Linguaggi e Traduttori
Anno Accademico 2014–2015
Prof. Giovanni Pighizzini

Note sulla macchina a stack

20 Maggio 2015

Questo documento contiene alcune note sulla macchina virtuale utilizzata per gli esempi di generazione di codice, nonché sulle classi Java che permettono di utilizzarla.

1 Macchina virtuale

La macchina ha un'area di memoria per i dati (organizzata a stack) e un'area per il codice. Vi sono inoltre due registri: lo *stack pointer* (`SP`) che indica la cima dello stack e il *program counter* (`PC`) che indica l'istruzione che deve essere eseguita. Le istruzioni vengono caricate nell'area di codice, dal file in cui si trovano, a partire dall'indirizzo 0. Il program counter è inizializzato a 0, pertanto l'esecuzione inizia da tale indirizzo. Inoltre, all'inizio dell'esecuzione lo stack è vuoto (esso verrà riempito a partire dall'indirizzo zero).

Le istruzioni della macchina possono avere al più un operando. Le istruzioni prive di operandi sono rappresentate con un `int`, le altre con due.

Segue l'elenco di istruzioni con il relativo significato. Salvo differente indicazione, ogni istruzione incrementa automaticamente `PC` (più precisamente, l'incremento è di 1 per le istruzioni prive di operando, di 2 per le altre, in modo che `PC` si riferisca automaticamente all'istruzione successiva). L'eventuale operando (un intero) viene indicato con `op`. Tra parentesi è indicato il nome esteso e la costante definita nella classe `Macchina` per indicare il nome dell'istruzione (se diversa dal codice mnemonico).

Istruzioni di trasferimento dati

- `PUSH op`
  Carica in cima alla pila il valore contenuto all'indirizzo `op`. Incrementa `SP`.
- `PUSH*` (push indiretta `PUSHIND`)
  Interpreta il valore contenuto in cima alla pila come indirizzo di memoria; sostituisce tale valore con il contenuto della cella che si trova all'indirizzo di memoria da esso specificato.
- `PUSH= op` (push immediata `PUSHIMM`)
  Carica in cima alla pila il valore `op` della memoria. Incrementa `SP`.
- `POP op`
  Copia il valore che si trova in cima alla pila all'indirizzo `op` di memoria. Decrementa `SP`.

- `POP*` (pop indiretta `POPIND`)
  Utilizza i due numeri più in alto nella pila, interpretando quello in cima come valore e quello immediatamente sotto come indirizzo: copia il valore che si trova in cima alla pila (cioè in posizione `SP`) all'indirizzo di memoria dato dal valore che si trova sotto di esso nella pila (cioè che si trova in posizione `SP`-1). Decrementa `SP` di due.

- `POP=` (pop immediata `POPIMM`)
  Elimina il valore che si trova in cima alla pila (cioè decrementa `SP`).

Istruzioni aritmetiche

- `ADD`
  `SUB`
  `MUL`
  `DIV`
  Sostituiscono i due elementi in cima alla pila con la loro somma, differenza, prodotto e quoziente, rispettivamente. Alla fine dell'operazione `SP` risulta decrementato. Si noti che l'operando sinistro è il valore che all'inizio si trova in posizione `SP`-1, mentre quello destro è il valore che all'inizio si trova in posizione `SP`.

Istruzioni di controllo e di salto

Nella spiegazione delle seguenti istruzioni le modifiche a `PC` sono indicate esplicitamente.

- `JUMP op` (salto incondizionato)
  Assegna a `PC` il valore `op`.

- `JZERO op` (salto condizionato per zero)
  Controlla il valore contenuto in cima allo stack. Se è zero assegna a `PC` il valore `op`, altrimenti incrementa normalmente `PC`. In entrambi casi decrementa `SP`.

- `JNZERO op` (salto condizionato per diverso da zero)
  `JGTZ op` (salto condizionato per maggiore di zero)
  `JGEZ op` (salto condizionato per maggiore o uguale a zero)
  `JLTZ op` (salto condizionato per minore di zero)
  `JLEZ op` (salto condizionato per minore o uguale a zero)
  Analoghe alla precedente, con le condizioni indicate.

- `JUMP*` (jump indiretta `JUMPIND`)
  Assegna a `PC` il valore che si trova in cima alla pila. Decrementa `SP`.

Altre istruzioni

- `INPUT`
  Attende l'inserimento di un intero da tastiera e lo pone in cima alla pila. Incrementa `SP`.

- `INPUTCH`
  Pone in cima allo stack il codice Unicode di un carattere letto da tastiera. La lettura avviene per righe. La prima volta che viene eseguita questa istruzione, la macchina attende l'inserimento da tastiera di una riga di testo e utilizza il primo carattere della stringa ottenuta. Nelle esecuzioni successive dell'istruzione vengono utilizzati i caratteri seguenti. Al raggiungimento della fine della stringa, viene fornito il carattere di codice 0. A questo punto, l'esecuzione di un'ulteriore istruzione `INPUTCH` provoca la lettura di una nuova stringa. Ad esempio, se viene inserita la stringa "`ciao`" le prime 4 istruzioni `INPUTCH` pongono sullo stack, i codici Unicode dei caratteri della stringa, la quinta istruzione il valore `0`. Una successiva istruzione `INPUTCH`

provoca la lettura di una nuova stringa e pone sullo stack il codice del primo carattere (0 se viene letta la stringa vuota). Quando viene eseguita un'istruzione `INPUT`, eventuali caratteri non utilizzati di un stringa letta in precedenza vengono eliminati. Pertanto una successiva `INPUTCH` provocherà la lettura di una nuova stringa.

- `OUTPUT`
  Visualizza l'intero che si trova in cima alla pila. Decrementa `SP`.
- `OUTPUTCH`
  Visualizza il carattere il cui codice Unicode si trova in cima alla pila. Decrementa `SP`.
- `MOVESP`
  Assegna a `SP` il valore che si trova in cima alla pila.
- `PUSHPC`
  Memorizza in cima alla pila il valore di `PC` (già incrementato).
- `PUSHSP`
  Incrementa `SP` e ne memorizza il valore in cima alla pila.
- `PUSHSIZE`
  Memorizza in cima alla pila la dimensione della pila stessa. Pertanto, se tale dimensione è n, gli indirizzi di memoria disponibili vanno da 0 a $n-1$.
- `HALT`
  Termina l'esecuzione.

2 Come generare il codice

Per generare il codice si utilizza la classe `Codice`. ll costruttore di tale classe riceve come parametro il nome del file su sui verrà scritto il codice. Il file verrà creato solo alla fine. A tale scopo si deve richiamare esplicitamente il metodo

```
public void fineCodice() throws IOException
```

Le istruzioni, con i relativi operandi, possono essere generate in ordine sequenziale (in posizioni numerate a partire da zero) utilizzando i metodi:

- `public void genera(int istr)`
  Aggiunge l'istruzione specificata tramite il parametro (utile per istruzioni prive di operandi).
- `public void genera(int istr, int oper)`
  Aggiunge l'istruzione specificata tramite il primo parametro con l'operando specificato dal secondo (utile per istruzioni con un operando).
- `public int generaParziale(int istr)`
  Aggiunge l'istruzione specificata tramite il parametro lasciando uno spazio per inserire successivamente l'operando. Il valore restituito è la posizione nel codice dove è stato lasciato lo spazio. Questa istruzione è utile per le situazioni in cui non si conosca il valore dell'operando (ad esempio dovendo generare i salti in avanti durante la compilazione dei cicli). L'operando potrà essere inserito successivamente ricorrendo al metodo `completaIstruzione`.

Per i nomi delle istruzioni è bene utilizzare le costanti definite nella classe `Macchina`, come ad esempio `Macchina.PUSH`, `Macchina.ADD`, ecc. Non viene effettuato alcun controllo sull'uso corretto dei parametri.

La classe `Codice` fornisce inoltre i metodi:

- `public void completaIstruzione(int posizione, int valore)`
  Assegna alla posizione di codice indicata dal primo parametro il valore indicato dal secondo. È utile per completare istruzioni generate con `generaParziale`.
- `public int indirizzoProssimaIstruzione()`
  Restituisce la posizione nella quale verrà generata la prossima istruzione.

3 Come utilizzare la macchina

Le classi `Macchina` e `Codice` si trovano in un package di nome `lt.macchina` (che dovrà essere dunque raggiungibile mediante la variabile di ambiente `CLASSPATH`). Il modo più semplice per utilizzare la macchina è quello di richiamarla impartendo il comando:

```
java lt.macchina.Macchina ...argomenti...
```

Gli argomenti che iniziano con il carattere `'-'` indicano opzioni. Quelle disponibili sono:

- `d`
  Debug: ogni istruzione viene visualizzata prima di essere eseguita.
- `l`
  List: viene visualizzato l'intero codice prima di essere eseguito.
- `L`
  List: viene visualizzato l'intero codice, ma non viene eseguito.

Ogni altro argomento viene considerato come nome di file contenente il codice eseguibile; se ne viene indicato più di uno, verrà eseguito solo l'ultimo, se non ne viene indicato nessuno, verrà eseguito un file di nome `"eseguibile"`.

È possibile inoltre richiamare la macchina da altre applicazioni. A tale scopo si può utilizzare:

- `public Macchina(String s, boolean debug) throws IOException`
  Predispone la macchina per lavorare con il codice prelevato dal file il cui nome si trova in `s`. Viene effettuato il debug se e solo se il secondo argomento è `true`.
- `public Macchina(String s) throws IOException`
  Equivalente al precedente con secondo argomento `false`.
- `public void esegui() throws IOException`
  Manda in esecuzione la macchina.
- `public String toString()`
  Restituisce una stringa che mostra il codice eseguibile.
- `public void visualizza()`
  Visualizza il codice eseguibile.

La classe contiene le costanti pubbliche corrispondenti alle istruzioni del programma (come `Macchina.PUSH`, `Macchina.PUSHIMM`, ecc, importabili mediante la direttiva `import static lt.macchina.Macchina.*`).
Le classi `Macchina` e `Codice` devono essere importate, mediante le opportune direttive, nelle classi che le utilizzano.

4 Un assemblatore per la macchina

La macchina carica il programma da un file in cui istruzioni e operandi sono codificati come interi. Il file può essere, ad esempio, generato da un compilatore, usando la classe `Codice`. È possibile anche scrivere sorgenti nel linguaggio della macchina, come nell'esempio seguente:

```
//legge una stringa da input e la visualizza sullo schermo
        PUSH= 63
        OUTPUTCH    //visualizza un prompt (punto interrogativo)
inizio: INPUTCH
        POP 0
        PUSH 0
        PUSH 0      //duplica il valore in cima allo stack
        JZERO fine
        OUTPUTCH
        JUMP inizio
fine:   PUSH= 10
        OUTPUTCH   //ritorno a capo
        HALT
```

E' possibile utilizzare etichette e commenti, come indicato nell'esempio. I sorgenti possono essere tradotti in eseguibili destinati alla macchina virtuale utilizzando la classe `Assembla`, fornendo il comando

```
java lt.macchina.Assembla <sorgente> <destinazione>
```

Se `<destinazione>` non viene specificato, il nome di default è `"eseguibile"`.[1]
Altro esempio:

```
// fattoriale di n (solo per n > 0!)
        PUSH= 5
        MOVESP      //per occupare qualche cella in fondo allo stack per le variabili
        PUSH= 63
        OUTPUTCH    //visualizza un prompt (punto interrogativo)
        INPUT       //numero in ingresso
        POP 0       //indirizzo 0: variabile n
        PUSH= 1
        POP   1     //indirizzo 1: risultato
alfa:   PUSH  0
        PUSH  1
        MUL
        POP   1      //risultato = risultato * n
        PUSH  0
        PUSH= 1
        SUB
        POP   0      // n = n - 1
        PUSH  0
        JNZERO alfa
        PUSH  1     //risultato
        OUTPUT
        PUSH= 10
        OUTPUTCH   //ritorno a capo
        HALT
```

[1] Si presti attenzione a eventuali estensioni nei nomi dei file che, se presenti, *devono* essere indicate.