count-char.asm – Analisi simulazione

a cura di Daniele Sana

Questo programma conta quante volte il carattere immesso da tastiera compare nella stringa memorizzata.

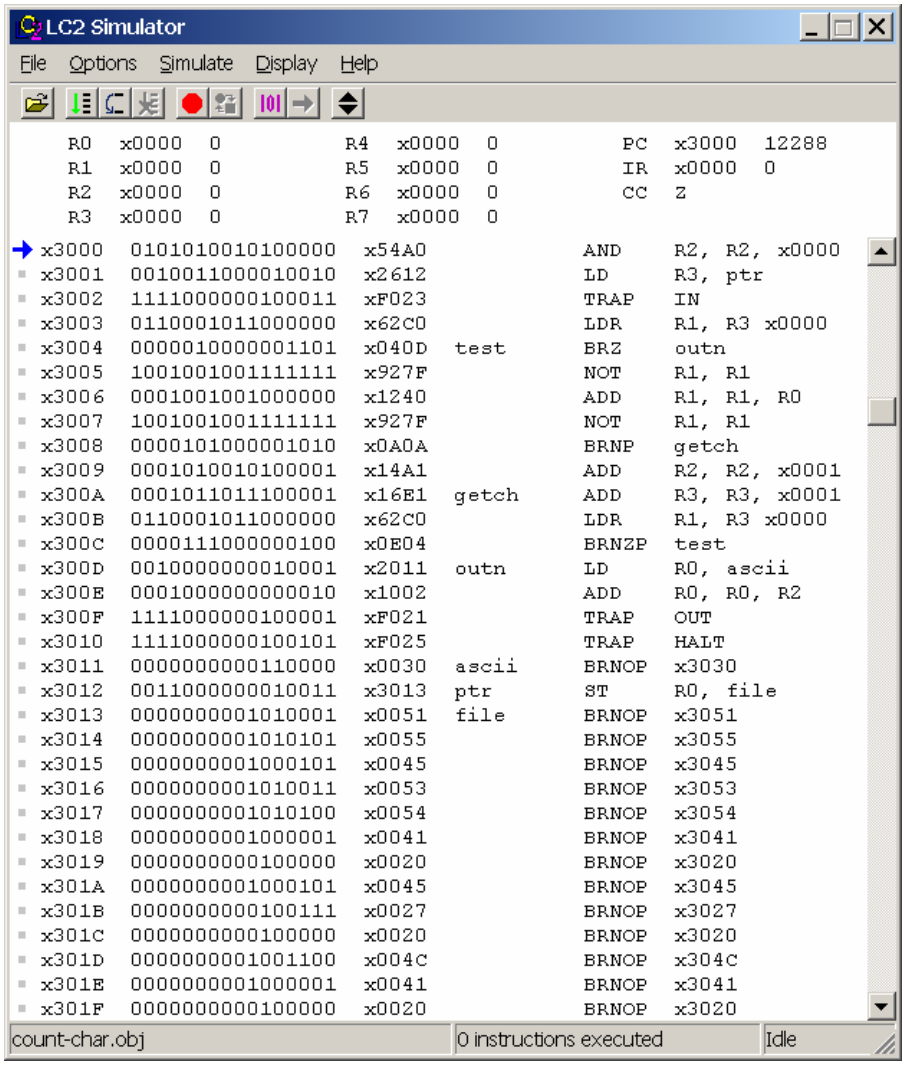
Il testo del programma principale è il seguente:

```
;
; Initialization
;
      .orig x3000
      AND   R2,R2,#0    ; R2 is counter, initialize to 0
      LD    R3,ptr            ; R3 is pointer to characters in the string
      TRAP  x23         ; R0 gets character input
      LDR   R1,R3,#0    ; R1 gets first character in string
;
; Test character for end of string
;
test  BRZ   outn        ; test for NULL; if done, prepare to output
;
; Test character for match
;
      NOT   R1,R1
      ADD   R1,R1,R0    ; if match, R1 = xFFFF
      NOT   R1,R1       ; if match, R1 = x0000
      BRNP  getch       ; if no match, skip the increment
      ADD   R2,R2,#1
;
; Get next character from string
;
getch ADD   R3,R3,#1    ; increment the pointer
      LDR   R1,R3,#0    ; R1 gets next character in string
      BR    test        ; loop to test character
;
; Output the count
;
outn  LD    R0,ascii    ; load the ASCII template for convert
      ADD   R0,R0,R2    ; convert binary to ASCII
      TRAP  x21         ; ASCII code in r0 is displayed
      TRAP  x25         ; Halt machine
;
; Storage for pointer, ASCII template and string
;
ascii .fill x0030
ptr   .fill file
file  .stringz "QUESTA E' LA STRINGA IN CUI CERCARE I CARATTERI"

      .end
```

A questo punto, dopo aver compilato il programma utilizzando il programma *LC2Edit*, passiamo allo strumento *Simulate* ed effettuiamo il *debugging*, ossia eseguiamo il programma un'istruzione alla volta.

In *Simulate* la situazione è la seguente:



Vediamo innanzitutto quali saranno gli effetti delle pseudo-istruzioni *Assembly* presenti nel codice:

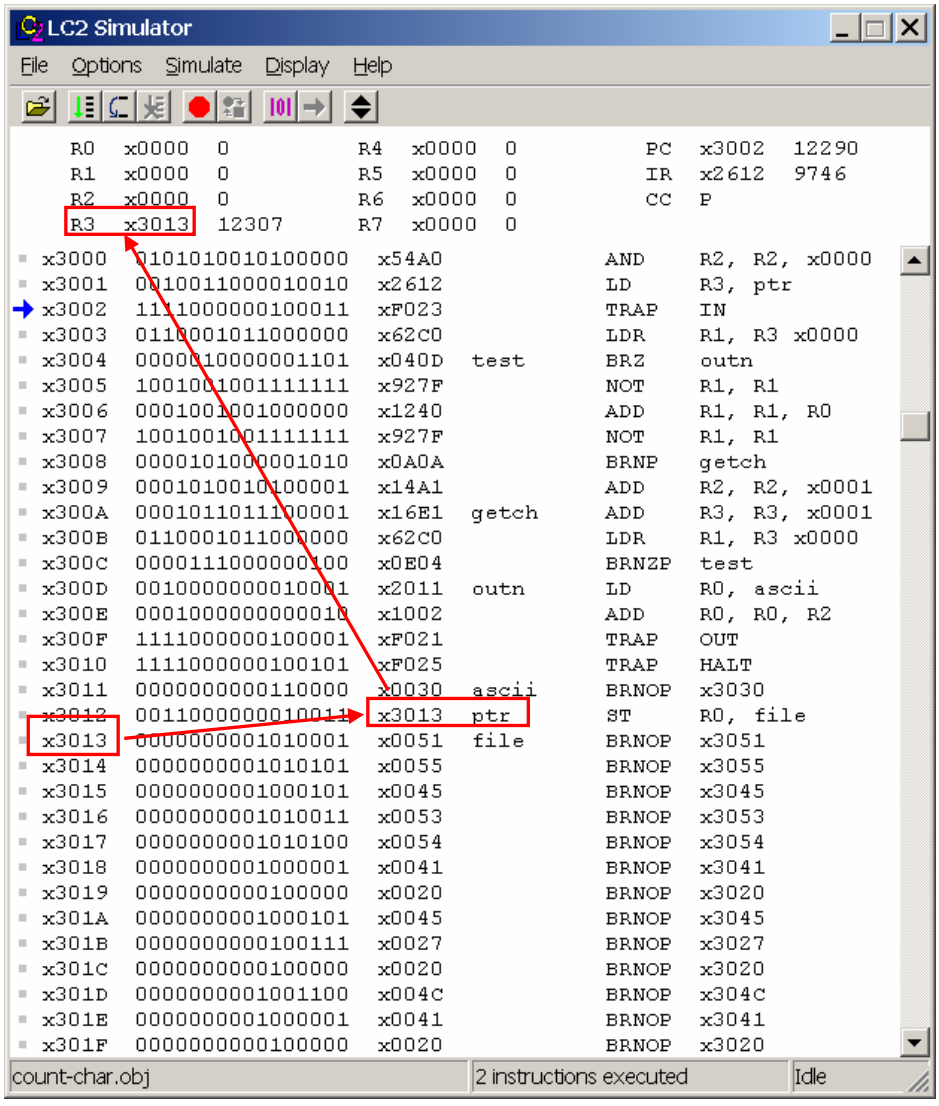
- **.orig x3000** ha segnalato all'*Assembler* che il programma deve essere caricato all'indirizzo x3000;
- la pseudo-istruzione **.fill x0030**, in corrispondenza della label **ascii**, ha riservato una cella di memoria contenente la costante x0030. Si ricorda che in codifica ASCII le cifre decimali partono dalla codifica x0030 (x0030 = '0', x0031 = '1', .. , x0039 = '9');
- la pseudo-istruzione **.fill file**, in corrispondenza della label **ptr**, ha riservato una cella di memoria contenente l'indirizzo del primo carattere della stringa memorizzata (che potrebbe quindi trovarsi ovunque nella memoria dell'LC-2);
- la pseudo-istruzione **.stringz**, in corrispondenza della label **file**, ha inizializzato una sequenza di celle di memoria con la codifica ASCII della frase "QUESTA E' LA STRINGA IN CUI CERCARE I CARATTERI", terminata dal valore 0.

La tabella dei simboli generata dall'*Assembler* è la seguente:

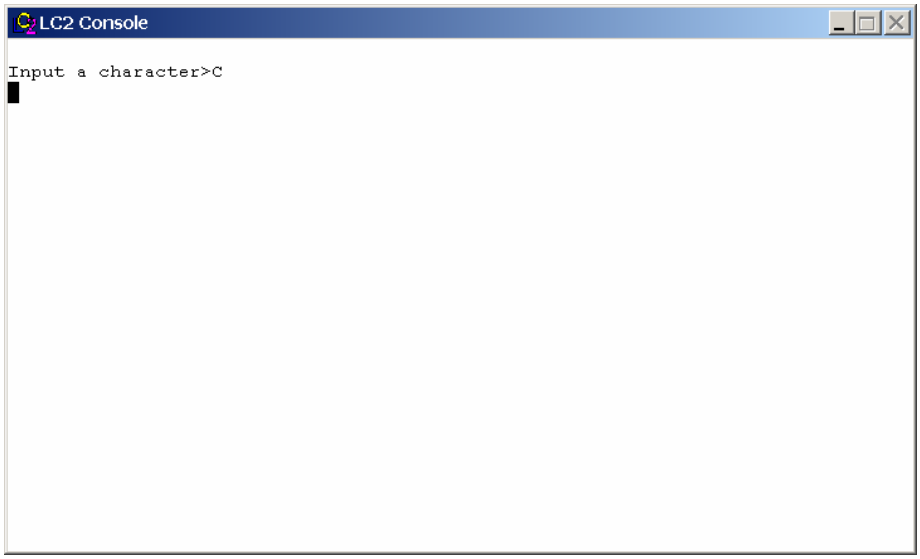
```
// Symbol table
// Scope level 0:
//    Symbol Name       Page Address
//    ----------------  ------------
//    ascii                     3011
//    file                      3013
//    getch                     300A
//    outn                      300D
//    ptr                       3012
//    test                      3004
```

Facciamo ora partire il programma con il comando *Simulate→ Step (F8)* e analizziamo ogni singola istruzione.

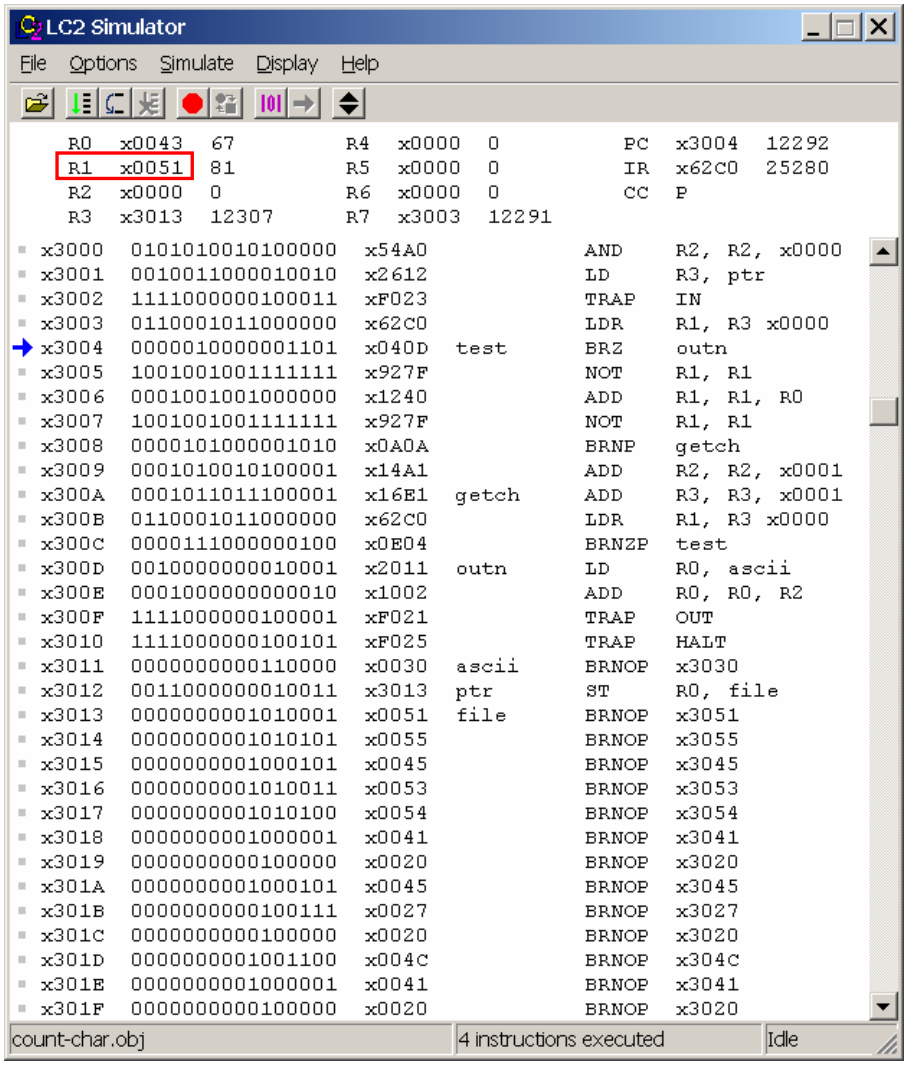
Dopo la prima istruzione **AND R2, R2, x0000**, che inizializza al valore 0 il contenuto del registro R2, viene eseguita l'istruzione **LD R3, ptr**, che carica nel registro R3 l'indirizzo di partenza della stringa (x3013).



Quando viene eseguita la successiva istruzione di **TRAP IN** la console si pone in attesa di un carattere, che sarà il carattere che andremo a cercare all'interno della stringa memorizzata. La codifica ASCII di tale carattere viene salvata nel registro R0. Supponiamo di voler contare quante volte il carattere 'C' è presente nella frase "QUESTA E' LA STRINGA IN CUI CERCARE I CARATTERI".



L'istruzione LDR R1, R3, x0000 legge la codifica ASCII del primo carattere della stringa e la salva nel registro R1. Il primo carattere letto è 'Q', pertanto la codifica ASCII è pari a x0051.



A questo punto inizia il ciclo del programma, identificato dalla label **test**: il ciclo terminerà quando verrà incontrato un valore nullo, che corrisponde al terminatore della stringa memorizzata. L'uscita dal ciclo è gestita dall'istruzione **BRZ outn**: fino a quando ci saranno caratteri da leggere, tale istruzione non porterà alla cella di memoria identificata dalla label **outn**.

Per confrontare fra loro due valori, in assenza di istruzione macchina specifica, si fa la somma in complemento a 2 e si vede se il risultato è nullo. Per operare tale somma, si può fare il complemento bit a bit del primo numero, sommare uno, quindi sommare il secondo numero e vedere se il risultato è 0; oppure si può fare il complemento bit a bit del primo numero, sommare direttamente il secondo numero e vedere se il risultato è -1 (ovvero xFFFF) cioè se il complemento bit a bit del risultato è 0. Gli autori del testo che introduce la struttura della CPU LC-2 hanno scelto questa seconda strada.

Partiamo dalle seguenti ipotesi:

- R0 = x0043 → codifica ASCII carattere 'C';

- R1 = x0051 → codifica ASCII carattere 'Q'.

Il confronto viene effettuato dalla seguente serie di istruzioni:

- **NOT R1, R1** → R1 = xFFAE = #-82
- **ADD R1, R1, R0** → R1 = xFFF1 = #-15
- **NOT R1, R1** → R1 = x000E = #14

Dopo queste tre istruzioni il contenuto di R1 NON è nullo, pertanto:

- l'istruzione **BRNP getch** effettua un salto all'istruzione identificata dalla label **getch**:
  - **ADD R3, R3, x0001** → R3 contiene ora l'indirizzo del carattere successivo della stringa;
  - **LDR R1, R3, x0000** → viene letto il nuovo carattere da confrontare;
  - **BRNZP test** → si ritorna all'inizio del ciclo.

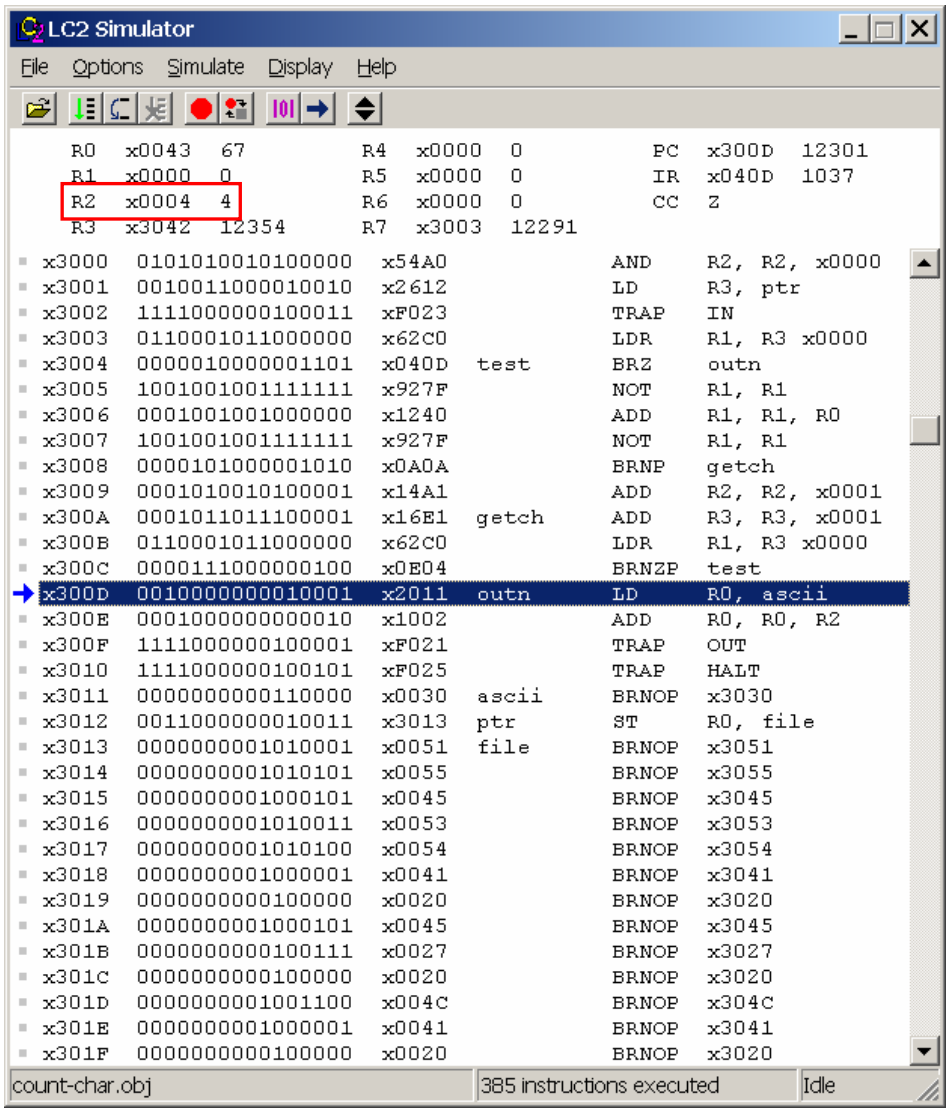
Per contro, quando si incontra un carattere 'C' nella stringa, il confronto è il seguente:

- **NOT R1, R1** → R1 = xFFBC = #-68
- **ADD R1, R1, R0** → R1 = xFFFF = #-1
- **NOT R1, R1** → R1 = x0000 = #0

Dopo queste tre istruzioni il contenuto di R1 è nullo, pertanto:

- l'istruzione **BRNP getch** NON effettua un salto all'istruzione identificata dalla label **getch;**
- **ADD R2, R2, x0001** → il contatore delle occorrenze del carattere da cercare viene incrementato di uno;
- **ADD R3, R3, x0001** → R3 contiene ora l'indirizzo del carattere successivo della stringa;
- **LDR R1, R3, x0000** → viene letto in nuovo carattere da confrontare
- **BRNZP test** → ritorno all'inizio del ciclo.

Al termine del ciclo, con l'ipotesi di aver cercato nella stringa il numero di occorrenze del carattere 'C', il registro R2 contiene il valore 4, ossia il carattere 'C' è stato trovato 4 volte all'interno della stringa memorizzata.



Infine il programma termina con le seguenti istruzioni:

- **LD R0, ascii** → viene caricato in R0 il valore x0030, che rappresenta il contenuto della cella identificata dalla label **ascii**;
- **ADD R0, R0, R2** → si somma a R0 il numero di occorrenze del carattere cercato, in modo da identificare la corretta codifica ASCII del numero da visualizzare;
- **TRAP OUT** → viene visualizzato il risultato sulla Console;
- **TRAP HALT** → il programma viene arrestato.

Si noti che il programma qui presentato funziona solamente se il numero di occorrenze del carattere cercato è minore di 10, e può quindi essere espresso con una sola cifra decimale.

Una possibile evoluzione – peraltro piuttosto complessa – consiste quindi nel costruire un sottoprogramma capace di convertire un valore a 16 bit nei 4 caratteri corrispondenti alle 4 cifre esadecimali che rappresentano tale valore, o addirittura nelle 5 cifre decimali che rappresentano tale valore, magari eliminando gli zeri inutili.