Architettura degli Elaboratori: Elaborato Assembly

Mirko Morati, Noè Murr

10 luglio 2016

Indice

1 Premessa

Per la corretta lettura del pdf si informa che sono stati inseriti dei riferimenti per ogni etichetta e variabile trattata in modo da poter vedere e approfondire il corrispettivo codice nel programma.

2 Descrizione del progetto

Si vuole realizzare un programma `Assembly` per il monitoraggio di un motore a combustione interna il quale, ricevuto come ingresso il numero di giri/minuto del motore, fornisca in uscita la modalità di funzionamento corrente del motore: *Sotto Giri, Ottimale, Fuori Giri*. Il programma deve contare e visualizzare in uscita il numero dei secondi trascorsi nella modalità di funzionamento attuale ed inoltre attivare il segnale di allarme nel caso in cui il motore si trovi in modalità *Fuori Giri* da più di 15 secondi.

Di seguito verranno descritte le funzioni presenti in ogni file del programma, etichette, eventuali variabili e loro scopo.

3 syscall.inc

Header file contenente la definizione di alcune costanti, tramite la pseudo-operazione `.equ`, relative alle chiamate di sistema e ad alcuni standard utilizzati in tutti i file e riportati di seguito:

SYS_EXIT	1
SYS_READ	3
SYS_WRITE	4
SYS_OPEN	5
SYS_CLOSE	6
STDIN	0
STDOUT	1
STDERR	2
SYSCALL	0x80

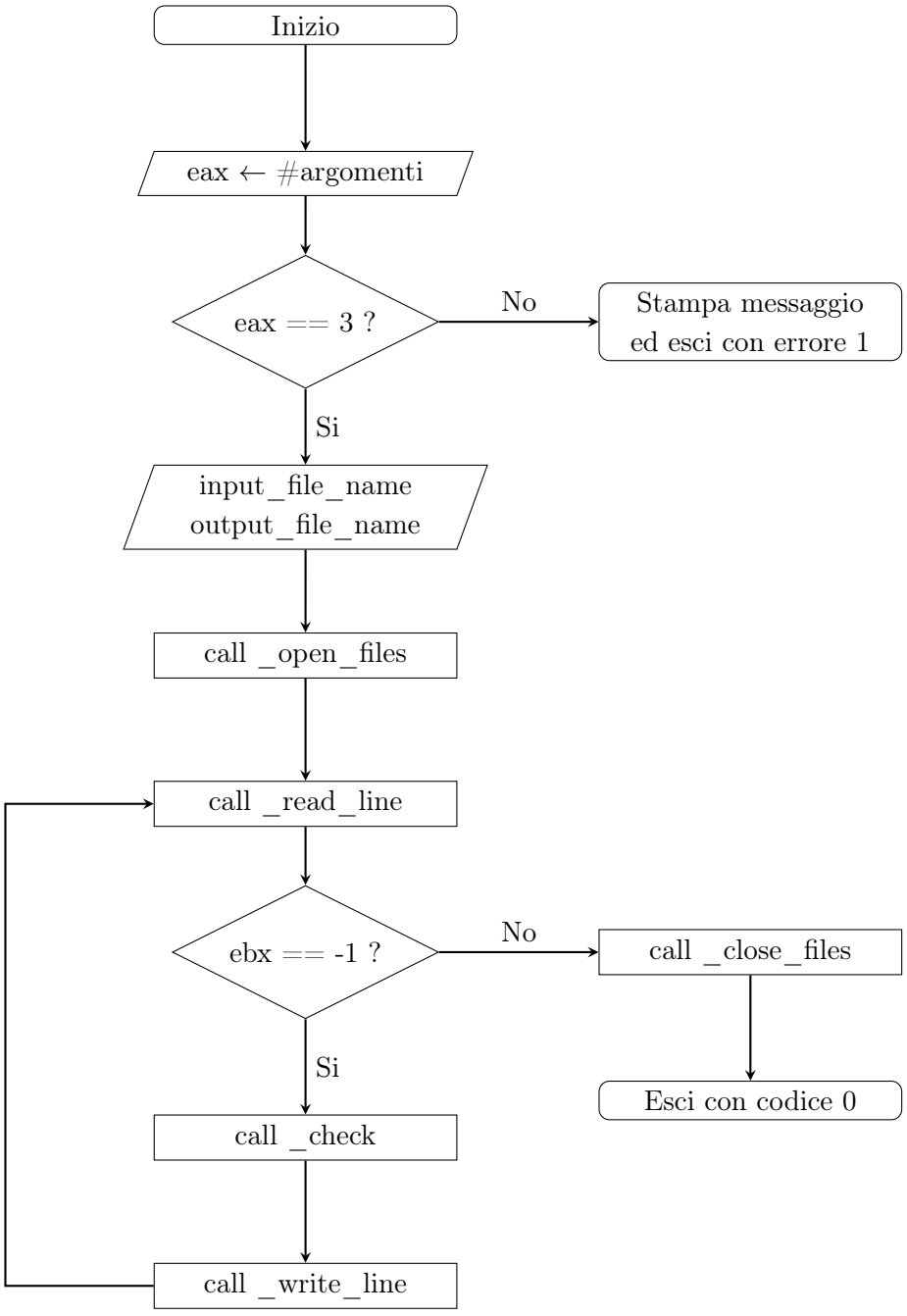
Scelte Progettuali

Nonostante il codice `Assembly` sia fortemente legato alla macchina sottostante, si è deciso, di utilizzare un "trucchetto" per permetterne, per quanto possibile, la portabilità agevole su un'altra piattaforma. Infatti sarà sufficiente cambiare i codici delle `SYS_CALL` nel file `syscall.inc` per utilizzare le `SYS_CALL` di un altro sistema operativo. Questo vale solo su sistemi unix-like poiché i parametri delle funzioni essenziali (read, write, ...) sono comuni grazie allo standard `POSIX`.

4 main.s

File principale del programma.

4.1 Flowchart



4.2 Variabili Globali

- `input_fd`: Contiene il descrittore del file di input;
- `output_fd`: Contiene il descrittore del file di output;
- `init`: Contiene il valore del segnale INIT corrente;
- `reset`: Contiene il valore del segnale RESET corrente;
- `rpm`: Contiene il valore del segnale RPM corrente;
- `alm`: Contiene il valore del segnale ALM corrente;
- `mod`: Contiene il valore del segnale MOD corrente;
- `numb`: Contiene il valore del segnale NUMB corrente.

Scelte Progettuali

Si è scelto di utilizzare in modo estensivo le variabili globali in quanto in un progetto di così ridotte dimensioni si aumenta notevolmente la leggibilità del codice e nel contempo se ne riduce la complessità di scrittura. Se il progetto avesse richiesto computazioni più complesse questo approccio non sarebbe stato l'ideale, poiché sarebbero state compromesse le prestazioni totali dell'applicazione.

4.3 Variabili Locali

- `usage`: Stringa per la descrizione del corretto utilizzo del programma;
- `USAGE_LENGTH`: Contiene la lunghezza della stringa, necessaria per la stampa.

4.4 Funzioni ed Etichette

- `_start`: Punto di entrata del programma. Si occupa di controllare che il numero di parametri sia corretto, in caso contrario stampa la stringa `usage` e termina. Dopo il controllo chiama la funzione `_open_files` definita nel file `open_files.s`.
- `_main_loop`: Loop principale. Viene chiamata la funzione `_read_line` definita nel file `read_line.s`, nel caso in cui il contenuto del registro `ebx` sia equivalente a -1 significa che il file di input è terminato (**EOF** → End Of File) quindi salta a `_end`, altrimenti chiama la funzione `_check` definita nel file `check.s` e la funzione `_write_line` definita nel file `write_line.s`, dopodiché riesegue il ciclo.
- `_end`: Si occupa di chiudere tutti i file aperti e della corretta uscita dal programma tramite la chiamata di sistema `exit`.
- `_show_usage`: Nel caso in cui i parametri non siano corretti stampa a video la stringa `usage` e termina il programma segnalando errore con il codice 1.

5 open_files.s

Contiene la funzione che si occupa di aprire i file in modo corretto.

5.1 Variabili Locali

- `error_opening_files`: Stringa di errore in caso di errata apertura dei file (e.g. file mancante, file corrotto, ...).
- `ERROR_OPENING_LENGTH`: Costante che contiene la lunghezza della stringa di errore.

5.2 Funzioni ed Etichette

- `_open_files`: Si occupa di aprire i file e gestisce eventuali errori. In caso il file di output non esistesse, questo viene creato in automatico con permessi di lettura e scrittura. I descrittori ottenuti vengono salvati nelle corrispondenti variabili globali.
- `_error_opening_files`: In caso di errore viene stampato su `STDERR` la stringa opportuna, dopodiché il programma viene terminato con codice di errore 2.

6 read_line.s

Contiene la funzione che si occupa di leggere ed interpretare una riga per volta del file di input.

6.1 Variabili Locali

- `input_buff`: Buffer di dimensione `INPUT_BUFF_LEN` che conterrà i caratteri della riga letta dal file di input.
- `INPUT_BUFF_LEN`: Dimensione del buffer.

Scelte Progettuali

Si è scelto di utilizzare un buffer di dimensione 9 byte corrispondente alla lunghezza di una singola riga. Questa scelta è molto importante perché permette di non realizzare soluzioni hardcoded. La consegna specifica che non verranno utilizzati file di input con più di 100 righe: sfruttando tale informazione si sarebbe potuto scrivere un codice semplicistico che leggesse in un buffer l'intero file. Soluzione poco elegante e mal ottimizzata. Un'altra strada prendibile sarebbe stata quella di leggere un byte alla volta dal file di input, ma questa avrebbe richiesto un numero spropositato e inutile di chiamate al kernel. Con la lettura di una riga per volta il buffer rimane snello e comodo da manipolare. Questa soluzione permette all'applicazione di lavorare con file di dimensione arbitraria, inoltre lo riteniamo un metodo ottimale in quanto stabilisce un compromesso tra l'uso di `SYS_CALL` e flessibilità del codice. Oltre a questo si sarebbe potuto utilizzare un ulteriore metodo: quello di mappare in memoria il file ed accedervi come in un array, ma questo sforava dalle nozioni fornite dal corso.

6.2 Funzioni ed Etichette

- `_read_line`: Legge dal file una riga tramite la chiamata di sistema `read` e si occupa di richiedere la traduzione dei caratteri letti in interi mediante la funzione `_atoi` definita nel file `atoi.s` salvandoli successivamente nelle rispettive variabili globali. In caso i caratteri letti siano pari a 0 salta all'etichetta `_eof`;
- `_eof`: In caso di `EOF` mette -1 in `ebx` e ritorna.

7 atoi.s

Contiene la funzione che si occupa di convertire una serie di caratteri `ASCII` in un numero intero.

7.1 Funzioni ed Etichette

- `_atoi`: Vengono inizializzati i registri necessari alla conversione;
- `_atoi_loop`: Loop principale, converte la stringa puntata dal registro `edi` in un intero salvato e ritornato in `eax`.

8 check.s

Contiene la funzione che si occupa di settare sulla base dei valori di input e dei valori del ciclo precedente i corretti parametri delle variabili `alm, mod, numb`.

8.1 Funzioni ed Etichette

- `_check`: In base ai valori di `init` e `rpm` si occupa di saltare all'etichetta corretta.
- `_fg / _sg / _opt`: Etichette corrispondenti alle modalità di funzionamento previste dalle specifiche. Si occupano di settare i corretti valori di `alm, mod, numb`. L'unica modalità che necessita di una gestione particolare è `fg` in cui bisogna settare l'eventuale allarme.
- `_reset_numb`: Nel caso in cui sia stata cambiata la modalità di funzionamento oppure il valore della variabile `reset` sia pari a 1, viene settata la modalità corretta, resettato il conteggio `numb` e "spento" `alm`.
- `_set_alm`: Se il motore è nella modalità `fg` da più di 15 secondi, viene "acceso" l'allarme portando il valore di `alm` a 1.
- `_init_0`: Se il valore di `init` è pari a 0 tutte le variabili di output vengono poste a 0 dal momento che il motore è spento.
- `_end_check`: Abbiamo ritenuto opportuno (per evitare di preoccupare troppo il conducente) considerare un numero di secondi di massimo due cifre. Prima di terminare la funzione si controlla che il valore di `numb` non sia superiore a 99, in caso si salta a `_numb_overflow` che azzera `numb`.

Scelte Progettuali

Si è scelto di dividere il progetto in relativamente poche funzioni e di tenere tutti i controlli relativi a input/output in un'unica funzione chiamata `_check`. Sarebbe stato tranquillamente possibile fare una divisione in funzioni minori per aumentare la leggibilità (e.g. `check_rpm`, `check_init`, ...), ma è stato ritenuto più pratico per la semplicità dei controlli mantenere il codice in un unico file.

9 write_line.s

Contiene la funzione che si occupa di creare la stringa di output e scriverla sul corrispondente file. Ricordiamo qui la codifica utilizzata delle modalità di funzionamento e la struttura di una singola riga di output:

Spento	0	00
SG	1	01
OPT	2	10
FG	3	11

`alm` (1 byte)	,	`mod` (2 byte)	,	`numb` (2 byte)	`\n`

9.1 Variabili Locali

- `output_buff`: Buffer di dimensione `OUTPUT_BUFF_LEN` che conterrà i caratteri della stringa da scrivere sul file di output.
- `OUTPUT_BUFF_LEN`: Dimensione del buffer.
- `MOD_XX`: Stringhe costanti che identificano la modalità di funzionamento corrispondente in binario.
- `MOD_LEN`: Dimensione delle stringhe di modalità.

9.2 Funzioni ed Etichette

- `_write_line`: Inizializza i registri necessari alla scrittura sulla variabile buffer.
- `_alm_X`: Aggiunge al buffer il corretto valore di `alm`.
- `_print_mod`: Aggiunge al buffer una virgola come separatore e in base al valore di `mod` salta alla corrispondente etichetta.
- `_mod_X`: La stringa corrispondente alla modalità X codificata in binario viene messa in `eax`, in seguito viene aggiunta al buffer nell'etichetta `_end_print_mod`.
- `_print_numb`: Si occupa di aggiungere al buffer il valore di `numb` opportunamente convertito in `ASCII` e di terminare la stringa con il carattere `\n`.

- `_numb_one_digit`: Se il valore di `numb` è minore di 10 si occupa di aggiungere uno 0 come prima cifra.

10 itoa.s

Contiene la funzione per convertire un valore da intero a una corrispondente stringa `ASCII`.

10.1 Funzioni ed Etichette

- `_itoa`: Inizializza i registri necessari alla conversione.
- `_itoa_dividi`: Si occupa di contare i caratteri necessari per la stringa e di posizionare il carattere `\0` alla fine della stringa.
- `_itoa_converti`: Scrive ogni cifra nella posizione corretta della stringa.

11 close_files.s

Contiene la funzione per chiudere correttamente un file.