UNIVERSITÀ
di VERONA

ELABORATO ASM

LABORATORIO DI ARCHITETTURA DEGLI ELABORATORI
2022/2023

**“Gestione del menù cruscotto
di un’automobile”**

COMPONENTI DEL GRUPPO:
CERIANI ALEX VR403780
TERZIU FABIO VR471449

INDICE:

1 INTRODUZIONE E SCOPO:

1.1 Funzionamento

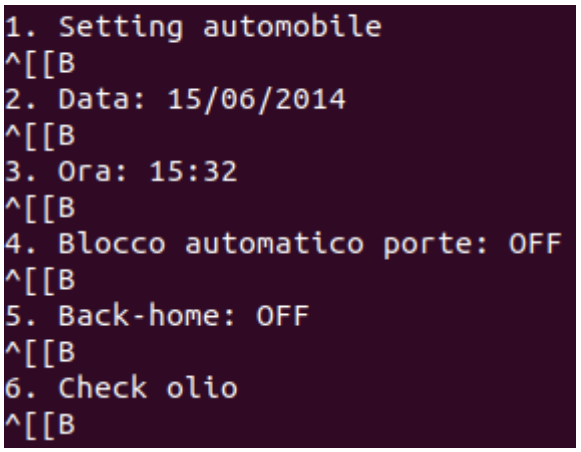
L' elaborato consiste nella realizzazione di un programma assembly per la gestione del menù cruscotto di un'automobile.

Il menù dovrà permettere la visualizzazione di 5 voci se acceduto in modalità utente:
- Data;
- Ora;
- Impostazione blocco automatico porte;
- Back-home;
- Check olio;

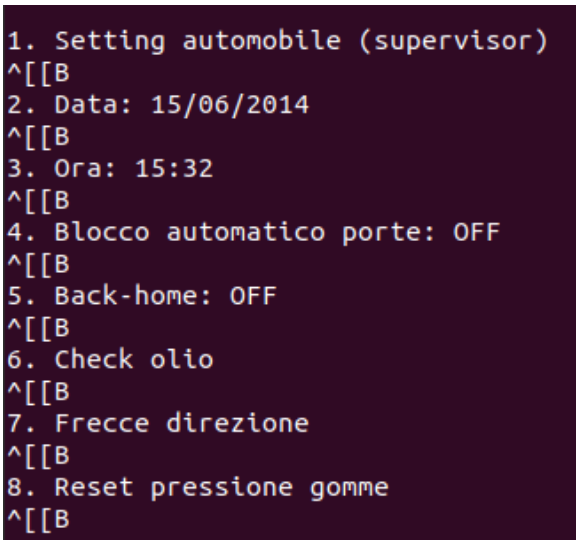
In modalità 'supervisor' invece oltre alle precedenti, ne verranno visualizzate altre due:
- Lampeggi frecce modalità autostrada;
- Reset pressione gomme;

Per avviare il programma:
- Supervisor: lanciando da riga di comando il nome dell'eseguibile seguito dal codice '2244'.
- Utente: lanciando solo il nome dell'eseguibile da riga di comando, l'importante è che non sia seguito dalla sequenza '2244'.

UTENTE:



SUPERVISOR:



È possibile scorrere il menù verso il basso oppure verso l'alto, premendo rispettivamente i tasti 'freccia giù+invio' e 'freccia su+invio'.
E' possibile passare dal primo all'ultimo punto premendo la freccia su, mentre dall'ultimo al primo premendo invece la freccia giù.

Le voci del menù 4, 5, 7, 8 hanno la possibilità di visualizzare anche il corrispettivo sottomenù, per permettere la visualizzazione dello stato attuale del setting e anche l'opzione per modificarlo.
Per accedere al sottomenu, innanzitutto bisogna trovarsi sulla voce del menù desiderata, e successivamente premere il comando 'freccia destra+invio'.

```
4. Blocco automatico porte: OFF
^[[C

OFF
^[[B

ON

4. Blocco automatico porte: ON
```

I sottomenù 4 "Blocco automatico porte" e 5 "Back-home", hanno la possibilità di impostazione ON/OFF dove:
una volta acceduto al sottomenù, si visualizzerà lo stato attuale del setting, e tramite i tasti 'freccia su+invio' o 'freccia giù+invio' si permette di modificare l'impostazione da ON a OFF e viceversa.

```
7. Frecce direzione
^[[C
Numero di freccia attuale: 3
1
Numero di freccia attuale: 2
6
Numero di freccia attuale: 5
4
Numero di freccia attuale: 4
50
Numero di freccia attuale: 5

7. Frecce direzione
^[[C
Numero di freccia attuale: 5
```

Il sottomenù 7 "Frecce direzione", permette la visualizzazione del numero dei lampeggi in modalità autostrada.
Questo numero è di default '3', con possibilità di variazione da un minimo di 2 ad un massimo di 5.
I valori fuori range assumeranno il valore del setting massimo/minimo.

Infine il sottomenù 8 "Reset pressione gomme", restituirà il messaggio "Pressione gomme resettata" e in automatico tornerà nel menù principale.

Il ritorno dal sottomenù al menù sarà permesso soltanto premendo il comando 'invio'.

1.2 Assembly

Questo elaborato ha come base il codice Assembly, prima di continuare l'esposizione è opportuno soffermarci per spiegare brevemente di cosa si tratta.

Il linguaggio Assembly è un linguaggio di programmazione di basso livello molto vicino al linguaggio macchina (linguaggio binario), si differenzia da alcune caratteristiche che lo rendono più agevole da scrivere e leggere per l'uomo (linguaggio mnemonico).
Esso è composto da microistruzioni che permettono di operare direttamente sui registri della CPU, inoltre permette di utilizzare liberamente lo stack.
I file assembly, non necessariamente richiedono l'estensione .s.
E' buona norma però aggiungerla, in quanto alcuni compilatori come il gcc leggendo l'estensione .s, assumono che il file contenga codice Assembly.

Assembly, rispetto ad altri linguaggi di alto livello come il C, può avere una maggiore efficienza del codice.
I programmi Assembly, una volta compilati, risultano più piccoli (memoria) e più veloci rispetto agli stessi scritti con codice di alto livello, nonostante i primi contengano un maggior numero di righe di codice.
Inoltre, questa programmazione, a volte è indispensabile per la scrittura di driver per hardware specifici.
Ci sono ovviamente vantaggi e svantaggi, un vantaggio è appunto quello della sua efficienza, mentre uno svantaggio può essere il fatto che sia più facile introdurre bug in quanto la programmazione risulta più complessa.

2 CODICE C:

2.1 Intro e commento codice

Implementazione del codice C:

Innanzitutto bisogna dichiararare le librerie, le variabili globali e le funzioni utilizzate.

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

//variabili globali
int indice=1;      //valore iniziale dello switch
int valore[5];     //variabile di inserimento da tastiera
int blocco=0;      //variabile per capire stato attuale on off
int blocco2=0;     //variabile per capire stato attuale on off
int s=0;           //variabile per capire se si è entrati o meno nel sottomenu
int dir=3;         //variabile lameggio frecce (default 3)
int garbage;       //variabile dove finiscono i valori richiesti se fuori range

//funzioni
void user(void);           //funzione utente
void supervisor(void);     //funzione supervisor solo con 2244
int sottomenu(void);       //finzione ingresso sottomenu
void freccedir(void);      //funzione sottomenu frecce direzione
void bloccoporte(void);    //funzione psottomenu bloccoporte
void backhome(void);       //funzione sottomenu backhome
```

Al main vengono passati due argomenti: argc e argv.

- argc contiene il numero di elementi in argv.
- argv è un array di stringhe contenente gli argomenti del programma:
  - ➔ argv[0]: nome del programma.
  - ➔ argv[1]: primo parametro.

Viene fatto un if dove: se da riga di comando viene inserita la stringa 2244 viene richiamata la funzione 'supervisor()', altrimenti viene richiamata la funzione 'utente'.

```
//main
//argc numeri parametri sulla riga di comando  -  argv array di puntatore a char
int main(int argc, char *argv[]){

    if(argc==2 && atoi(argv[1])==2244){      //solo se il secondo argomento è 2244 entra in supervisor
        supervisor();
    }
    else{
        user();
    }

    return 0;
}
```

La differenza tra queste due modalità di accesso come spiegato in precedenza è che in supervisor si ha la possibilità di visualizzare più voci e permettere l'impostazione di alcune di esse.
Andiamo a vederla più nello specifico:

```
//funzione supervisor
void supervisor(){
do{
switch(indice){

    case 1:
        printf("1. Setting automobile (supervisor):\n");
        break;

    case 2:
        printf("2. Data: 15/06/2014\n");
        break;

    case 3:
        printf("3. Ora: 15:32\n");
        break;

    case 4:
        if(blocco==1){
        printf("4. Blocco automatico porte: ON\n");
        }
        else if (blocco==0){
        printf("4. Blocco automatico porte: OFF\n");
        }
        s=sottomenu();
        if(s==1){
        bloccoporte();
        }
        break;

    case 5:
        if(blocco2==1){
        printf("5. Back-home: ON\n");
        }
        else if (blocco2==0){
        printf("5. Back-home: OFF\n");
        }
        s=sottomenu();
        if(s==1){
        backhome();
        }
        break;

    case 6:
        printf("6. Check olio\n");
        break;

    case 7:
        printf("7. Frecce direzione\n");
        s=sottomenu();
        if(s==1){
        freccedir();
        }
        break;

    case 8:
        printf("8. Reset pressione gomme\n");
        s=sottomenu();
        if(s==1){
        printf("Pressione gomme resettata\n");}
        break;
    default:
}
```

Un ciclo ‘do while’ che racchiude l’intero contenuto della funzione, cominciando prima dello switch mostrato nell’immagine precedente e concludendosi alla fine della funzione come mostrato nella figura successiva. La condizione per uscire da questo ciclo e quindi chiudere il programma è ‘freccia sinistra+invio'.

Lo switch case comprende le varie voci del menù, da 1 a 8 in questo caso (supervisor):

- nei casi: 1,2,3 e 6 viene stampato a video il nome della rispettiva voce.
- nel caso 4, come il 5, necessitiamo di una condizione tramite la variabile ‘blocco’ per aggiungere al nome del menu se lo stato attuale è ‘ON’ oppure ‘OFF’.
  Successivamente viene chiamata la funzione sottomenù, che permette di inserire il comando ‘freccia destra+invio’ se si desidera accedere al sottomenù.
  Se si, la variabile assumerà il valore ‘1’ e si accederà alla funzione ‘bloccoporte()’ per il caso 4 e ‘backhome’ per il 5.

Questa linea da seguire vale anche per il caso 7.
Per l’ultimo e ottavo invece, dopo essere entrati nel sottomenu,verrà visualizzato a video il messaggio ‘Pressione gomme resettata’ ed in automatico si tornerà al menù.

Osservando ora l’immagine successiva, si fa un controllo per verificare se si è tornati da un caso contentente il sottomenù, per evitare di fare due volte la richiesta di input, che sarà gestita da un controllo di superamento dimensioni dell’array ‘valore’.
Successivamente si controlla se il valore inserito da tastiera è una freccia in alto (ASCII 65) o una freccia in basso (ASCII 66), seguite da un invio (ASCII 10), in modo da poter scorrere il menù.

```
int i=0;
   if(indice ==4 || indice==5 || indice==7 || indice==8){
   goto fine;
   }

while((valore[i]=getchar())!='\n' && i<4){
      i++;
}

if(valore[4]!=10 && i==4){
        while((garbage=getchar())!='\n');
}



fine:
if (valore[0]==27 && valore[1]==91 && valore[2]==66 && valore[3]==10){
    if(indice==8){
        indice=1;
    }
    else{
        indice ++;
    }
}

else if(valore[0]==27 && valore[1]==91 && valore[2]==65 && valore[3]==10){
    if(indice==1){
        indice=8;
    }
    else{
        indice--;
    }
}

}while(valore[2]!=68);
}
```

La funzione ‘freccedir()’ gestisce il sottomenù dell’indice 7:

```
flag=0;
if(valore[4]!=10 && i==4){
        while((garbage=getchar())!='\n');
        flag=1;
}

if(valore[1]==10 && valore[0]!=10){  //se 1 cifra

        if(valore[0]>=48 && valore[0]<=57){ //se numero
                if(valore[0]<=50){  //se 2 o minore stampa 2
                dir=2;
                }
                else if(valore[0]==51){  //se 3 stampa 3
                dir=3;
                }
                if(valore[0]==52){  //se 4 stampa 4
                dir=4;
                }
                if(valore[0]>=53){  //se meggiore uguale a 5 stampa 5
                dir=5;
                }
        }
        else{ //se non numero
                dir=dir;
                }
printf("Numero di freccia attuale: %i\n",dir);
        }

else if(flag==0 && valore[0]!=10){   //se piu di una cifra
        for(i=0;i<4;i++){  //controllo a 4 cifre

                if(valore[i]<48 || valore[i]>57 ){ //se non è un numero
                dir=dir;
                break;
                }
                else{
                dir=5;
                }
        }

        printf("Numero di freccia attuale: %i\n",dir);
}

else if(valore[0]!=10){
printf("Numero di freccia attuale: %i\n",dir);
}
```

Dove innanzitutto viene stampato di default il valore ‘3’ dei lampeggi.
Viene fatta la richiesta di inserimento con controllo che l’input inserito sia corretto.
Se dovesse superare la dimensione massima dell’array, viene posto il contenuto delle dimensioni eccessive in una variabile ‘garbage’.

Si prosegue con la verifica:

- se l’input contiene un solo carattere ed è un numero:
  - ➔ stampa 2 se premuto il tasto ‘2’ o minore.
  - ➔ stampa 3 se premuto il tasto ‘3’.
  - ➔ stampa 4 se premuto il tasto ‘4’.
  - ➔ stampa 5 se premuto il tasto ‘5’ o maggiore.

altrimenti se contiene un carattere ma non è un numero stampa lo stato precedente.

- se l’input contiene più di una carattere:
  - ➔ ciclo for che valuta se è un numero, stampa il valore massimo, altrimenti stampa lo stato precedente.

I numeri devono essere numeri interi positivi che non superino le 4 cifre, altrimenti si visualizzerà lo stato precedente del setting.
Si potrà uscire dal sottomenù soltanto premendo il pulsante ‘invio’.

Come ultima funzione, c'è quella richiamata se ci si trova nel caso 4 (analogo al caso 5):

```
if (valore[0]==27 && valore[1]==91 && valore[2]==66 && valore[3]==10){
if(blocco==1){
printf("OFF\n");
blocco=0;
}
else if(blocco==0){
printf("ON\n");
blocco=1;
}
}
else if(valore[0]==27 && valore[1]==91 && valore[2]==65 && valore[3]==10){
        if(blocco==1){
        printf("OFF\n");
        blocco=0;
        }
        else if(blocco==0){
        printf("ON\n");
        blocco=1;
        }
}
else if(valore[0]==10){
    v=1;
    valore[2]=0;
}
else{
if(blocco==1){
printf("ON\n");
}
else if(blocco==0){
printf("OFF\n");
}
}
```

Tramite la variabile globale 'blocco', che definisce lo stato attuale del setting, stampo se=1 'ON' oppure se =0 'OFF'.

Dentro ad un 'do while':

- Viene fatta la richiesta di inserimento input, con i controlli sulla dimensione dell'array.
- Se l'input è 'freccia in su+invio' oppure 'freccia giù+invio' e lo stato attuale è 'ON', allora stampa 'OFF' e la variabile 'blocco' che appunto definisce lo stato attuale va a 0 (OFF).
- Altrimenti se lo stato attuale è 'OFF', allora stampa 'ON' e la variabile 'blocco' assume valore 1 (ON).
- Se invece viene ricevuto in ingresso soltanto 'invio' esco dal sottomenù per tornare al menù.
- Altrimenti se ricevuto un qualsiasi altro input stampo soltanto lo stato precedente del setting.

3 ASSEMBLY & PSEUDOCODICE:

3.1 Makefile

Il progetto è strutturato come segue:

- Makefile: tramite il comando ‘make’ digitato direttamente nella shell è possibile compilare i file sorgenti e linkare i relativi file oggetto.
  L’output è un singolo file eseguibile.
  Il Makefile contiene tutte le dipendenze per la compilazione e il linking.

- src: contiene i file sorgente, ovvero i file in codice assembly, oltre al file contentente il codice C.

- obj: contiene i file oggetto creati dal Makefile.

- bin: contiene il file eseguibile finale.

Andiamo ora a descrivere il Makefile creato:

```
BIN= bin/menu
AS_FLAGS= as --32 -gstabs
LD_FLAGS= ld -m elf_i386
OBJ= obj/main.o obj/menu.o obj/sottomenu.o obj/rileva_direzione.o obj/rileva_numero.o

all: $(OBJ)
        $(LD_FLAGS) $(OBJ) -o $(BIN)

obj/main.o: src/main.s
        $(AS_FLAGS) src/main.s -o obj/main.o

obj/menu.o: src/menu.s
        $(AS_FLAGS) src/menu.s -o obj/menu.o

obj/sottomenu.o: src/sottomenu.s
        $(AS_FLAGS) src/sottomenu.s -o obj/sottomenu.o

obj/rileva_direzione.o: src/rileva_direzione.s
        $(AS_FLAGS) src/rileva_direzione.s -o obj/rileva_direzione.o

obj/rileva_numero.o: src/rileva_numero.s
        $(AS_FLAGS) src/rileva_numero.s -o obj/rileva_numero.o

.PHONY:clean
clean:
        rm -f $(OBJ) $(BIN)
```

Variabili:

- BIN: destinazione dell'eseguibile (path/nomefile).

- AS_FLAGS: trasforma il file sorgente .s in file oggetto .o
  'as –32 -gstabs' dove:
  - as:trasforma il sorgente in file oggetto facendo un controllo sulla sintassi, questa trasformazione serve a rendere il codice leggibile dalla macchina.
  - '32' sta per la compilazione a 32 bit.
  - 'gstabs' genera una serie di informazioni che servono al debugger.

- LD_FLAGS: esegue il link dei file oggetto.
  L'opzione '-m elf_i386' serve a fare il link con librerie a 32 bit (librerie OS).

- OBJ: sono i file oggetto da creare

Il makefile è composto da un insieme di target e di regole corrispondenti strutturati nella seguente maniera:

- Target : lista dei file da analizzare
- Regola

Se la lista dei file da analizzare non ha subito modifiche, il target non viene eseguito.

Il suo obiettivo è la creazione del file eseguibile senza dover scrivere tutti i comandi necessari per la compilazione e il linking.
Digitando il comando 'make' vengono eseguite le operazioni descritte nell'etichetta 'all', per eseguire altre etichette è necessario specificare il nome, per esempio 'make clean'.

```
fabione@fabione-pc:~/Scrivania/asm$ make clean
rm -f obj/main.o obj/menu.o obj/sottomenu.o obj/rileva_direzione.o obj/rileva_nu
mero.o bin/menu
fabione@fabione-pc:~/Scrivania/asm$ make
as --32 -gstabs  src/main.s -o obj/main.o
as --32 -gstabs  src/menu.s -o obj/menu.o
as --32 -gstabs  src/sottomenu.s -o obj/sottomenu.o
as --32 -gstabs  src/rileva_direzione.s -o obj/rileva_direzione.o
as --32 -gstabs  src/rileva_numero.s -o obj/rileva_numero.o
ld -m elf_i386 obj/main.o obj/menu.o obj/sottomenu.o obj/rileva_direzione.o obj/
rileva_numero.o -o bin/menu
```

Tramite il comando 'bin/menu' lanciamo l'eseguibile.

3.2 Variabili

Nella cartella ‘src’ sono contenuti cinque file .s, contenenti il codice sorgente Assembly. Andiamo ora a descrivere le variabili utilizzate nel codice:

main_asm.s

user	long inizializzato a 0, possibili valori 0|1 - 0 indica la modalità utente - 1 supervisor
posizione_menu	long inizializzato a 1, possibili valori 1|2|3|4|5|6|7|8 indica la posizione attuale nel menu, serve per sapere quale stringa stampare a terminale viene utilizzata sia nel menù che nel sottomenù
max_posizione	long inizializzato a 6, possibili valori 6|8 indica il valore massimo della variabile posizione_menu, dipende dal valore di user: - se user= 0, allora 6 - se user= 1, allora 8 viene confrontata con posizione_menu in caso di incremento di quest'ultima
sottomenu	long inizializzato a 0, possibili valori 0|1 se valorizzata a 1 indica che ci troviamo in un sottomenù
porte	long inizializzato a 0, possibili valori 0|1 - 0 indica che il blocco automatico porte è OFF - 1 indica che il blocco automatico porte è ON
home	long inizializzato a 0, possibili valori 0|1 - 0 indica che il back home è OFF - 1 indica che il back home è ON
frecce	long inizializzato a 3, possibili valori 2|3|4|5 indica il valore attuale delle frecce direzione

rileva_direzione.s

input_freccia	ascii inizializzato a "00000" utilizzato per salvare la stringa inserita dall'utente nel terminale - i primi tre byte servono per memorizzare la freccia - il quarto per verificare che sia terminata la stringa (carattere invio) - il quinto per verificare che non ci siano altri caratteri nel buffer

menu.s
Sono presenti solo variabili di stampa con relativa lunghezza

supervisor	“1. Setting automobile (supervisor)”
supervisor_len	lunghezza di supervisor
utente	“1. Setting automobile:”
utente_len	lunghezza di utente
data	“2. Data: 15/06/2014”
data_len	lunghezza di data
ora	“3. Ora: 15:32”
ora_len	lunghezza di ora
Blocco_automatico_porte_on	"4. Blocco automatico porte: ON"
Blocco_automatico_porte_on_len	lunghezza di Blocco_automatico_porte_on
Blocco_automatico_porte_off	"4. Blocco automatico porte: OFF"
Blocco_automatico_porte_off_len	lunghezza di Blocco_automatico_porte_off
back_home_on	"5. Back-home: ON"
back_home_on_len	lunghezza di back_home_on
back_home_off	"5. Back-home: OFF"
back_home_off_len	lunghezza di back_home_off
check_olio	"6. Check olio"
check_olio_len	lunghezza di check_olio

frecce_direzione	"7. Frecce direzione"
frecce_direzione_len	lunghezza di frecce_direzione
Reset_pressione_gomme	"8. Reset pressione gomme"
Reset_pressione_gomme_len	lunghezza di Reset_pressione_gomme

sottomenu.s
Sono presenti solo variabili di stampa con relativa lunghezza

ON	"ON"
ON_len	lunghezza di ON
OFF	"OFF"
OFF_len	lunghezza di OFF
pressione_gomme	"Pressione gomme resettata"
pressione_gomme_len	lunghezza di pressione_gomme
valore2	"Numero di freccia attuale: 2"
valore2_len	lunghezza di valore2
valore3	"Numero di freccia attuale: 3"
valore3_len	lunghezza di valore3
valore4	"Numero di freccia attuale: 4"
valore4_len	lunghezza di valore4
valore5	"Numero di freccia attuale: 5"
valore5_len	lunghezza di valore5

rileva_numero.s

input_numero	memorizza ‘input_numero’

3.3 Fase 1 (controllo parametro)

```
_start:
                        #controllo se è stato passato un parametro, se = 2244 sono supervisor
   xorl %eax, %eax      #pulisco registro eax
   movl (%esp), %eax    #metto argc in eax
   cmpl $2, %eax        #se passo un parametro, argc = 2
   je verificacodice
   jmp inizializzazione


verificacodice:
   xorl %edx, %edx
   xorl %eax, %eax

   movl 8(%esp), %eax       #in questa zona di memoria trovo il parametro passato,
                            #(%esp) contiene argc(numero di valori nel vettore argv), 4(%esp) contiene il primo valore di argv

   movb (%eax,%edx), %cl    #edx è lo spiazzamento, adesso è = 0
   testb %cl, %cl           #controllo se ci sono altri caratteri
   jz inizializzazione      #se risultato di test è zero esco dal controllo
   cmpb $50, %cl            #comparo il 2 (50 in ascii) con cl
   jne inizializzazione

   incl %edx
```

Verifico se mi è stato passato un parametro, se ho un parametro salto in verificacodice (etichetta), altrimenti salto a inizializzazione (fase 2, inizializzazione).

Verificacodice: recupero dallo stack il parametro passato e inizio a scorrere i valori, se la sequenza non è 2244 (mostrato in figura solo il primo valore) salto a inizializzazione, altrimenti se arrivo a fine sequenza cambio il valore di ‘user’ a 1 e il valore di ‘maxposizione’ a 8, poi salto a inizializzazione.

3.4 Fase 2 (inizializzazione)

```
inizializzazione:             #carico tutte le variabili che serviranno alle funzioni chiamate,
                              #non carico i valori ma i relativi indirizzi di memoria
   leal user, %eax            # esp+20
   pushl %eax
   leal frecce, %eax          # esp+16
   pushl %eax
   leal home, %eax            # esp+12
   pushl %eax
   leal porte, %eax           # esp+8
   pushl %eax
   leal posizione_menu, %eax  # esp+4
   pushl %eax
```

Carico nello stack tutte le variabili:
user, frecce, home, porte, posizione menu.

Successivamente salto a inizio_menu(etichetta) e valuto la variabile sottomenu:

- se è 1 continuo con la fase 3(Sottomenu)
- se è 0 continuo con la fase 4 (Stampa menu)

3.5 Fase 3 (sottomenu)

```
inizio_sottomenu:
   call print_sottomenu
   leal posizione_menu, %edi
   cmpl $8, (%edi)            #se sono in posizione 8 devo solo stampare una stringa senza rilevare un input
   je invio


   leal posizione_menu, %edi            #
   cmpl $4, (%edi)
   je cambiaporta
   cmpl $5, (%edi)
   je cambiahome
   cmpl $7, (%edi)
   je cambiafreccia
   jmp invio
```

inizio_sottomenu: chiamo la funzione print_sottomenu (sottomenu.s)
print_sottomenu: recupero la posizione del menù (sempre dallo stack):

- se è 4 recupero la variabile porte(stack) e stampo il valore attuale(on|off)
- se è 5 recupero la variabile home(stack) e stampo il valore attuale(on|off)
- se è 7 recupero la variabile frecce(stack) e stampo il valore attuale(2|3|4|5)
- se è 8 stampo la stringa "pressione gomme resettata".

Dopo aver stampato torno al main.

- Se ‘posizione_menu’ è ‘7’, chiamo la funzione rileva_numero (fase 7, rileva_numero).
- In caso contrario chiamo la funzione rileva_direzione (fase 5, rileva input).

3.6 Fase 4 (stampa)

Chiamo la funzione print_menu: recupero dallo stack la posizione (posizione_menu), in base alla posizione stampo la relativa stringa.

```
print_menu:

    movl 4(%esp), %eax

    cmpl $1, (%eax)
    je posizione1
    cmpl $2, (%eax)
    je posizione2
    cmpl $3, (%eax)
    je posizione3
    cmpl $4, (%eax)
    je posizione4
    cmpl $5, (%eax)
    je posizione5
    cmpl $6, (%eax)
    je posizione6
    cmpl $7, (%eax)
    je posizione7
    cmpl $8, (%eax)
    je posizione8
```

```
posizione1:
movl 20(%esp), %eax          #modalita utente o supervisor

cmpl $1, (%eax)
je modalità_supervisor      #se 1 entro nella modalita supervisore
movl $4, %eax
movl $1, %ebx
leal utente, %ecx
movl utente_len, %edx
int $0x80
jmp fine
```

- Se sono in posizione 1
  recupero dallo stack la variabile user, in quanto la stringa da stampare dipende dal valore di quest'ultima.
- Se sono in posizione 4 recupero dallo stack il valore di porte
- Se sono in posizione 5 recupero dallo stack il valore di home

Stampo la stringa e torno nel main
Salto a inizio_rilevamento e chiamo la funzione rileva_direzione nel file rileva_direzione (fase 5, Rileva input).

3.7 Fase 5 (rileva input)

```
rileva_direzione:          #FASE5
xorl %ecx, %ecx

movl %esp, %esi
movl $3, %eax
movl $1, %ebx
leal input_freccia, %ecx  #il risultato viene salvato in ecx
movl $5, %edx
int $0x80

xorl %edx, %edx

movb (%ecx, %edx), %al    #sposto il primo byte(carattere)
cmpb $10, %al             #se invio devo uscire
je fine
cmpb $27, %al             #se ^[ vado avanti, se valore diverso esco
jne svuota_buffer
incl %edx

movb (%ecx, %edx), %al   #sposto secondo carattere
cmpb $91, %al            #se [ vado avanti, se valore diverso esco
jne svuota_buffer
incl %edx

movb (%ecx, %edx), %al   #sposto terzo carattere
incl %edx                #non faccio controlli sul 3 carattere

movb (%ecx, %edx), %ah   #sposto quarto carattere
cmpb $10, %ah            #se invio è molto probabile che abbia una freccia
je fine

incl %edx
movb (%ecx, %edx), %ah   #sposto quinto carattere
cmpb $48, %ah            #se [ vado avanti, se valore diverso esco
je fine                  #FASE5
```

rileva_direzione: systemcall sys_read, si mette in attesa dell'input, e salva i primi 5 caratteri inseriti nella variabile input_freccia.

Valuto i 5 caratteri uno alla volta, confrontando i relativi valori decimali nella codifica ascii:

- se rilevo solo un invio, ho un l'input valido e salvo il valore (decimali in ascii)nei primi 8 bit di EAX(%al),
- se rilevo freccia+invio, ho un input valido e salvo la direzione nei primi 8 bit di EAX(%al),
- in tutti gli altri casi non vengono eseguite operazioni (fase 6 Comandi).

Valuto il 5° carattere, se diverso da 0, inizio ciclo di sys_read per svuotare stream di input.
Nel caso sia stata inserita una stringa più lunga di 4 caratteri (freccia+invio occupa 4 char)

- primo secondo e terzo carattere li salvo in %al per il confronto sulla sequenza.
- il quarto carattere lo salvo in %ah, in questo modo se l'input è una freccia, nel quarto abbiamo l'invio '10', e nel terzo la direzione della freccia.

Se il primo carattere è invio salvo in %al e torno al main.

3.8 Fase 6 (comandi)

```
inizio_rilevamento:
   call rileva_direzione


    cmpb $65, %al             # comparo con A la freccia
    je su
    cmpb $66, %al             # comparo con B la freccia
    je giu
    cmpb $67, %al             # comparo con C la freccia
    je destra
    cmpb $10, %al             # controllo se ho ricevuto solo un invio
    je invio
    cmpb $0, %al
    je input_errato
   cmpb $68, %al
    je fine

    jmp fine                  # se non ho nessuno dei casi precedenti esco
```

Compare di %al e confronto con valori 65,66,67,68,10,0:

- se 0: input errato e ristampo il menu (iniziomenu).
- se 68: (freccia sinistra), esco dal programma con system call sys_exit (solo da menù)
- se 10: (invio), resetta sottomenu (la variabile sottomenu la pongo a 0), se ci si trovava nel sottomenu si esce, altrimenti rimango nel menu e ristampo.
- se 65: (freccia su), recupero la variabile 'posizione_menu' e controllo se sono in posizione 1:
  - se posizione_menu = 1 allora posizione_menu= max_posizione
  - se posizione_menu != 1 allora posizione_menu= posizione_menu-1(decremento).
- se 66: (freccia giù), recupero la variabile posizione_menu e controllo se sono in max_posizione:
  - se posizione_menu = max_posizione allora posizione_menu=1
  - se posizione_menu != max_posizione allora posizione_menu = posizione_menu +1(incremento)
- se 67: (freccia destra), recupero la variabile posizione_menu e controllo se è valorizzata (4|5|7|8), in questo caso attivo sottomenu a 1, mentre in caso contrario ristampo il menu

3.9 Fase 7 (rileva_numero)

rileva_numero: systemcall sys_read, si mette in attesa dell'input e salva i primi 5 caratteri inseriti nella variabile input_numero.

Valuto i 5 caratteri uno alla volta, confrontando i relativi valori decimali nella codifica ASCII:

- se rilevo soltanto invio ho l'input valido e salvo il valore (decimali in ASCII) nei primi 8 bit di EAX(%al).
- se rilevo 'numero+invio', ho un input valido e faccio un controllo sul numero inserito:
  - ➔ se incluso tra 2 e 5, salvo il valore nella variabile frecce (recuperando il puntamento dello stack).
  - ➔ se minore o uguale a 2, salvo il numero 2 nella variabile frecce.
  - ➔ se maggiore o uguale a 5, salvo il numero 5 nella variabile frecce.
- se rilevo da 2 a 4 numeri+invio, salvo il numero 5 nella variabile frecce.
- in tutti gli altri casi ho un input non valido.

Se sono stati inseriti più di 5 caratteri viene svuotato il buffer dedicato.
In seguito torno al main ristampando il menù nello stesso stato in cui si trovava prima di ricevere l'input.

```
primo_carattere:
  movb (%ecx, %edx), %al    #sposto il primo byte(carattere)
        cmpb $10, %al             #se invio devo uscire
        je fine
        cmpb $48, %al                       #controllo se è maggiore di 0 in ascii
        jl input_errato
  cmpb $57, %al             #controllo se è maggiore di 9 in ascii
  jg input_errato
        incl %edx

secondo_carattere:
  movb (%ecx, %edx), %al    #sposto il primo byte(carattere)
  cmpb $10, %al             #se invio devo uscire
  je fine_sequenza
  cmpb $48, %al             #controllo se è maggiore di 0 in ascii
  jl input_errato
  cmpb $57, %al             #controllo se è maggiore di 9 in ascii
  jg input_errato
  incl %edx


  ciclo:
  cmpl $6,%edx
  je input_errato
  movb (%ecx, %edx), %al    #sposto il secondo byte(carattere)
  cmpb $10, %al             #se invio devo uscire
  je max
  cmpb $48, %al             #controllo se è maggiore di 0 in ascii
  jl input_errato
  cmpb $57, %al             #controllo se è maggiore di 9 in ascii
  jg input_errato
  incl %edx
  jmp ciclo
```

3.10 Fase 8 (ciclo)

Salto a inizio_menu, e riprendo il ciclo dalla fase 3 (Controllo sottomenu).

4 SCELTE PROGETTUALI & CONCLUSIONI:

4.1 Comando inesistente

La prima delle nostre scelte progettuali è la gestione di un comando al di fuori di quelli prestabiliti e con una funzione ben precisa.
Inizialmente è stato pensato di stampare un messaggio di errore e proseguire a richiedere un ulteriore input.
Non è stata questa però, la nostra scelta finale, abbiamo ritenuto più corretto ignorare semplicemente l'ingresso fino a riceverne uno corretto, ristampando così il messaggio precedente.

4.2 Frecce direzione

Riguardo al punto 7 del menù, i valori che non sono numeri vengono gestiti come sopra citato, ovvero si ignora la richiesta, e viene quindi stampato lo stato precedente del setting.
Vengono accettati però soltanto numeri naturali e con 4 cifre al massimo.
Soltanto numeri naturali perché a parere nostro risulterebbe un po' fuori norma ricevere un valore negativo per settare il numero dei lampeggi.

4.3 Chiusura programma

La freccia a sinistra, aggiunta da noi come scelta progettuale, serve per chiudere completamente il programma, come una sorta di exit.
Il programma può essere chiuso soltanto se ci si trova nel menù.
Se ci si dovesse trovare nel sottomenù, bisognerebbe scalare con 'invio' per tornare al menù e avere quindi la possibilità di chiusura completa.

4.4 Problemi & Soluzioni

Un problema che abbiamo notato, è che utilizzando la system call 'sys_read', se vengono inseriti a terminale più valori rispetto al numero specificato in '%edx', questi rimangono salvati nello strem e vengono utilizzati alla successiva chiamata di 'sys_read'.
Per ovviare a questo problema abbiamo aggiunto un controllo sulla lunghezza della stringa inserita, se quest'ultima supera la lunghezza definita lo stream viene svuotato richiamando ciclicamente la 'sys_read' finché non viene rilevato un invio.

```
svuota_buffer:   #controllo
movl %esp, %esi
movl $3, %eax
movl $1, %ebx
leal input_freccia, %ecx
movl $1, %edx
int $0x80

movb (%ecx), %al
cmpb $10, %al
jne svuota_buffer  #controllo
```