ELABORATO ASM 2020

24 LUGLIO

Architettura degli Elaboratori

Autore: Simone Pavanello VR446236

Autore: Mattia Burato VR445864

Presentazione del progetto

Si ottimizzi un codice in linguaggio C che controlla un dispositivo per la gestione intelligente di un parcheggio mediante l'uso di Assembly inline. Il parcheggio è suddiviso in 3 settori: i settori A e B hanno 31 posti macchina ciascuno, mentre il settore C ha 24 posti macchina. Al momento dell'ingresso l'utente deve dichiarare in quale settore vuole parcheggiare, analogamente al momento dell'uscita l'utente deve dichiarare da quale settore proviene.

Il parcheggio rimane libero durante la notte, permettendo a tutte le macchine di entrare e uscire a piacimento. La mattina il dispositivo viene attivato manualmente da un operatore che inserisce manualmente il numero di automobili presenti nei tre settori. Nel suo funzionamento autonomo il sistema riceve in ingresso una stringa contenente il tipo di richiesta che può essere **IN** oppure **OUT**, seguito dal settore in cui l'utente vuole parcheggiare (**A**, **B** o **C**). Ad ogni richiesta il dispositivo risponde aprendo una sbarra e aggiornando il conteggio dei posti liberi nei vari settori. Se un utente chiede di occupare un settore già completo il sistema non deve aprire alcuna sbarra. Nel caso in cui la stringa sia corrotta il dispositivo non deve aprire nessuna sbarra.

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:
`$ ./parking testin.txt testout.txt`
Il programma deve leggere il contenuto di `testin.txt` e restituire il risultato delle elaborazioni in `testout.txt`.

Il file `testin.txt` contiene nelle prime tre righe il numero di automobili presenti in ogni settore (non necessariamente nell'ordine prestabilito) con la sintassi **SETTORE-NPOSTI** (esempio: A-24). A partire dalla quarta, ogni riga rappresenta una richiesta nella forma **TIPO-SETTORE** (esempio: IN-A oppure OUT-C).

Il file `testout.txt` contiene in ogni riga lo stato del Sistema per quanto riguarda le sbarre e l'occupazione dei settori. La sintassi da utilizzare è **SBARRE-NPOSTIA-NPOSTIB-NPOSTIC-LIGHTS**, con il seguente significato:
• **SBARRE**: è una coppia di char che rappresentano l'apertura (O, open) o chiusura (C, closed) delle sbarre di ingresso e uscita rispettivamente (esempio: CO significa che la sbarra in ingresso è chiusa e quella in uscita aperta).
• **NPOSTIx** (dove x può essere A, B o C): sono il numero di posti attualmente occupati nel rispettivo settore. Usare per tutti i settori rappresentazione a due cifre (es. 04 invece che 4).
• **LIGHTS**: è una terna di valori che rappresenta lo stato di accensione di una luce rossa in corrispondenza a ciascun settore per indicare che è pieno (esempio: 110 significa che i settori A e B sono pieni mentre ci sono ancora posti disponibili nel settore C).

Le variabili e il loro scopo

Main.s

NUMERO_A: contiene il valore del parcheggio A letto da bufferin sotto codifica ASCII, serve per poi trasformarlo in long grazie ad ATOI
NUMERO_B: contiene il valore del parcheggio B letto da bufferin sotto codifica ASCII, serve per poi trasformarlo in long grazie ad ATOI
NUMERO_C: contiene il valore del parcheggio C letto da bufferin sotto codifica ASCII, serve per poi trasformarlo in long grazie ad ATOI
A: Rappresenta il numero dei parcheggi di A convertito in intero
B: Rappresenta il numero dei parcheggi di B convertito in intero
C: Rappresenta il numero dei parcheggi di C convertito in intero

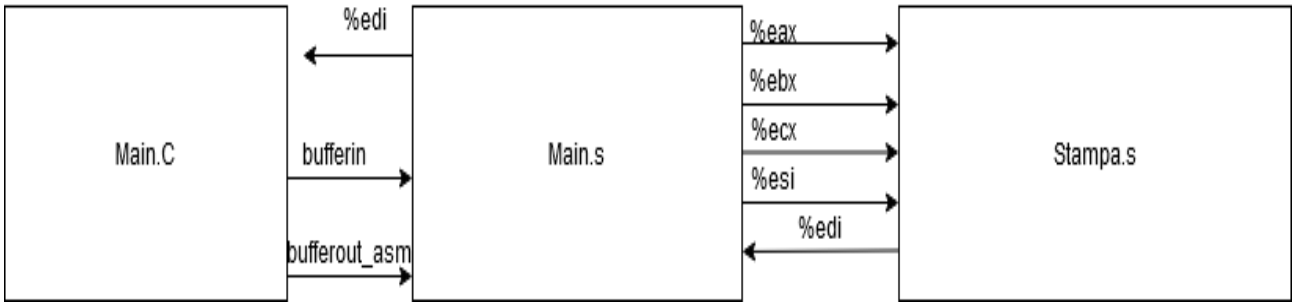
Stampa.s

P_A: dopo che chiamo la funzione stampa svuoto il registro %eax che contiene il valore A al suo interno, per poterlo poi continuare a usare solo quando ho bisogno di aggiornare il contatore in base alle entrate e alle uscite.
P_B: dopo che chiamo la funzione stampa svuoto il registro %ebx che contiene il valore B al suo interno, per poterlo poi continuare a usare solo quando ho bisogno di aggiornare il contatore in base alle entrate e alle uscite.
P_C: dopo che chiamo la funzione stampa svuoto il registro %ecx che contiene il valore C al suo interno, per poterlo poi continuare a usare solo quando ho bisogno di aggiornare il contatore in base alle entrate e alle uscite.

"Abbiamo inoltre le variabili bufferin e bufferout_asm che ci vengono passate dal file .C e che salviamo in due registri %esi per bufferin e %edi per bufferout"

Le modalità di passaggio/restituzione dei valori delle funzioni create

Il nostro progetto ha 2 funzioni ASM, una che viene chiamata dal C e una che viene chiama dalla funzione ASM che viene chiamata dal C.



La funzione C passa bufferin e bufferout_asm che verranno prelevati dallo stack dal main.s, caricherà nei registri eax,ebx,ecx i valori dei parcheggi letti da esi che conteneva il puntatore di bufferin e invierà questi registri più edi che contiene l'indirizzo di bufferout a stampa.s che preleverà tutto dallo stack, continuerà a leggere bufferin, aggiornerà i registri e restituira infine allo stack tutto quanto tra cui edi che contiene lo stesso indirizzo di memoria di bufferout_asm, solamente che alla fine quando, stampa.s farà il ret il contenuto di edi sarà quello che poi verrà passato nel C grazie al main.S e poi verrà scritto sul file grazie al C.

Makefile

Per una più agevole compilazione del codice abbiamo inserito nel materiale da consegnare un **MAKEFILE,** il quale permette di compilare e linkare il progetto attraverso il comando **make** o di ripulire la cartella dai file creati con il comando **make clean.**

Scelte Progettuali

1) Salvataggio dei valori dei parcheggi letti in delle variabili.
2) Divisione del progetto ASM in due file.s: uno per l'inizializzazione dei parcheggi e l'altro per la scrittura su bufferout, che viene chiamato tramite call dal primo.
3) Controlliamo ogni posizioni di esi(bufferin) e in base all'input che troviamo aggiungiamo il carattere per la stama ad edi(bufferout).
4) Nel momento in cui in Esi troviamo una I o una O controlliamo le posizioni dopo per essere sicuri che sia scritto in maniera corretta e in tal caso aumentiamo o decrementiamo il parcheggio che sarà selezionato
5) Abbiamo infine generato un ciclo con il contatore ecx, in modo da continuare a leggere esi fino al termine e scrivere su edi in posizione di ecx.