Architettura degli elaboratori: Elaborato di Assembly

Marian Statache, Elia Piccoli

A.A. 2017/2018

Sommario

Introduzione

Affronteremo qui un sistema già visto tramite SIS. Si tratta, infatti, della rielaborazione dello stesso sistema realizzato in linguaggio Assembly. Non sarà, tuttavia, una traduzione tra linguaggi, poiché diversi linguaggi permettono utilizzi diversi della macchina, quindi affronteremo problemi identici con logiche differenti. Sarà presenti inoltre, l'algoritmo di funzionamento scritto anche in linguaggio C, ma anche in questo caso, non avremo una relazione 1 a 1 tra le parti scritte in C ed in Assembly. Rivediamo in breve quello che dovrà fare il nostro sistema:

si desidera realizzare un programma per il monitoraggio del consumo degli elettrodomestici.
Ricevuta in ingresso una stringa contenente i tre reset e un elenco dei consumi per ognuno degli n test, restituisce una stringa con la relativa fascia di consumo energetico: *Fascia 1*, *Fascia 2*, *Fascia 3*, o sovraccarico (*Overload*); inoltre, sempre su questa stringa, vengono riportati i valori degli interruttori del generale, della lavatrice e della lavastoviglie. Queste ultime vengono definite linee critiche di consumo, ovvero, in caso di 4 o più cicli in Overload, verrà spenta la lavastoviglie, poi al quinto ciclo la lavatrice e, infine, se si giunge al sesto ciclo di clock avendo un consumo fuori fascia, verrà spento l'interruttore generale, e, di conseguenza, l'intero sistema.

Funzionamento - Pseudocodice

Il funzionamento dell'algoritmo può essere spiegato e riassunto tramite il seguente pseudocodice.

INIZIO

SE RES_GEN = 0 il sistema è **SPENTO**
ALTRIMENTI ACCENDO tutti gli **INT, AZZERA CONTATORE DEI CICLI** e **CALCOLA FASCIA**

Passo al **PROSSIMO INPUT**

INIZIO

SE INT_GEN = 1 controllo **RES_DW** e **RES_WM**
ALTRIMENTI
SE RES_GEN = 0 e **INT_GEN = 0** il sistema è **SPENTO** e passo al **PROSSIMO INPUT**
ALTRIMENTI ACCENDO il sistema e **CALCOLO FASCIA**

Una volta stabiliti i valori degli **INT** calcolo il **CONSUMO TOTALE**
Per ogni elettrodomestico **i da 1 a 10**
SE LOAD[i] = 1 l'elettrodomestico è **ACCESO** e sommo il suo carico

Finito di calcolare il **CONSUMO TOTALE** controllo in che **FASCIA** siamo
SE CONSUMO < 150daW siamo in **FASCIA 1** e **AZZERO CONTATORE**
ALTRIMENTI SE CONSUMO < 300daW siamo in **FASCIA 2** e **AZZERO CONTATORE**
ALTRIMENTI SE CONSUMO < 450daW siamo in **FASCIA 3** e **AZZERO CONTATORE**
ALTRIMENTI siamo in **OVERLOAD**

SE OVERLOAD controllo numero di cicli in OL
SE 4 SPENGO DW
SE 5 SPENGO DW e **WM**
SE 6 SPENGO SISTEMA

(PROSSIMO INPUT)
INCREMENTO [in] in modo che punti all'**INPUT SUCCESSIVO**
AGGIORNO i **REGISTRI** che conterranno valore degli **INT** per il prossimo input
INCREMENTO [out] in modo che punti alla casella dove andrà il **PROSSIMO OUTPUT**

SE [in] PUNTA A '\0' FINE
ALTRIMENTI RICOMINCIO

FINE

FINE

Variabili e registri

Come prima cosa abbiamo il passaggio alla funzione assembly degli indirizzi delle stringhe *'bufferin'* e *'bufferout_asm'*, trasferendoli in registri generici tramite il comando "r" e attribuendo loro delle etichette, ovvero i nomi con cui chiameremo in causa questi registri, rispettivamente [in] e [out].

Registri:

Vediamo ora i registri utilizzati e lo specifico ruolo di ognuno.

- **AX** ← somma dei consumi
- **BH** ← valore dell'interruttore generale
- **BL** ← valore dell'interruttore della lavastoviglie
- **CH** ← valore dell'interruttore della lavatrice
- **CL** ← contatore del numero di cicli in overload

Scelte progettuali

Per ottimizzare il più possibile il nostro elaborato abbiamo, ovviamente, preso delle decisioni ed effettuato delle scelte, più o meno incisive, ma comunque tutte mirate ad una maggiore efficienza.

Innanzitutto abbiamo deciso di non occupare interi registri per una sola variabile. Abbiamo notato, infatti, che gli interruttori occupano soltanto un byte, essendo caratteri ASCII, e questo ci ha permesso di utilizzare le frazioni di registri **BH**, **BL**, e **CH**, che hanno dimensione appunto 1 byte.
C'è poi il contatore dei cicli in overload, che dovendo arrivare fino a 6, necessita di soli 3 bit, e, non essendo possibile accedere ad una parte così piccola dei registri, abbiamo utilizzato **CL**, anch'esso da 1 byte.
Infine, c'è il consumo totale, che al massimo vale 954daW, rendendo sufficienti 10bit, quindi abbiamo potuto metterlo solo su **AX**, invece che su tutto EAX.

Altra importante scelta, è stata quella di strutturare il codice a blocchi funzionali, in modo da accorciarlo e semplificarlo il più possibile, non usando, però, funzioni, quindi call, ma solo istruzioni di salto, per avere prestazioni ancora più alte. Ciò implica chiaramente, una rigidità del codice molto alta, poiché ad esempio l'inversione di istruzioni o anche di blocchi di codice è una faccenda molto delicata.

Restando sempre sulle istruzioni di salto, abbiamo anche riflettuto su dove posizionarle e sulle condizioni di ogni salto, per fare in modo che vengano effettuati meno salti possibili.

I vari consumi specifici, pur essendo delle costanti, abbiamo deciso di scriverli direttamente nel codice, senza dichiararli come costanti, poiché servono solamente una volta nel codice.

Ultima scelta, ma non meno importante, è stata quella di non convertire i valori della stringa in input in interi, per poi riconvertirli in ASCII dopo il calcolo per riscriverli sul *bufferout_asm*. Tutto ciò non è stato necessario perché i caratteri di input dovevano essere utilizzati solo da istruzioni *compare*, quindi abbiamo potuto lavorare direttamente in ASCII, risparmiando tempo e risorse che sarebbero andate sprecate per due conversioni inutili.

Test e statistiche: conclusioni finali

Una volta finalizzato il tutto, ci siamo concentrati sulla valutazione delle prestazioni del nostro elaborato. Abbiamo prima di tutto sottoposto i 18 test forniti dal prof, e, dopo aver verificato la loro correttezza, confrontando il file *testout* generato dal programma con il file *trueout*, sempre fornito dal prof, abbiamo sottoposto altri 96 test pensati da noi mirati ai casi particolari.

Abbiamo poi preso una media dei tempi calcolata su 20 esecuzioni di entrambi i file, prima nel laboratorio alfa e poi nel laboratorio delta, su due macchine scelte casualmente. Confrontando la media dei tempi di esecuzione dell'algoritmo scritto in C e quello scritto in Assembly, il miglioramento di prestazioni dal primo al secondo è circa del 247% (vedi il file *Tempi.xlsx*, il quale mostra tutti i dati raccolti).

Da notare che questo miglioramento è stato possibile in gran parte grazie alla possibilità di lavorare più a basso livello, grazie ad Assembly. Anche la lunghezza dello stesso codice, contro le aspettative, si è rivelata inferiore al C.

Un'ultima considerazione da fare è che, se avessimo saputo come accedere ed utilizzare la parte di registro complementare rispetto ad **CX** di **ECX**, ad esempio, avremmo potuto salvare i tre interruttori e il contatore di cicli in overload solamente su un registro, **ECX** appunto, e insieme alla parte di registro **AX** usato per il consumo totale, ci sarebbero bastati in tutto un registro e mezzo, coinvolgendo i due registri **EAX** ed **ECX**. Poiché, però, queste conoscenze esulano dalle nostre competenze (in realtà non sappiamo nemmeno se tale cosa è possibile), abbiamo utilizzato tre registri **EAX**, **EBX** ed **ECX**, comunque un ottimo risultato.