Fondamenti di Informatica

Esercizi sul Little Man Computer

(a.a. 2019/2020, prof. Gianluca Amato – Francesca Scozzari)

I programmi contenuti in questo documento sono scritti utilizzando la notazione del simulatore LMC di Riven e non quella "ufficiale" su https://en.wikipedia.org/wiki/Little_man_computer. Le differenze sono principalmente due:

- Le etichette che indicano locazioni di memoria vanno SEGUITE dal simbolo di due punti quando vengono dichiarate, e precedute dal simbolo di due punti quando vengono USATE.
- La pseudo-istruzione DAT non esiste, ma al suo posto è sufficiente scrivere direttamente il numero che si vuole inserire in una cella di memoria.

Ad esempio, il programma che con la notazione standard sarebbe

```
ciclo   OUT
        LDA one
        BRA ciclo
one     DAT 1
```

con questa notazione diventa

```
ciclo:  OUT
        LDA :one
        BRA :ciclo
one:    1
```

Esercizio 1

Scrivere un programma per LMC che prende in input due numeri e ne restituisce la somma.

Soluzione

Diamo tre versioni di questo programma. La prima in linguaggio macchina, ovvero usando i codici numerici delle istruzioni, la seconda in assembly ma utilizzando indirizzi numerici per le celle di memoria, la terza in assembly con l'uso delle etichette. Nei compiti si consiglia sempre di scrivere il programma in assembly con l'uso delle etichette.

Si noti l'uso della locazione 99 (o della locazione all'etichetta `sum`) per conservare il primo dato in input fino al momento in cui si rivelerà utile per la somma.

1° versione (linguaggio macchina)

```
901
399
901
199
902
```

```
            000
```

2° versione (assembly con indirizzi numerici)

```
            INP
            STA 99
            INP
            ADD 99
            OUT
            HLT
```

3° versione (assembly con etichette)

```
            INP
            STA :sum
            INP
            ADD :sum
            OUT
            HLT
sum:        000
```

Esercizio 2

Scrivere un programma per LMC che prende in input un numero e restituisce il suo complemento a 9 (ovvero 999-*n*).

Soluzione

Utilizziamo una locazione inizializzata con il valore 999 da usare come numero base a cui sottrarre l'input.

```
            INP
            STA :x
            LDA :y
            SUB :x
            OUT
            HLT
x:          000
y:          999
```

Esercizio 3

Scrivere un programma per LMC che prende in input due numeri e restituisce il valore assoluto della differenza.

Soluzione

Da notare l'uso della istruzione `BRP`. Se `dat2-dat1` è positivo, salta alla istruzione etichettata da `end` che visualizza il risultato e termina l'esecuzione. Altrimenti il `BRP` non ha effetto e l'esecuzione continua con le istruzioni immediatamente successive a `BRP`, che provvedono a calcolare `dat1-dat2`. L'istruzione `OUT` viene comunque eseguita in entrambi i casi, ma provenendo da punti diversi del programma.

```
            INP
            STA :dat1
            INP
            STA :dat2
            SUB :dat1
            BRP :end
            LDA :dat1
            SUB :dat2
end:        OUT
```

```
            HLT
dat1:       0
dat2:       0
```

Esercizio 4

Scrivere un programma per LMC che prende in input due numeri e restituisce in output il massimo dei due.

Soluzione

```
            INP
            STA :first
            INP
            STA :second
            SUB :first
            BRP :m2
            LDA :first
            OUT
            HLT
m2:         LDA :second
            OUT
            HLT
first:      000
second:     000
```

Esercizio 5

Scrivere un programma per LMC che prende in input tre numeri e restituisce in output il massimo dei tre.

Soluzione

Ci sono varie soluzioni possibili. Quella che vi propongo qui sotto consente di riutilizzare quanto fatto nell'esercizio 4. Prima di tutto prendiamo in input due numeri e calcoliamo il loro massimo. A differenza che nell'esercizio 4, questo massimo non lo mandiamo in output, ma lo salviamo nella locazione con etichetta `max`. Poi prendiamo in input il terzo numero e, sempre sulla falsa riga del programma dell'esercizio 4, calcoliamo il massimo tra quest'ultimo e la locazione `max`.

```
            // prendo due input, calcolo il massimo, e metto
            // il risultato nella locazione max
            INP
            STA :first
            INP
            STA :second
            SUB :first
            BRP :m2
            LDA :first
            BRA :continua
m2:         LDA :second
continua:   STA :max
            // adesso prendo il terzo input, e faccio il massimo tra
            // questo input e max
            INP
            STA :third
            SUB :max
            BRP :m3
            LDA :max
            OUT
            HLT
```

```
m3:         LDA :third
            OUT
            HLT
first:      000
second:     000
third:      000
max:        000
```

Esercizio 6

Il seguente programma per LMC prende due numeri in input e restituisce un numero in output. Che relazione c'è tra l'output e l'input ?

```
            INP
            STA :first
            INP
            STA :second
            LDA :first
            BRZ :x
            OUT
            HLT
x:          LDA :second
            OUT
            HLT
first:      000
second:     000
```

Soluzione

Il programma restituisce in output il primo numero, se non è nullo, altrimenti restituisce il secondo numero.

Esercizio 7

Scrivere un programma che prende una sequenza di numeri (terminata dal numero 0) e restituisce in output il massimo della sequenza.

Soluzione

In questo e in altri esercizi che seguono, non si può sapere in anticipo quanto sarà lunga la sequenza di input: il programma deve prevedere un ciclo che continua a chiedere il prossimo numero fino a che l’utente non immette zero. Una locazione di memoria servirà a contenere il massimo corrente, che deve essere aggiornato ogni volta che si inserisce un nuovo numero.

Non provate a salvare in memoria i numeri immessi e a fare il conto alla fine, perché il linguaggio dell’LMC è molto primitivo e non consente facilmente di fare questa operazione.

Notare che, mentre in molti casi il valore iniziale delle locazioni di memoria usate per i dati non è importante, in questo programma è invece fondamentale che la locazione `max` sia inizializzata a zero.

```
ciclo:      INP
            BRZ :fine
            STA :input
            LDA :max
            SUB :input
            BRP :ciclo
            LDA :input
            STA :max
            BRA :ciclo
fine:       LDA :max
```

```
            OUT
            HLT
input:      0
max:        0
```

Esercizio 8

Scrivere un programma per il LMC che prende una sequenza di numeri in input e produce come output la sequenza delle differenze tra due numeri successivi. Il programma si interrompe quando viene immesso il numero 0. Esempio: se l'input è la sequenza 2, 4, 10, 3, 0 l'output sarà 2 (ovvero 4-2), 6 (ovvero 10-4), 993 (ovvero -7 in complemento a 10).

È importante notare che ogni numero (tranne il primo e l'ultimo) è usato due volte, una volta per la sottrazione col numero precedente, uno per la sottrazione col numero successivo. In caso di incertezza, guardare attentamente l'esempio proposto.

Soluzione

Molte soluzioni sono possibili. Eccone una:

```
            INP
            BRZ :fine
            STA :old
ciclo:      INP
            BRZ :fine
            STA :new
            SUB :old
            OUT
            LDA :old
            STA :new
            BRA :ciclo
fine:       HLT
old:        000
new:        000
```

Esercizio 9

Scrivere un programma per il LMC che prende una sequenza di numeri in input e produce come output la sequenza delle somme tra due numeri successivi. Il programma si interrompe quando viene immesso il numero 0. Esempio: se l'input è la sequenza 2, 4, 10, 3, 0 l'output sarà 6 (ovvero 4+2), 14 (ovvero 10+4), 13 (ovvero 3+10).

È importante notare che ogni numero (tranne il primo e l'ultimo) è usato due volte, una volta per la somma col numero precedente, uno per la somma col numero successivo. In caso di incertezza, guardare attentamente l'esempio proposto.

Esercizio 10

Il seguente programma per LMC prende un numero in input e restituisce un numero in output. Che relazione c'è tra l'output e l'input ?

```
       INP
       STA :n
ciclo: LDA :d
       ADD :d
       SUB :n
       BRP :fine
       LDA :d
```

```
       ADD :uno
       STA :d
       BRA :ciclo
fine:  LDA :d
       OUT
       HLT
n:     000
d:     000
uno:   001
```

Esercizio 11

Scrivere programma che prende in input due numeri (positivi) e restituisce il loro prodotto. Tenere conto del fatto che a*b = 0 + a + a + ... + a, dove la somma viene effettuata b volte.

Esercizio 12

Scrivere un programma per il LMC che, preso in input un numero *n* positivo, restituisca in output il numero 2^n. Si ignorino eventuali overflow.

Esercizio 13

Scrivere un programma per il LMC che, presa in input una sequenza di numeri terminata dal valore 99, restituisca 1 se la sequenza è crescente (in senso largo), 0 altrimenti. Ad esempio, l'output sarà 1 per la sequenza “3 10 21 21 30 99” mentre sarà 0 per “3 10 21 8 10 99”.

Esercizio 14

La successione di Fibonacci, indicata con F_n, è una successione di numeri naturali ottenuta come segue:

$$F_1 = 1 \qquad F_2 = 1$$

$$F_n = F_{n-1} + F_{n-2} \text{ per } n > 2$$

Ad esempio, $F_3 = 2$, $F_4 = 3$, $F_5 = 5$, etc… Scrivere un programma per il LMC che prende in input un numero n e restituisce in output il valore F_n.

Esercizio 15

Scrivere un programma per il LMC che prende in input due numeri, dopo di che manda in output, progressivamente, tutti i numeri interi dal più grande di quelli immessi al più piccolo. Ad esempio, se l’utente immette 5 e 2, il programma dovrà mandare in output i numeri 5, 4, 3, e 2. Se l’utente immette prima 2 e poi 5, l’output deve essere sempre 5, 4, 3, 2.

Esercizio 16

Scrivere un programma per il LMC che prende in input due numeri, dopo di che manda in output, progressivamente, tutti i numeri pari compresi tra il primo numero immesso ed il secondo. Ad esempio, se l’utente immette 3 e 8, il programma dovrà mandare in output i numeri 4, 6 ed 8. Si può assumere che il secondo numero immesso sia maggiore del primo.

Esercizio 17

Scrivere un programma per il LMC che prende in input una sequenza di numeri, e si ferma quando viene immesso un numero che è maggiore della somma dei due numeri precedenti. Il programma non produce alcun output.

Esercizio 18

Scrivere un programma per il LMC che prende in input una sequenza di numeri e produce in output la stessa sequenza, ma limitatamente ai soli numeri pari. Ad esempio, se la sequenza di input è 3 4 2 19 10, la sequenza di output corrispondente sarà 4 2 10.

Esercizio 19

Scrivere un programma per il LMC che prende in input una sequenza di numeri $x_1, \ldots, x_n$ e produce in output una sequenza $y_1, \ldots, y_n$ tale che $y_i = x_i + i$. Un valore di input nullo termina la sequenza. Ad esempio, alla sequenza di input 5, 3, 9, 12, 1, 0 corrispondente la sequenza di output 6 (5+1), 5 (3+2), 13 (9+3) e 5 (1+4).

Esercizio 20

Scrivere un programma per il LMC che prende in input una sequenza di numeri e produca in output la sequenza delle somme parziali. Il programma termina quando riceve l'input 0. Ad esempio, se l'input è la sequenza 1, 4, 12, 3, 0 l'output sarà la sequenze 1, 5, 17, 20.

Esercizio 21

Scrivere un programma per il LMC che prenda in input una sequenza di numeri (da interpretare come una stringa in codice ASCII), e produca in output un'altra sequenza di numeri (anch'essa da interpretare come stringa sulla base del codice ASCII) tale che:

- per ogni carattere alfanumerico in input (lettera maiuscola/minuscola o cifra decimale) venga emesso in output esattamente lo stesso carattere;
- ogni altro carattere venga sostituito con lo spazio.

La stringa in input termina con il codice zero, che deve essere ripetuto nella stringa in output.

Ad esempio, se l'input è la stringa “Ancora␣un␣po'␣d'aglio” l'output dovrà essere la stringa “Ancora␣ un␣po␣␣d␣aglio”, dove è stato usato il simbolo ␣ per indicare uno spazio.

Esercizio 22

Scrivere un programma per il LMC che prenda in input una sequenza di numeri, terminata dal numero zero, e restituisca il prodotto di tutti i numeri immessi (escluso lo zero).

Esercizio 23

Scrivere un programma per il LMC che prende in input due numeri e calcola quoziente (intero) e resto della divisione tra il primo e il secondo numero.

Esercizio 24

Scrivere un programma per il LMC che e prende due numeri in input e restituisce il massimo

comune divisore usando la seguente variante dell'algoritmo di Euclide:

```
    function gcd(a, b)
        if a = 0
           return b
        while b ≠ 0
            if a > b
               a := a − b
            else
               b := b − a
        return a
```

Esercizio 25

Scrivere un programma per il LMC che prende in input un numero e determina se è primo. Si riutilizzi il programma per la divisione o per il massimo comun divisore.

Esercizio 26 (molto molto difficile)

Scrivere un programma che prende in input una sequenza di numeri e salva tutti i numeri in memoria in locazioni consecutive.

Questo esercizio è estremamente difficile perché richiede di scrivere un programma che si auto-modifica. Esercizi di questa difficoltà NON SARANNO presenti nel compito d'esame.

Soluzione

Potete pensare a locazioni di memoria consecutive come agli elementi di un array in un linguaggio ad alto livello come il Java. Il programma quindi, se lo si guarda ad un livello un po' più alto, consiste fondamentalmente nel riempimento un array con i dati in input. Purtroppo con l'LMC questa è una cosa difficilissima perché il suo linguaggio macchina è molto primitivo. L'unica possibilità di realizzare una tale operazione è fare un programma che si modifica da solo, cosa che in generale è da evitare. Ecco il codice:

```
ciclo:      INP
            BRZ :esci
ista:       STA :inizio
            LDA :ista
            ADD :one
            STA :ista
            BRA :ciclo
esci:       HLT
one:        001
inizio:     000
```

La locazione etichettata con `inizio` è quella nella quale il programma mette il primo input dell'utente, gli altri verranno messi nelle locazioni di memoria successive. Il punto cruciale del programma è la sequenza:

```
            LDA :ista
            ADD :one
            STA :ista
```

La prima istruzione carica nell'accumulatore il contenuto della locazione di memoria etichettata con `ista`. Questa locazione contiene il codice operativo dell'istruzione `STA :inizio`. Siccome la locazione `inizio` è la locazione 9, il suo codice operativo è 309. Il successivo `ADD` incrementa il valore da 309 a 310. Se la pensiamo come istruzione per l'LMC, 310 è il codice operativo dell'istruzione `STA 10`, istruzione che salva il valore dell'accumulatore nella locazione immediatamente successiva a `inizio`. Il codice operativo di `STA 10` viene poi riscritto al posto

del codice operativo di `STA :inizio`.

In questo modo, la volta successiva che l'istruzione di etichetta `ista` verrà eseguita, essa non conterrà più `STA 9`, come prima, ma `STA 10`, per cui il secondo input viene salvato nella locazione 10 e non nella locazione 9. A questo punto, quando rieseguiamo di nuovo la tripletta `LDA :ista`, `ADD :one`, `STA :ista`, l'istruzione STA 10 viene cambiata in STA 11 e così via: ogni volta che si inserisce un nuovo input, questo input viene salvato in memoria, e l'istruzione di salvataggio in ista viene modificata per puntare alla locazione di memoria successiva.

Esercizio 27 (molto molto difficile)

Scrivere un programma che restituisce la lunghezza di una sequenza di numeri, che inizia alla locazione *inizio* e termina con il valore 000.

Soluzione

Anche la soluzione a questo esercizio richiede un programma che si auto-modifica:

```
// Conta la sequenza di numeri che comincia alla locazione
// inizio e termina con zero. Il risultato dovrebbe essere 4.
ciclo:      LDA :inizio
            BRZ :fine
            LDA :conta
            ADD :one
            STA :conta
            LDA :ciclo
            ADD :one
            STA :ciclo
            BRA :ciclo
fine:       LDA :conta
            OUT
            HLT
conta:      0
one:        1
inizio:     45
            34
            22
            11
            0
```