Progetto di programmazione avanzata e parallela

Parte 2 di 2 - Linguaggio Python

Il little man computer (LMC) è un semplice modello di computer creato per scopi didattici. Esso possiede 100 celle di memoria (numerate da 0 a 99) ognuna della quali può contenere un numero da 0 a 999 (estremi inclusi). Il computer possiede un solo registro, detto accumulatore, una coda di input ed una coda di output. LMC possiede un numero limitato di tipi di istruzioni ed un equivalente assembly altrettanto semplificato. Lo scopo del progetto è quello di produrre un simulatore del LMC che dato il contenuto iniziale delle memoria (una lista di 100 numeri) e una sequenza di valori di input simuli il comportamento del LMC e produca il contenuto della coda di output dopo l'arresto del LMC. Un assembler che, dato un file scritto nell'assembly semplificato del LMC produca il contenuto iniziale della memoria. La parte rimanente di questo testo dettaglierà l'architettura del LMCe la sintassi dell'assembly.

Architettura del LMC

Il LMC è composto dalle seguenti componenti:

1. Una memoria di 100 celle numerate tra 0 e 99. Ogni cella può contenere un numero tra 0 e 999. Non viene effettuata alcuna distinzione tra dati e istruzioni: a seconda del momento il contenuto di una certa cella può essere interpretato come un'istruzione (con una semantica che verrà spiegata in seguito) oppure come un dato (e, ad esempio, essere sommato ad un altro valore).
2. Un registro accumulatore (inizialmente zero).
3. Un program counter. Il program counter contiene l'indirizzo dell'istruzione da eseguire ed è inizialmente zero. Se non viene sovrascritto da altre istruzioni (salti condizionali e non condizionali) viene incrementato di uno ogni volta che un'istruzione viene eseguita. Se raggiunge il valore 999 e viene incrementato il valore successivo è zero.
4. Una coda di input. Questa coda contiene tutti i valori forniti in input al LMC, che sono necessariamente numeri compresi tra 0 e 999. Ogni volta che l'LMC legge un valore di input esso viene eliminato dalla coda.
5. Una coda di output. Questa coda è inizialmente vuota e contiene tutti i valori mandati in output dal LMC, che sono necessariamente numeri compresi tra 0 e 999. La coda è strutturata in modo tale da avere in testa il primo output prodotto e come ultimo elemento l'ultimo output prodotto: i valori di output sono quindi in ordine cronologico.
6. Un flag. Ovvero un singolo bit che può essere acceso o spento. Inizialmente esso è

zero (flag assente). Solo le istruzioni aritmetiche modificano il valore del flag e, in particolare, un flag a uno (flag presente) indica che l'ultima operazione aritmetica eseguita ha prodotto un risultato maggiore di 999 o minore di 0. Un flag assente indica invece che il valore prodotto dall'ultima operazione aritmetica ha prodotto un risultato compreso tra 0 e 999.

La seguente tabella rappresenta come le istruzioni presenti in memoria debbano essere interpretate:

Valore	Nome Istruzione	Significato
1xx	Addizione	Somma il contenuto della cella di memoria xx con il valore contenuto nell'accumulatore e scrive il valore risultante nell'accumulatore. Il valore salvato nell'accumulatore è la somma modulo 1000. Se la somma non supera 1000 il flag è impostato ad assente, se invece raggiunge o supera 1000 il flag è impostato a presente.
2xx	Sottrazione	Sottrae il contenuto della cella di memoria xx dal valore contenuto nell'accumulatore e scrive il valore risultante nell'accumulatore. Il valore salvato nell'accumulatore è la differenza modulo 1000. Se la differenza è inferiore a zero il flag è impostato a presente, se invece è positiva o zero il flag è impostato ad assente.
3xx	Store	Salva il valore contenuto nell'accumulatore nella cella di memoria avente indirizzo xx. Il contenuto dell'accumulatore rimane invariato.
5xx	Load	Scrive il valore contenuto nella cella di memoria di indirizzo xx nell'accumulatore. Il contenuto della cella di memoria rimane invariato.
6xx	Branch	Salto non condizionale. Imposta il valore del program counter a xx.
7xx	Branch if zero	Salto condizionale. Imposta il valore del program counter a xx solamente se il contenuto dell'accumulatore è zero e se il flag è assente.
8xx	Branch if positive	Salto condizionale. Imposta il valore del program counter a xx solamente se il flag è assente.
901	Input	Scrive il contenuto presente nella testa della coda in input nell'accumulatore e lo rimuove dalla coda di input.
902	Output	Scrive il contenuto dell'accumulatore alla fine della coda di output. Il contenuto dell'accumulatore rimane invariato.
0xx	Halt	Termina l'esecuzione del programma. Nessuna ulteriore istruzione viene eseguita.

Alcuni devi valori non corrispondono a nessuna istruzione. Ad esempio tutti i numeri tra 400 e 499 non hanno un corrispettivo. Questo significa che corrispondono a delle istruzioni non valide (illegal instructions) e che LMC si fermerà con una condizione di errore.

Quindi, ad esempio, se l'accumulatore contiene il valore 42, il program counter ha valore 10 ed il contenuto della cella numero 10 è 307, il LMC eseguirà una istruzione di store, dato che il valore è nella forma 3xx. In particolare, il contenuto dell'accumulatore, ovvero 42, verrà scritto nella cella di memoria numero 7 (dato che xx corrisponde a 07). Il program counter verrà poi incrementato di uno, assumendo il valore 11. La procedura

verrà ripetuta finché non verrà incontrata un'istruzione di halt o un'istruzione non valida.

Assembly per LMC

Oltre a fornire una simulazione per il LMC si deve essere anche in grado di passare da un programma scritto in codice assembly per LMC (che verrà successivamente dettagliato) al contenuto iniziale della memoria del LMC, convertendo quindi il codice assembly in codice macchina.

Nel file sorgente assembly ogni riga contiene al più una etichetta ed una istruzione. Ogni istruzione corrisponde al contenuto di una cella di memoria. La prima istruzione assembly presente nel file corrisponderà al valore contenuto nella cella 0, la seconda al valore contenuto nella cella 1, e così via.

Istruzione	Valori possibili per xx	Significato
ADD xx	Indirizzo o etichetta	Esegui l'istruzione di addizione tra l'accumulatore e il valore contenuto nella cella indicata da xx
SUB xx	Indirizzo o etichetta	Esegui l'istruzione di sottrazione tra l'accumulatore e il valore contenuto nella cella indicata da xx
STA xx	Indirizzo o etichetta	Esegue una istruzione di store del valore dell'accumulatore nella cella indicata da xx
LDA xx	Indirizzo o etichetta	Esegue una istruzione di load dal valore contenuto nella cella indicata da xx nell'accumulatore
BRA xx	Indirizzo o etichetta	Esegue una istruzione di branch non condizionale al valore indicato da xx
BRZ xx	Indirizzo o etichetta	Esegue una istruzione di branch condizionale (se l'accumulatore è zero e non vi è il flag acceso) al valore indicato da xx.
BRP xx	Indirizzo o etichetta	Esegue una istruzione di branch condizionale (se non vi è il flag acceso) al valore indicato da xx.
INP	Nessuno	Esegue una istruzione di input
OUT	Nessuno	Esegue una istruzione di output
HLT	Nessuno	Esegue una istruzione di halt
DAT xx	Numero	Memorizza nella cella di memoria corrispondente a questa istruzione assembly il valore xx
DAT	Nessuno	Memorizza nella cella di memoria corrispondente a questa istruzione assembly il valore 0 (equivalente all'istruzione DAT 0).

Ogni riga del file assembly con una istruzione può contenere prima dell'istruzione una etichetta, ovvero una stringa di caratteri usata per identificare la cella di memoria dove verrà salvata l'istruzione corrente. Le etichette possono poi essere utilizzate al posto degli indirizzi (ovvero numeri tra 0 e 99) all'interno del sorgente. Si veda, ad esempio:

Assembly	Cella di memoria	Codice macchina
`LABEL ADD 4`	0	104
`BRA LABEL`	1	600

In questo caso l'etichetta presente nella prima riga assume il valore 0 tutte le volte che appare come argomento di un'altra istruzione. Per un esempio già completo si osservi:

Assembly	Cella di memoria	Codice macchina
`LOAD  LDA 0`	0	500
`OUT`	1	902
`SUB ONE`	2	208
`BRZ ONE`	3	708
`LDA LOAD`	4	500
`ADD ONE`	5	108
`STA LOAD`	6	300
`BRA LOAD`	7	600
`ONE  DAT 1`	8	1

In questo caso ci sono due etichette (LOAD e ONE) che assumono valore 0 e 8 rispettivamente. Questo corrisponde alla cella di memoria dove la versione convertita in codice macchina dell'istruzione assembly presente nella stessa riga è stata salvata. Si noti inoltre come le etichette possano essere utilizzate prima di venire definite. Se una riga contiene una doppia barra (//) tutto il resto della riga deve venire ignorato e considerato come commento. L'assembly è inoltre case-insensitive e possono esserci uno o più spazi tra etichetta e istruzione e etichetta e argomento. Le seguenti istruzioni sono quindi tutte equivalenti:

```
ADD 3
Add 3
add 3 // Questo è un commento
ADD 3 // Aggiunte il valore della cella 3 all'accumulatore
aDD    3
```

Si consiglia di sfruttare i codici di esempio allegati al progetto per testare e comprendere meglio il funzionamento dell'assembly per LMC.

Classi da implementare

1. Una classe `Assembler` che permetta, dato un file di assembly per LMC di produrre lo stato iniziale della memoria dell'LMC come lista di interi.
2. Una classe `LMC` che permetta di eseguire (sia fino al termine che una istruzione alla

volta) un programma di LMC. Deve essere possibile ispezionare lo stato interno dell'LMC (memoria, program counter, etc.)

Documentate adeguatamente come compilare ed eseguire un programma per LMC.

Requisiti

- In caso di errore il codice deve generare delle eccezioni coerenti con il tipo di condizione che ha provocato l'errore. È possibile definire delle eccezioni apposite o usare quelle predefinite (si sconsiglia sollevare eccezioni troppo generiche, come `Exception`).
- Ogni funzione rilevante (i.e., non di 2-3 righe) deve essere adeguatamente commentata spiegando che compito svolge, che input richiede e che output genera.
- Ogni file .py deve contenere all'inizio come commento il proprio nome, cognome e numero di matricola.

Consegna

Le date di consegna sono **sempre** esattamente 7 giorni prima dell'appello *orale*.
Guardate teams per i giorni esatti.

Per appelli successivi verranno comunicate le date di consegna. Non inviate al di fuori delle date di consegna.

Istruzioni per la consegna:

- Preparare il progetto e salvarlo come un file `.zip` o `tar.gz` contenente tutti i file *sorgenti* necessari. **NON** devono essere allegati i file binari. Può essere allegato (ed è anzi consigliato) un file `README.txt` o `README.md` con istruzioni e note.
- Inviare una mail a `lmanzoni@units.it` con il seguente oggetto:
  `[Consegna Progetto Programmazione Avanzata e Parallela] Nome Cognome Matricola`
  usando la vostra email istituzionale. Dovete chiaramente allegare il progetto, per il testo della mail non vi sono vincoli.

Note importanti

- Questo progetto è relativo solamente alla parte in Python del corso. Per sostenere la parte di progetto del corso è necessario consegnare sia la parte in C (parte 1) che la parte in Python (parte 2).
- Sebbene appaia ovvio, quando viene chiesto di inserire Nome, Cognome e Matricola, dovete inserire il vostro nome, cognome e numero di matricola a non, letteralmente, il testo "Nome", "Cognome" e "Matricola".
- Il progetto rimane valido anche per gli appelli successivi, non è necessario inviarlo nuovamente.

- Ogni consegna successiva annulla quella precedente, si sconsiglia però di inviare decine di versioni durante il periodo di consegna.
- Devono essere sempre consegnate entrambe le parti, **NON** solo una delle due.
- Verificate se il progetto è stato aggiornato con chiarimenti, questo avviene se diverse persone fanno osservazioni che si richiede siano da chiarire per tutti.