Università degli Studi di Verona
Facoltà di Scienze MM.FF.NN.
AA 2015/2016
Corso di laurea in Informatica

Elaborato Assembly
Laboratorio Architettura degli Elaboratori

Università degli studi di Verona

Corso di laurea Informatica

Faggion Andrea (VR401550)
Mori Samuele (VR397217)
Valentino Wilma (VR397645)

07/06/2016

Indice

Specifiche elaborato

Si sviluppi un programma Assembly che effettua il monitoraggio di un motore a combustione interna. Ricevendo come input il numero di giri/minuto del motore (RPM) e, una volta impostate le soglie minima e massima per il funzionamento ottimale, il programma fornisce in uscita una modalita' di funzionamento del motore: sotto-giri (SG), in regime ottimale (OPT) o fuori-giri (FG). Si vuole inoltre avere in uscita da quanto tempo il sistema si trova nello stato attuale ed un ulteriore output di allarme che vale 1 se e soltanto se il sistema si trova in stato FG da piu' di 15 secondi (cicli di clock).

Il programma deve essere lanciato da riga di comando con due stringhe come parametri, la prima stringa identifica il nome del file .txt da usare come input, la seconda quello da usare come output:
./run filenameInput.txt filenameOutput.txt

Il programma deve leggere il contenuto di filenameInput.txt contenente in ogni riga i seguenti valori:

INIT,RESET,RPM

• INIT: valore binario, quando vale 0 il programma deve restituire una linea composta di soli 0; quando vale 1 il programma deve fornire in uscita tutti i valori come da specifica.
• RESET: valore binario, se posto a 1 il contatore dei secondi deve essere posto a zero.
• RPM: valore del numero di giri ricevuto dal rilevatore (valore massimo 6500).

Il programma deve restituire i risultati del calcolo in filenameOutput.txt in cui ogni riga contiene:

ALM,MOD,NUMB

• MOD: indica in quale modalita' di funzionamento si trova l'apparecchio al momento corrente (00 spento, 01 SG, 10 OPT, 11 FG)
• NUMB: indica i secondi trascorsi nell attuale modalita'.
• ALM: valore binario, messo a 1 se viene superato il tempo limite in FG.

I valori delle soglie sono i seguenti:
• RPM < 2000 → SG
• 2000 ≤ RPM ≤ 4000 → OPT
• RPM > 4000 → FG

Assieme al presente documento sono forniti due files di test testinput.txt e testoutput.txt. Per un voto sufficiente sull'elaborato e' richiesto il perfetto funzionamento del programma su questi due files.

Ai fini del test del progetto, verranno usati in fase di esame dei files diversi, che tuttavia avranno le stesse caratteristiche di quelli di esempio ed una lunghezza non superiore alle 100 righe.

Funzioni

Nel nostro elaborato abbiamo creato le seguenti funzioni:

- main
- atoi1
- atoi4
- contatore
- controllo_stato
- divisione_stampa
- sistema_spento
- stampa_riga

Per ogni funzione abbiamo anche specificato le variabili e le modalità di passaggio e restituzione dei valori.

Funzione MAIN

La funzione ha il compito principale di leggere un file in ingresso, elaborare i dati in esso contenuto e scriverli in un secondo file.
Vengono utilizzate le seguenti variabili:

- ***input*** : di tipo long, inizializzata a zero, viene salvato il puntatore al file di input.
- ***output*** : di tipo long, inizializzata a zero, viene salvato il puntatore al file di output.
- ***init*** : di tipo long, inizializzata a zero, viene salvato init.
- ***last_init*** : di tipo long, inizializzata a zero, viene salvato l'init del ciclo predente.
- ***reset*** : di tipo long, inizializzata a zero, viene salvato reset.
- ***rpm*** : di tipo long, inizializzata a zero, viene salvato rpm.
- ***alm*** : di tipo long, inizializzata a zero, viene salvato alm.
- ***mod*** : di tipo long, inizializzata a zero, viene salvato mod.
- ***last_mod*** : di tipo long, inizializzata a zero, viene salvato il mod del ciclo precedente.
- ***numb*** : di tipo long, inizializzata a zero, viene salvato numb.
- ***buff_size*** : di tipo long, inizializzata a nove, è la dimensione del buff.
- ***cont*** : di tipo int, inizializzata a zero, viene salvato il contatore.

La funzione all'inizio apre il file da leggere e il file da scrivere, creando due puntatori: **Input** e **Output.**

Si entra quindi nel ciclo principale del codice:

1) Viene controllato il contatore del ciclo (il file di lettura può avere al max 100 righe), se siamo arrivati al 101° giro esco immediatamente dal programma. Altrimenti viene incrementato il contatore e si continua con il codice.
2) Lettura riga per riga del file sorgente, utilizzando il puntatore precedentemente creato (Input) che tiene traccia di quale riga è stata letta, consentendo ad ogni ciclo di leggere la riga successiva.
   La riga letta viene salvata in un **buff** delle dimensioni di **buff_size**.
   Controllando poi che non sia presente il carattere di fine file, tramite il comando test.

3) Utilizzando la funzione *atoi1* leggo il primo carattere della riga → **init**.
   Comparandolo con 0, se vero salto all'etichetta Spento, dove scrivo nel file di uscita e alla riga corrispondente la stringa di 0,00,00.
   Altrimenti continuo con il flusso del programma.
4) Spaziando all'interno della riga letta (cioè sommando 2 al puntatore al buff), posso leggere i valori di **reset** e **rpm**, tramite le funzioni *atoi1* e *atoi4*, salvandoli.
5) Caricando il valore attuale dei giri del motore (**rpm**), invoco la funzione *Controllo_Stato,*che mi restituisce la modalità attuale del mio sistema salvata in **mod**.
6) Controllo se **last_init** è uguale a 0, se SI salto al etichetta *controllo_caso_particolare*, caso che si verifica quando il precedente valore di **init** era 0, mentre l'attuale valore è di 1, quindi possiamo reimpostare il contatore e l'allarme a 0.
7) Immettendo i valori di **mod**, **reset**, **last_mod** e **numb**, richiamiamo la funzione *Contatore* che ci restituisce i valori aggiornati di **alm** e **numb**.
   Salviamo inoltre la modalità attuale in **last_mod** che ci servità nel prossimo ciclo.
8) Abbiamo ora tutti i valori aggiornati da scrivere sul nostro secondo file, passando tali valori e il puntatore al file, possiamo invocare la funzione *Stampa_Riga*.
   Salvando infine il valore attuale di **init** in **last_init** e ritornando all'inizio del ciclo.

Il main termina con la syscall 1 di uscita quando arriviamo al ciclo 101, oppure se finiamo di leggere ed elaborare i dati del file di lettura.

Funzione ATOI1

La funzione Atoi1 converte un singolo carattere, il cui indirizzo si trova in eax, in un numero che viene restituito nel registro eax.
In questa funzione usiamo una sola variabile:

- ***car*** : di tipo byte, viene usata per convertire un carattere ASCII in un numero intero.

La funzione modifica ebx, ecx, edx, quindi abbiamo fatto in modo di salvare il loro contenuto nello stack con delle push per poi a fine funzione riprestinarle con dei pop.
La funzione restituisce:

- Nel registro eax il valore numerico del carattere letto.

Funzione ATOI4

La funzione Atoi4 converte una stringa formata da quattro caratteri (devono essere numerici), il cui indirizzo si trova in eax, in un numero che viene restituito nel registro eax.
In questa funzione usiamo le variabili:

- ***car*** : di tipo byte, viene usata per convertire un carattere ASCII in un numero intero.
- ***counter*** : di tipo int, viene usata nel ciclo come contatore in modo che non faccia più di quattro cicli.

La funzione modifica ebx, ecx, edx, quindi abbiamo fatto in modo di salvare il loro contenuto nello stack con delle push per poi a fine funzione riprestinarle con dei pop.
La funzione restituisce:

- Nel registro eax il valore numerico del carattere letto.

Funzione CONTATORE

La funzione Contatore riceve in input:

- MOD salvato nel registro eax
- RESET salvato nel registro ebx
- LAST_MOD salvato nel registro ecx
- NUMB salvato nel registro edx

La funzione modifica NUMB in base ai valori di RESET.
Se RESET è a 1, NUMB viene azzerato. Se MOD e LAST_MOD sono uguali, NUMB viene incrementato di uno.
Se MOD è uguale a 11 e NUMB è maggiore di 14, ALM viene impostato a 1.
In questa funzione non utilizziamo variabili.
La funzione restituisce:

- MOD nel registro eax
- ALM nel registro ebx
- NUMB nel registro edx

Funzione CONTROLLO_STATO

La funzione Controllo_stato riceve in input RPM, salvato nel registro eax. Restituisce MOD in base al valore di RPM:

- Se RPM è 0, MOD viene impostato a 00.
- Se RPM è minore di 2000, MOD viene impostato a 01.
- Se RPM è maggiore di 4000, MOD viene impostato a 11.
- Altrimenti (quando RPM è compreso fra 2000 e 4000), MOD viene impostato a 10.

In questa funzione non utilizziamo variabili.
La funzione restituisce:

- MOD nel registro eax

Funzione DIVISIONE_STAMPA

La funzione Divisione_stampa riceve in input:

- un numero intero (formato da due cifre) salvato nel registro eax
- il puntatore al file di output salvato nel registro ecx

La funzione divide il numero per dieci, il risultato viene salvato nella variabile decine mentre il resto in quella delle unita. Le due variabili vengo stampate sul file di output.
In questa funzione usiamo le seguenti variabili:

- ***unita*** : di tipo long, viene salvato il resto della divisione
- ***decine*** : di tipo long, viene salvato il risultato della divisione
- ***output*** : di tipo long, viene salvato il puntatore al file di output

La funzione non restituisce nessun parametro.

Funzione SISTEMA_SPENTO

La funzione Sistema_spento riceve in input il puntatore al file di output salvato nel registro eax e stampa sul file la stringa "0,00,00\n".
In questa funzione usiamo una sola variabile:

- ***string*** : di tipo ascii, viene salvata la stringa da stampare ("0,00,00\n").

La funzione modifica ebx, ecx, edx, quindi abbiamo fatto in modo di salvare il loro contenuto nello stack con delle push per poi a fine funzione riprestinarle con dei pop.
La funzione non restituisce nessun parametro.

Funzione STAMPA_RIGA

La funzione Stampa_riga riceve in input:

- MOD salvato nel registro eax.
- ALM salvato nel registro ebx.
- OUTPUT salvato nel registro ecx.
- NUMB salvato nel registro edx.

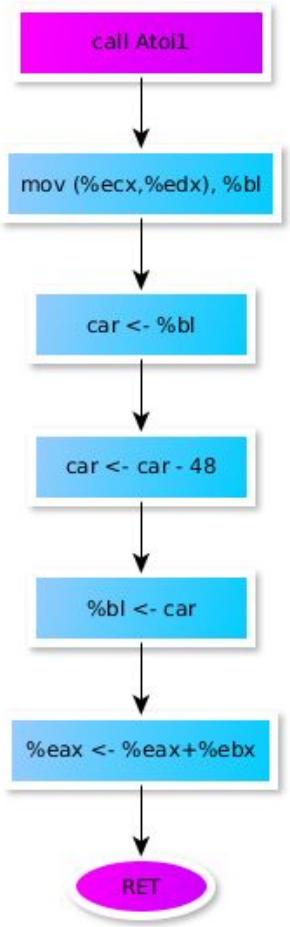
In primis vengono salvati i dati passati dai registri nelle corrispettive variabili.
La funzione stampa sul file di **output** una riga formata da: ALM, il carattere virgola, MOD (usando la funzione Divisione_stampa), il carattere virgola,NUMB (usando la funzione *Divisione_stampa*) e il carattere new line.
La funzione viene richiamata ad ogni ciclo per scrivere passo passo il file di output.
Al termine di ogni ciclo di stampa infatti un jump fa ripartire da capo la lettura del file!
In questa funzione usiamo le seguenti variabili:

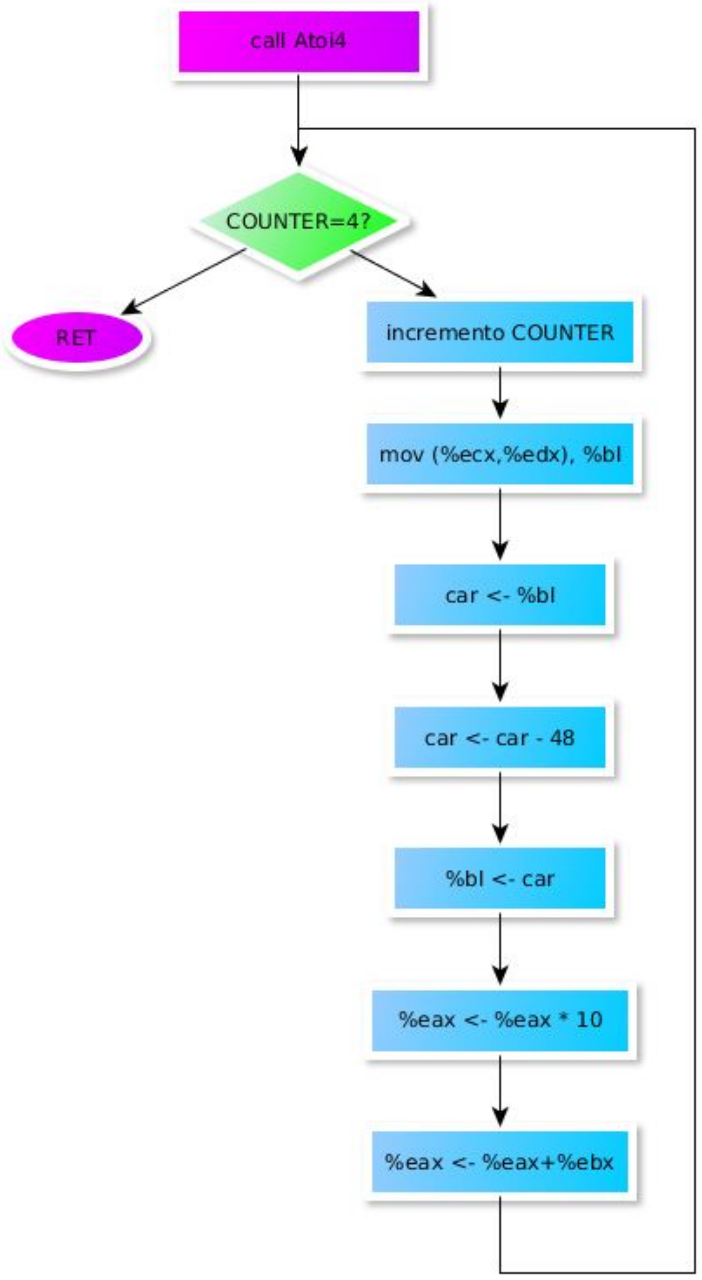
- ***virgola*** : di tipo ascii, stringa formata dal carattere ‘,’ (virgola)
- ***newline*** : di tipo ascii, stringa formata dal carattere ‘\n’ (new line)
- ***output*** : di tipo long, viene salvato il puntatore al file di output
- ***alm*** : di tipo long, viene salvato il valore di ALM
- ***mod*** : di tipo long, viene salvato il valore di MOD
- ***numb*** : di tipo long, viene salvato il valore di NUMB

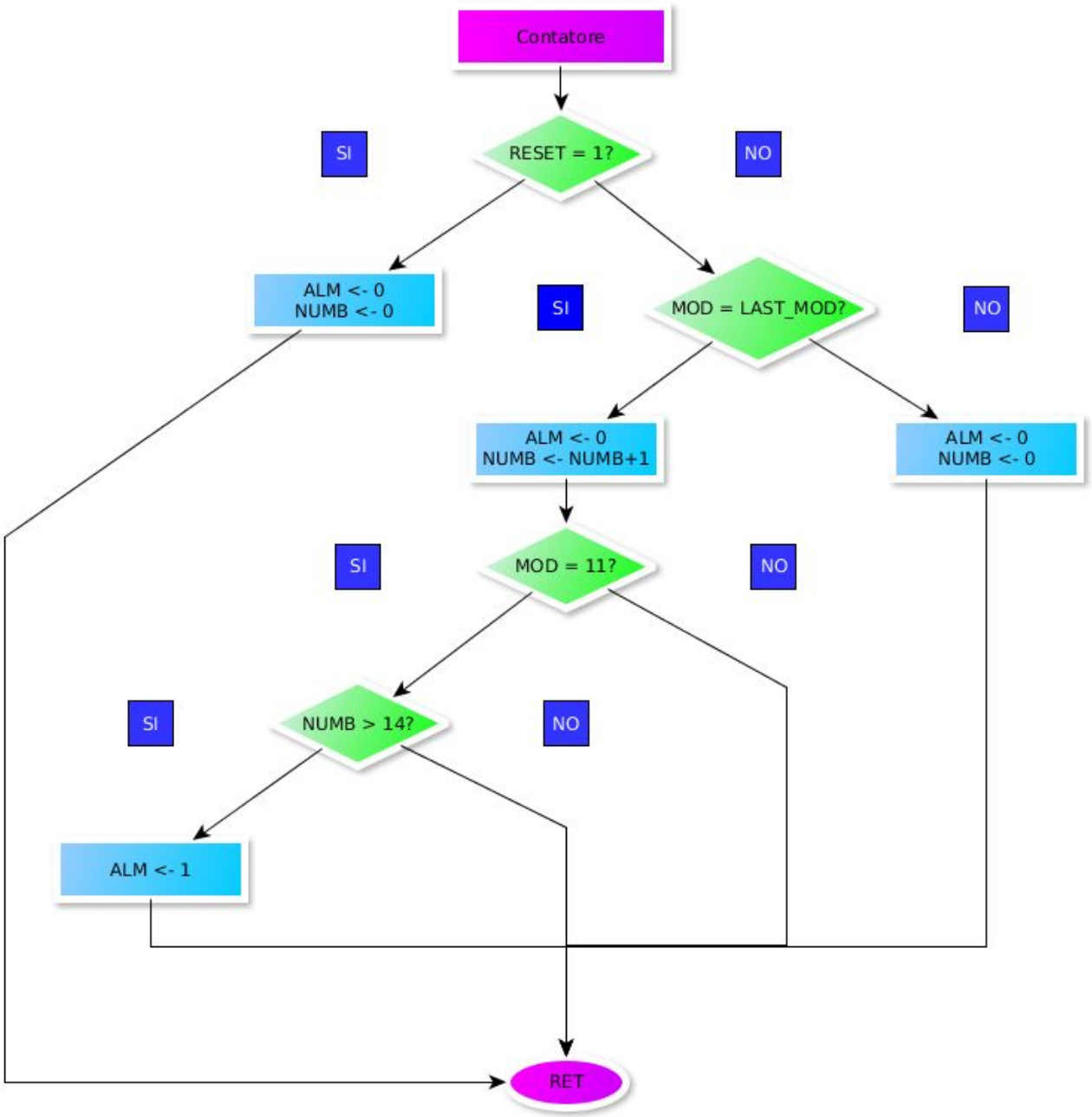
La funzione non restituisce nessun parametro.

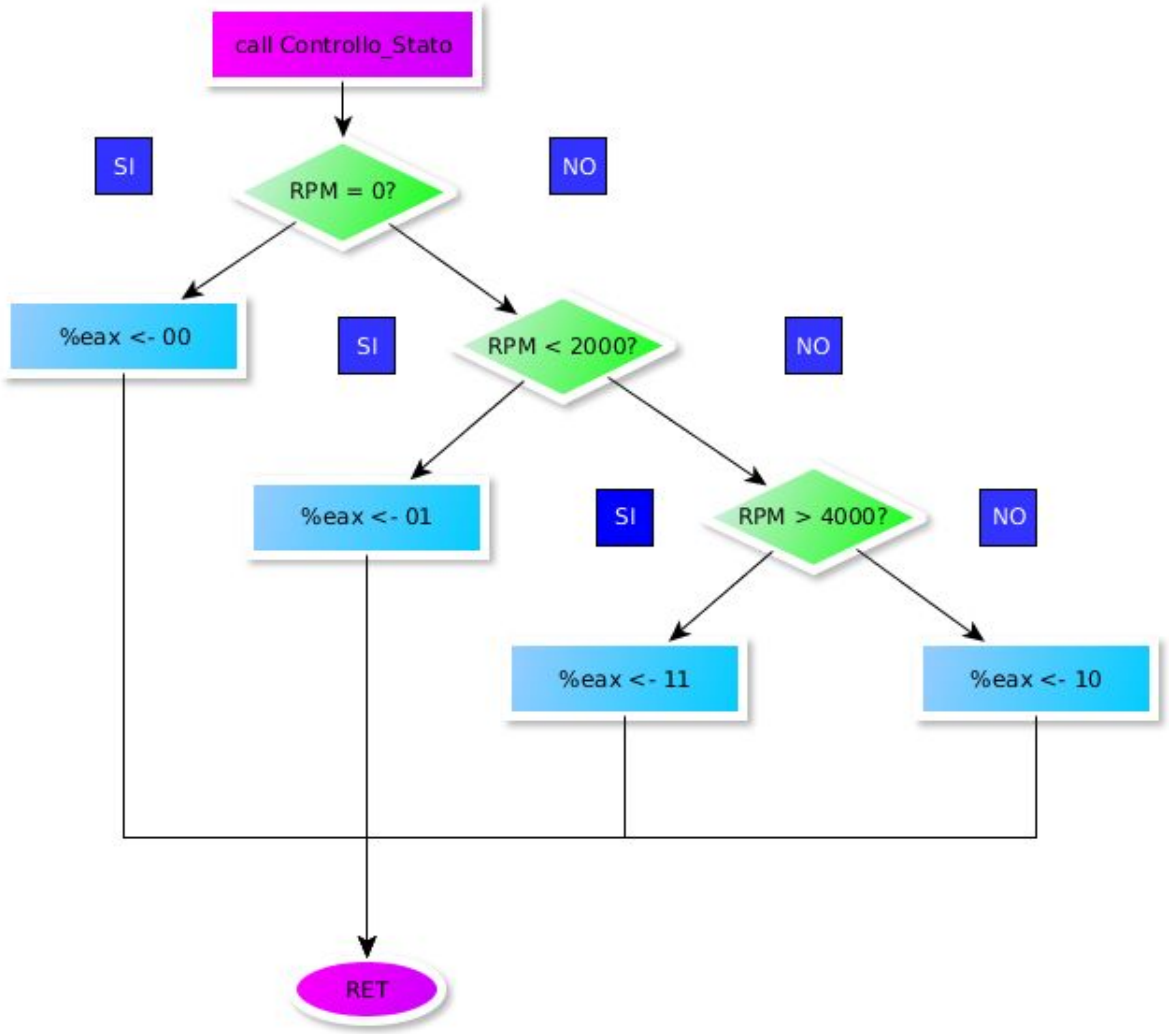
Diagramma di flusso

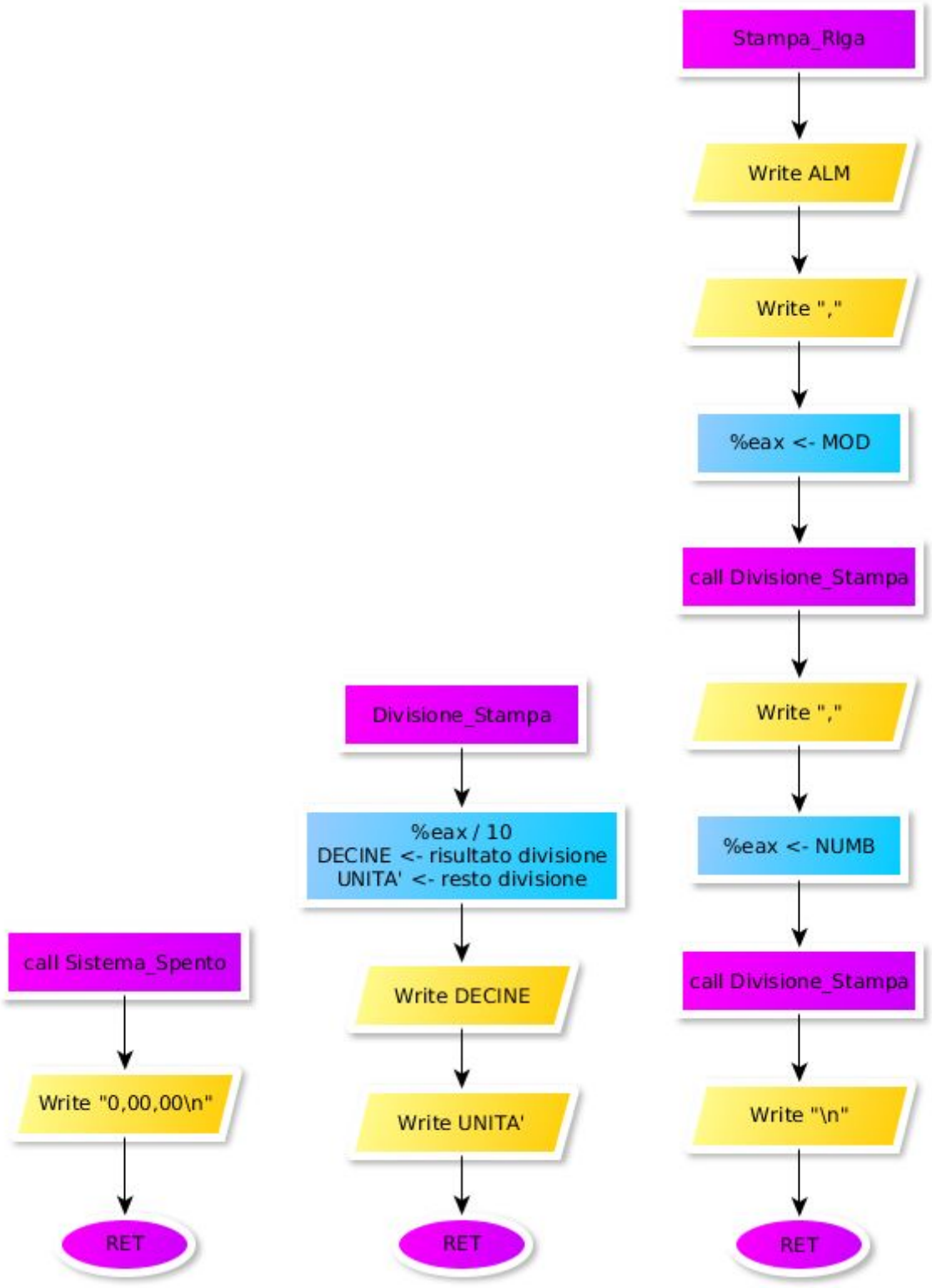
(diagrammi disponibili in file .svg e .png nella cartella Diagrammi)

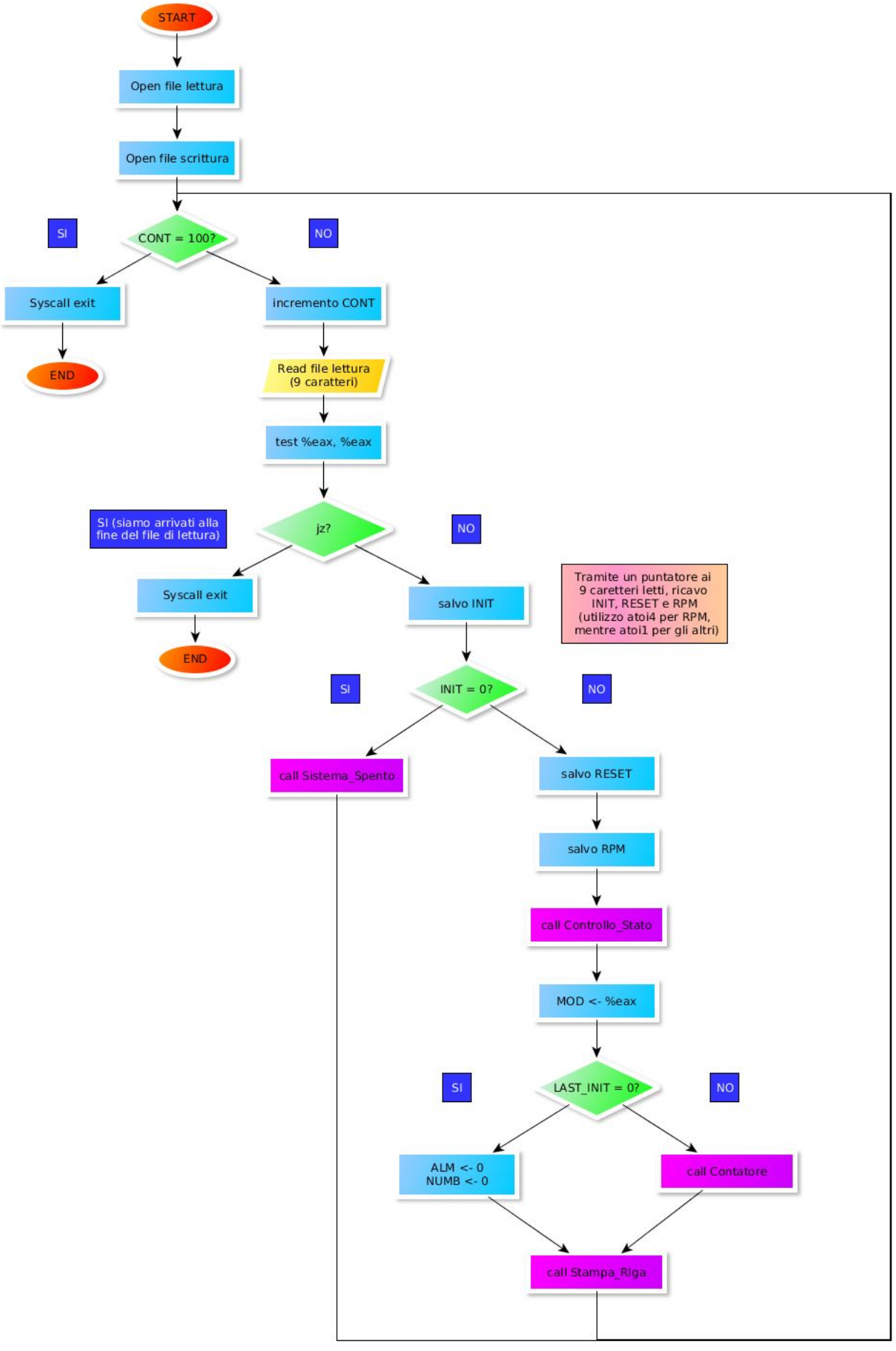





Contatore
RESET = 1?
SI
NO
ALM <- 0
NUMB <- 0
MOD = LAST_MOD?
SI
NO
ALM <- 0
NUMB <- NUMB+1
ALM <- 0
NUMB <- 0
MOD = 11?
SI
NO
NUMB > 14?
SI
NO
ALM <- 1
RET


call Controllo_Stato
SI
RPM = 0?
NO
%eax <- 00
SI
RPM < 2000?
NO
%eax <- 01
SI
RPM > 4000?
NO
%eax <- 11
%eax <- 10
RET


Stampa_Riga
Write ALM
Write ","
%eax <- MOD
call Divisione_Stampa
Write ","
%eax <- NUMB
call Divisione_Stampa
Write "\n"
RET
Divisione_Stampa
%eax / 10
DECINE <- risultato divisione
UNITA' <- resto divisione
Write DECINE
Write UNITA'
RET
call Sistema_Spento
Write "0,00,00\n"
RET


START
Open file lettura
Open file scrittura
SI
CONT = 100?
NO
Syscall exit
incremento CONT
END
Read file lettura
(9 caratteri)
test %eax, %eax
SI (siamo arrivati alla fine del file di lettura)
jz?
NO
Syscall exit
salvo INIT
Tramite un puntatore ai 9 caretteri letti, ricavo INIT, RESET e RPM (utilizzo atoi4 per RPM, mentre atoi1 per gli altri)
END
SI
INIT = 0?
NO
call Sistema_Spento
salvo RESET
salvo RPM
call Controllo_Stato
MOD <- %eax
SI
LAST_INIT = 0?
NO
ALM <- 0
NUMB <- 0
call Contatore
call Stampa_Riga

Scelte progettuali

- Si è scelto di suddividere il programma in più file per aver maggiore chiarezza nel codice.

- Funzione *Divisione_Stampa* creata per evitare ridondanza di codice nella funzione *Stampa_Riga.*

- Implementazione di *Atoi1* e *Atoi4* per convertire direttamente 1 e 4 caratteri in valori numerici.

- Lettura e Scrittura eseguite riga per riga.

- Implementazione dello stato SPENTO (MOD = 00) dove segnaliamo il tempo di rimanenza nel caso in cui il motore rimanga spento, ma in circuito sia acceso.

- Implementazione del caso particolare che si verifica quando il precedente valore di **init** è 0.