UNIVERSITA’ DEGLI STUDI DI VERONA
Dipartimento di Informatica

Elaborato ASM

Laboratorio di Architettura degli elaboratori
A.A. 2015/2016

Alin Cristinel Sandu VR400082
Vladislav Bragoi VR397344

INDICE

PRESENTAZIONE ELABORATO 3
MAIN E VARIABILI GLOBALI 4
FUNZIONI E VARIABILI 5
MODALITÀ DI PASSAGGIO / RESTITUZIONE PARAMETRI 6
DIAGRAMMA DI FLUSSO 8
SCELTE PROGETTUALI 10
CODICE AD ALTO LIVELLO DEL PROGRAMMA 11

PRESENTAZIONE ELABORATO

Il presente elaborato sviluppa un programma Assembly che effettua il monitoraggio di un motore a combustione interna.
Il programma riceve in input il numero di giri/minuto del motore (RPM) e, impostate le soglie minima e massima per il funzionamento ottimale, fornisce in uscita una modalità di funzionamento del motore:

sotto-giri (SG), in regime ottimale (OPT) o fuori-giri (FG).

Inoltre si avrà in uscita:

1. da quanto tempo il sistema si trova nello stato attuale;
2. output di allarme che vale 1 se il sistema si trova in stato FG da piu' di 15 secondi (cicli di clock).

Il programma viene 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.

Es: ./run filenameInput.txt filenameOutput.txt

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

INIT, RESET, RPM

INIT: valore binario, quando vale 0 il programma restituisce una linea composta di soli 0; quando vale 1 il programma fornisce in uscita tutti i valori come da specifica.

RESET: valore binario, se posto a 1 il contatore dei secondi viene posto a zero.

RPM: valore del numero di giri ricevuto dal rilevatore (valore massimo 6500).

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

ALM, MOD, NUMB

MOD: indica in quale modalità di funzionamento si trova l'apparecchio al momento corrente (00 spento, 01 SG, 10 OPT, 11 FG);

NUMB: valore decimale, indica i secondi trascorsi nell' attuale modalità.

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

MAIN E VARIABILI GLOBALI

Variabili e costanti locali

- init_zero: costante di tipo string usata per scrivere direttamente nel file i valori “0,00,00\n” nel caso in cui il valore rilevato di INIT fosse 0.
- eseguito: costante di tipo string per la stampa del messaggio “Eseguito!” a video.
- eseguito_len: segue dalla precedente, che ne determina la lunghezza tramite un valore intero.
- INIT: variabile contenente il valore del segnale di INIT (inizializzata a 0);
- RESET: variabile contenente il valore del segnale di RESET (inizializzata a 0);

Nota: INIT e RESET, anche se contengono un valore binario, sono dichiarate di tipo “long” per evitare errori di sovrascritture dei registri “low” con cui si andrebbero ad eseguire le varie operazioni.

Variabili globali

- buff: puntatore alla zona di memoria allocata di dimensione 9, che accoglierà i valori letti dal file di input;
- buff_size: variabile contenente la dimensione del buffer inizializzata a 9 (lunghezza standard della stringa di lettura dal file di input);
- descrittore: variabile contenente il valore del descrittore del file di input;
- descrittore_write: variabile contenente il valore del descrittore del file di output;
- RPM: variabile contenente il valore dei giri del motore;
- ALM: variabile contenente il valore del segnale di allarme;
- MOD: variabile contenente il valore del segnale “modalità di funzionamento” del motore, rilevato in base al numero di giri dello stesso;
- NUMB: variabile contenente il valore dei secondi trascorsi (ogni stringa in input dal file corrisponde ad un secondo trascorso).

FUNZIONI E VARIABILI

Le funzioni utilizzate dal programma sono le seguenti:

File open.s:

- open_error: variabile di tipo string contenente il messaggio di errore causato da una mancata apertura del file;
- open_len: segue dalla precedente, che ne determina la lunghezza del messaggio.

File decodifica.s:

- error_message: variabile di tipo string contenente il messaggio di errore causato da un errato inserimento dei valori in input (riferiti precisamente ai segnali di INIT e RESET);
- error_len: variabile che contiene la lunghezza della stringa precedente.

File codifica.s:

tale funzione non contiene variabili dichiarate, bensì delle macro che eseguono in ordine:

- scrittura sul file di output del valore 0 (in ASCII 48);
- scrittura sul file di output del valore 1 (in ASCII 49);
- scrittura sul file di output della virgola (in ASCII, 44).

File atoi.s:

funzione leggermente modificata che converte una stringa delimitata dal carattere di "a capo" (valore 10 in ASCII) in intero:

- error: costante di tipo string contenente un messaggio di errore nel caso il valore degli RPM non sia di tipo decimale (viene salvato nel file di output);
- error_len: costante contenente la lunghezza in intero della stringa precedente;
- car: variabile di tipo byte utilizzata per catturare un carattere alla volta dalla stringa da convertire.

File itoa.s:

funzione modificata in modo che la scrittura venga fatta direttamente sul file di output:

- car: variabile di tipo byte utilizzata per convertire un numero alla volta in ASCII e salvarlo direttamente sul file di output. Termina salvando anche il carattere di a capo a fine stringa.

File mod_detect.s:

- mod_error: costante di tipo string contenente un messaggio di errore dovuto al superamento della soglia massima rilevata dal segnale di RPM (6500). Tale messaggio viene salvato direttamente nel file di output;
- mod_len: costante intera contenente la lunghezza della stringa precedente.

MODALITÀ DI PASSAGGIO / RESTITUZIONE PARAMETRI

Funzione **OPEN**:

Apre il file cui nome è contenuto in ebx con la modalità di apertura indicata nel valore contenuto in ecx. Restituisce il descrittore del file aperto nel registro eax, che verrà poi salvato nella corrispettiva variabile.

Funzione **DECODE**:

[Usata per decodificare il valore in ASCII del segnale INIT e RESET.

Riceve nel registro esi (source index) il puntatore alla stringa da decodificare (vale solo per valori booleani, 0 o 1 presi singolarmente). Ritorna un messaggio di errore se per due cicli consecutivi non ha trovato alcun valore booleano, altrimenti ne ritorna il primo identificato tramite registro edx.

Funzione **CODIFICA**:

Analizza le variabili globali ALM, MOD e NUMB per codificarne il relativo valore in ASCII e successivamente scriverlo direttamente nel file di output. Scrive inoltre due virgole (#44 in ASCII), rispettivamente dopo il valore di ALM e MOD.

Nota: anche "descrittore_write" è una variabile globale usata per identificare il file in cui scrivere i valori. Inoltre, tale funzione presenta le macro anticipate precedentemente e utilizzate per semplificare la lettura e scrittura del codice.

Funzione **ATOI**:

[Usata per decodificare il valore in ASCII del segnale RPM.

Converte una stringa di caratteri, delimitata dal carattere di invio (new line, #10 in ASCII), e puntata dal valore contenuto nel registro eax. Il numero intero convertito inoltre, viene anch'esso ritornato alla funzione chiamante tramite lo stesso registro eax e la funzione si preoccupa anche di riportare un messaggio di errore nel file di output se il valore degli RPM presenta caratteri non associati a valori decimali.

Funzione **ITOA**:

Converte un numero intero, passato alla funzione tramite il registro eax, in una stringa che viene salvata – carattere per carattere - direttamente nel file di output. Inoltre, alla fine della codifica del valore in ASCII, si preoccupa anche di inserire il carattere di invio (a capo) alla fine della sua esecuzione.

Nota: anche questa funzione utilizza il “descrittore_write”, variabile di tipo globale, per identificare il file su cui scrivere.

Funzione **ATOI**:

Converte una stringa di caratteri ricevuta tramite puntatore nel registro eax (delimitata dal carattere di invio - #10 in ASCII) in un numero, che viene restituito sempre tramite lo stesso registro. Controlla inoltre che la stringa in input contenga solo caratteri relativi a valori decimali, in caso contrario scrive sul file un messaggio di errore e l’esecuzione del programma riprende dalla stringa successiva nel file di input.

Funzione **MOD_DETECT**:

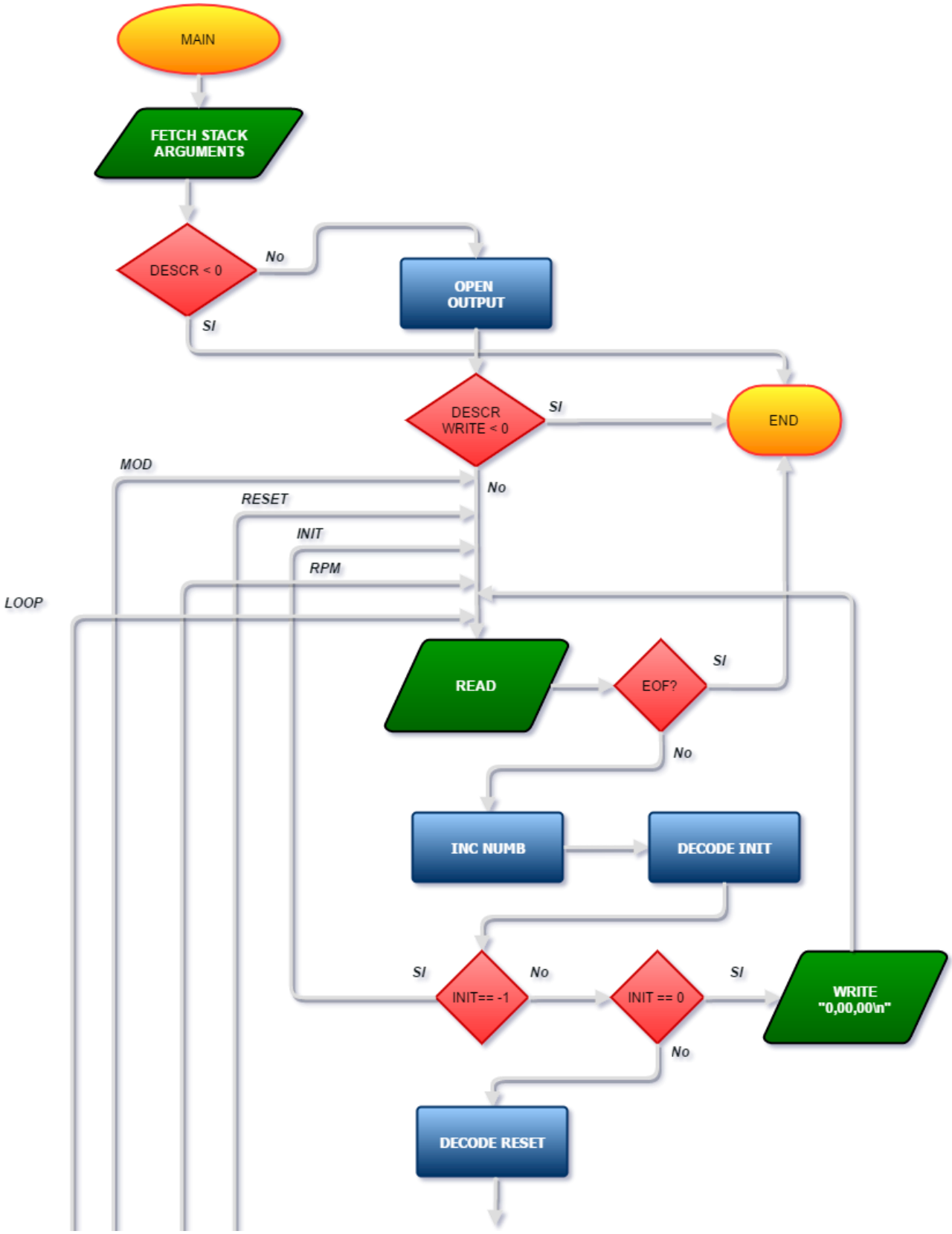
Analizza la variabile globale RPM e salva nella variabile globale MOD il relativo stato di funzionamento del motore in base alle soglie definite dalla consegna del progetto:

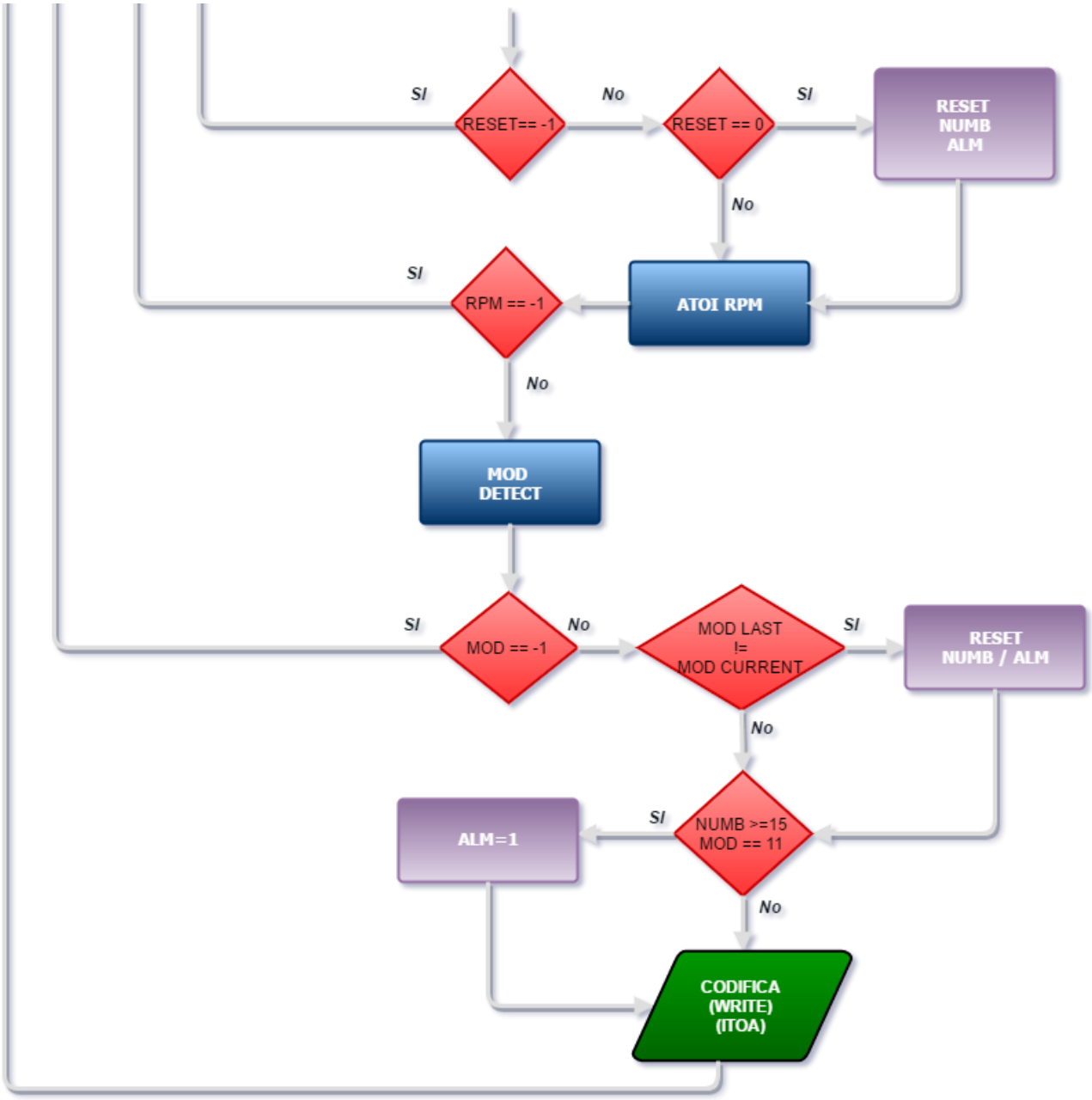
RPM < 2000 → SG

2000 ≤ RPM ≤ 4000 → OPT

RPM > 4000 → FG, con soglia massima di 6500.

DIAGRAMMA DI FLUSSO




SI
RESET== -1
No
RESET == 0
SI
RESET
NUMB
ALM
No
SI
RPM == -1
ATOI RPM
No
MOD
DETECT
SI
MOD == -1
No
MOD LAST
!=
MOD CURRENT
SI
RESET
NUMB / ALM
No
ALM=1
SI
NUMB >=15
MOD == 11
No
CODIFICA
(WRITE)
(ITOA)

SCELTE PROGETTUALI

Per realizzare questo elaborato in Assembly, in fase di progettazione abbiamo scelto:

1. di leggere il file riga per riga, attraverso un buffer di dimensione 9, dichiarato globalmente in modo tale da essere visibile a tutte le funzioni;
2. di salvare i vari descrittori del file di input e quello di output in apposite variabili, anch'esse visibili globalmente;
3. di decodificare metà buffer, ovvero la parte contenente i valori di INIT e RESET tramite la funzione DECODE, e la parte contenente i valori di RPM tramite la funzione ATOI:
   Nota: la funzione DECODE converte solo caratteri associati a valori booleani (uno alla volta), mentre la ATOI converte un'intera stringa di caratteri contenente valori interi.
4. di identificare il nuovo valore di MOD e confrontarlo con il vecchio per verificare un eventuale cambiamento di stato, e dunque un reset dei valori di NUMB e ALM.
   tale operazione viene eseguita utilizzando il registro eax come "appoggio" temporaneo.
5. di codificare ciascun valore di ALM e MOD tramite la funzione CODIFCA e scriverne i valori direttamente all'interno del file di output. La stessa funzione aggiunge due virgole e inoltre richiama la funzione ITOA per convertire in ASCII anche il valore di NUMB, che anch'esso verrà scritto direttamente sul file, assieme ad un ultimo carattere di invio a fine riga.

Inoltre, per una maggiore stabilità del codice sono state previste le seguenti verifiche e controlli:

1. La funzione OPEN, oltre ad aprire il file passatole in input si preoccupa anche di restituire un messaggio di errore a video se l'operazione non è stata eseguita con successo. Tale messaggio provoca anche la chiusura diretta del programma.
2. La funzione DECODE, se non riconosce per due cicli consecutivi un carattere contenente un valore "zero" oppure "uno", restituisce un messaggio di errore, che viene salvato sul file di output (alla riga corrispondente) e riprende l'esecuzione del programma caricando nel buffer la riga successiva.
3. La funzione ATOI prevede anch'essa un messaggio di errore, che viene scritto sul file di output se gli RPM sono stati riconosciuti come valore NON decimale. Anche in questo caso l'esecuzione riprende alla riga successiva dal file di input.
4. Infine, anche la funzione MOD si preoccupa di verificare la congruità dei valori inseriti in input. Se gli RPM superano la soglia massima impostata a 6500 giri, viene scritto sul file di output il messaggio di errore relativo al superamento di tale soglia.

Il programma termina quando la funzione READ restituisce nel registro eax il valore 0 (associabile alla costante EOF oppure NULL in C), che identifica la completa lettura del file. Da questo segue un ultimo messaggio di avvenuta esecuzione del programma.

CODICE AD ALTO LIVELLO DEL PROGRAMMA

```
#include <stdio.h>
#include <stdlib.h>

/**********VARIABILI GLOBALI***********/
FILE *fd_input, *fd_output;
char buff[9], *s, init_zero[]="0,00,00\n";
int RPM, ALM, NUMB, MOD;

/*********FIRME DELLE FUNZIONI**********/
int DECODE(char *dec);
int MOD_DETECT(int RPM);
void CODIFICA();

/*************** MAIN *****************/
int main (int argc,char **argv){

    int INIT = 0, RESET = 0;
    char tmp;

    //apertura file di lettura
    fd_input = fopen(argv[1], "r");

    if(fd_input == NULL){ //controllo apertura file
        perror("Errore di apertura del file\n");
        return(-1);
    }

    //apertura file di scrittura
    fd_output = fopen(argv[2], "w");

    if(fd_input == NULL){ //controllo apertura file
        perror("Errore di apertura del file\n");
        return(-1);
    }
    //READ = legge la riga e la mette nella stringa buffers
    inizio:
    while (fgets(buff, 9, fd_input)!=NULL) {
        NUMB++; //incrementa i secondi
        s = buff;
        /* ==============  IDENTIFICO IL VALORE DI INIT   ============== */
        INIT = DECODE(s);

        if(INIT == 0)
            goto write;         //scrive direttamente la stringa init_zero nel file
        else if(INIT == -1)
            continue;           //salta alla riga successiva del file di input

        /* ==============  IDENTIFICO IL VALORE DI RESET  ============== */
        RESET = DECODE(++s);

        if (RESET == 1) {
            NUMB = 0;             //resetto i secondi e l'allarme
            ALM = 0;
        }else if(RESET == -1)
            continue;           //salta alla riga successiva del file di input
```

```
        /* ============== IDENTIFICO IL VALORE DEGLI RPM  ============== */
        s+=3; //incremento del puntatore per considerare gli rpm
        RPM = atoi(s);

        if(MOD != MOD_DETECT(RPM)){
            NUMB = 0;             //resetto i secondi e l'allarme
            ALM = 0;
        }
        MOD = MOD_DETECT(RPM);    //aggiorno il valore di MOD

        if(MOD == -1)
            continue;           //salta alla riga successiva del file di input

        if (NUMB >= 15 && MOD == 11)
            ALM = 1; //attiva l'allarme

        // CODIFICA I VALORI E LI SCRIVE NEL FILE DI OUTPUT
        CODIFICA();
        tmp= fgetc(fd_input);//scarto il carattere di invio della stringa
    }

    // CHIUSURA DEI FILE
    fclose(fd_input);
    fclose(fd_output);

    printf("Eseguito!\n");
    return 0;
    write:
        fprintf(fd_output, "%s", init_zero);
        tmp = fgetc(fd_input);
        goto inizio;
}

/***** METODI DELLE FUNZIONI ******/
int DECODE(char *s){
    int counter = 0;

start:  switch (*s++) {
            case '0':
                return 0;
            case '1':
                return 1;
            case ',':
                if (counter < 1){ //controllo se per due cicli consecutivi non siano
                    counter++;    //stati trovati valori binari
                    goto start;
                }
            default:
                fprintf(fd_output, "%s", "Errore!\n");
                return -1;
        }
}
```

```
int MOD_DETECT(int RPM){

    if (RPM < 2000) {
        return 1;
    }else if (RPM >= 2000 && RPM <= 4000) {
        return 10;
    }else if (RPM > 4000 && RPM <= 6500) {
        return 11;
    }else
        return -1;
}

void CODIFICA(){

    /* ============== SCRIVO ALM SUL FILE  ============== */
    if(ALM == 0)
        fputc('0', fd_output);
    else if (ALM == 1)
        fputc('1', fd_output);

    fputc(',', fd_output);//scrivo una virgola

    /* ============== SCRIVO MOD SUL FILE  ============== */
    if (MOD == 1) {
        fputc('0', fd_output); fputc('1', fd_output);
    }else if (MOD == 10){
        fputc('1', fd_output); fputc('0', fd_output);
    }else if (MOD == 11) {
        fputc('1', fd_output); fputc('1', fd_output);
    }

    fputc(',', fd_output);//scrivo una virgola

    /* ============== SCRIVO NUMB SUL FILE  ============== */
    if (NUMB < 10)
        fputc('0', fd_output);
    fprintf(fd_output, "%d", NUMB);

    fputc('\n', fd_output);; //scrivo il carattere di a capo a fine riga
}
```