Funzione principale che prende in input da tastiera i nomi del file di input e output, mette il contenuto del file di input in una stringa, chiama la funzione `telemetry.s`, e infine scrive un'altra stringa sul file di output.

Funzione **telemetry.s**:

Funzione principale assembly che riceve in input i puntatori alle stringhe di input e output e, chiamando ulteriori funzioni, genera la stringa di output con le specifiche richieste, che verrà poi scritta nel file di output da `main.c`.

Funzione **stoid.s**:

Riceve un puntatore alla stringa `nome_pilota` nel registro `ESI` e, chiamando la funzione `cmpstr.s` per tutti i piloti, trova l'ID corrispondente (se possibile) e ne mette il valore in `AL`. Se il nome del pilota non è valido, l'ID viene impostato a 20.

Funzione **cmpstr.s**:

Riceve gli indirizzi di 2 stringhe e le confronta, restituendo `AH = 1` se sono uguali, altrimenti `AH = 0`.

Funzione **atoi.s**:

Riceve un puntatore a stringa (in `ESI`) e la converte in numero intero, restituendolo nel registro `EAX`. Se la stringa è formata da soli caratteri numerici l'operazione va a buon fine e restituisce `BL = 1`, in caso contrario `BL = 0`.

Funzione **itoa.s**:

Riceve un intero in `EAX` e lo converte in stringa (puntata da `ESI`). Ritorna inoltre in `DH` la lunghezza della stringa (nel nostro caso 5) e in `DL` la lunghezza effettiva del numero (senza contare gli 0 iniziali); ciò serve per scrivere correttamente i numeri sul file di output.

Funzione **leggi.s**:

Funzione di supporto a `telemetry.s`, per migliorare la leggibilità del codice.

Riceve un puntatore ad una stringa, sulla quale ricopia il contenuto del file di input fino alla prima virgola, fine riga o fine file.

In particolare è utilizzata per: leggere tutta la prima riga (il nome del pilota) e successivamente, per ogni riga, tempo, id, velocità, rpm e temperatura.

Funzione **scrivi_output.s**:

Riceve come parametri, sullo Stack, una serie di puntatori a stringhe (in particolare 4) e numeri interi (4) e scrive una riga intera in output, con le specifiche date (non la riga finale).

Funzione **output_level.s**:

Riceve un intero in `EAX` e le due soglie (limite LOW e HIGH) in `ECX` e `EDX`, secondo le specifiche date. È una funzione di supporto a `scrivi_output.s` e scrive in output i livelli (LOW, MEDIUM oppure HIGH) per ogni dato valido.

Funzione **scrivi_stats.s**:

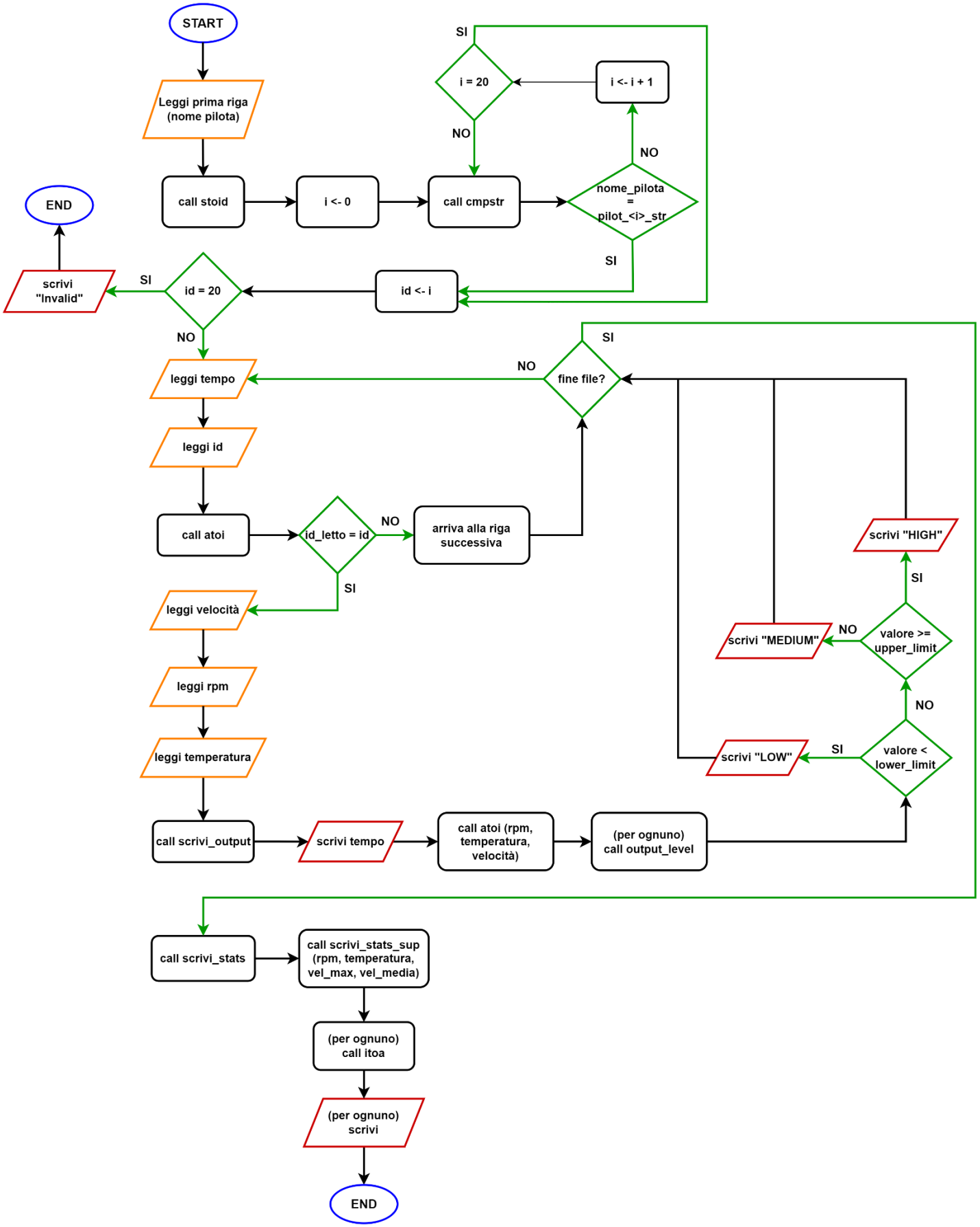
Simile a `scrivi_output.s`, riceve sullo Stack 5 interi, i quali servono per scrivere in output l'ultima riga del file.

Funzione **scrivi_stats_sup.s**:

Funzione di supporto a `scrivi_stats.s`, che riceve un intero in `EAX`, lo converte e lo scrive sul file di output.

Le interazioni tra le varie funzione verranno spiegate più dettagliatamente nel diagramma di flusso che segue.

Diagramma di flusso



Tempi di esecuzione

Abbiamo misurato tempi di esecuzione dello stesso programma in 3 diversi linguaggi: Assembly, C e Python.

Qui riportiamo i tempi medi (in secondi) di 5 esecuzioni:

	Python	C	Assembly
`real`	`3,787`	`7,647`	`0,268`
`user`	`3,519`	`0,390`	`0,196`
`sys`	`0,353`	`0,461`	`0,064`

Dai dati si evince che nei linguaggi più a basso livello si ha un tempo di esecuzione inferiore (stiamo guardando il tempo `user`): in particolare (in questo caso), C è circa 9 volte più veloce di Python, mentre Assembly è circa 2 volte più veloce di C.

Si può notare che C ha un tempo `real` molto alto, probabilmente a causa della bassa priorità assegnatagli dal sistema.

I test sono stati effettuati su una macchina virtuale (VMware) con S.O. Ubuntu 21.10, con 6GB di RAM e 3 cores a disposizione e il minor numero di finestre aperte.

Riportiamo in seguito le specifiche del pc usato per la misurazione dei tempi:

- CPU: Intel Core i7-10750H, 2.60 GHz, 6 Cores, 12 Threads
- RAM: 16GB, SO-DIMM DDR4, 2933 MHz
- SSD: INTEL SSDPEKNW512G8H
- GPU: NVIDIA GeForce GTX 1650Ti, 4GB

Scelte progettuali

Variabili

La stringa `nome_pilota` ha 20 caratteri perché il nome più lungo è di 18 caratteri, quindi ce n'è 1 in più per il nome errato (se per esempio c'è una lettera aggiunta in fondo al nome) e 1 per il carattere terminatore `'\0'`.

Tutte le stringhe hanno un numero di caratteri sufficiente (e non eccessivo) per leggere correttamente tutti i dati in input.

`id` è una variabile di tipo `byte`, perché sicuramente l'ID non supererà 20 (può arrivare comunque fino a 255).

Salvataggio valori registri

I registri importanti che verranno modificati nelle funzioni, vengono salvati (nello Stack) nel file prima di chiamare la funzione.

Funzioni

I parametri delle funzioni usate vengono salvati principalmente nei registri. Per 2 funzioni è stato usato lo Stack perché i parametri erano numerosi.

Abbiamo deciso di utilizzare funzioni di supporto per semplificare il codice e renderlo più leggibile.

Per la funzione `atoi.s` è previsto un controllo su errori in input. Se si vuole convertire una stringa formata da caratteri non numerici, viene stampato a video un messaggio di errore e il programma termina.

Considerazioni finali

Il numero totale di file creati (contando anche main.c) è 11.

È stato necessario creare alcuni funzioni di supporto, in quanto scrivere tutto su un unico file rendeva praticamente impossibile la comprensione del codice; facendo in questo modo, si può lavorare su ogni parte quasi indipendentemente, come con linguaggi ad alto livello.