Macchina Astratta

Andrea Zanelli

28 dicembre 2008

Indice

1 Introduzione

In questo documento viene elencato e descritto l'instruction set della macchina astratta e viene illustrata la sua struttura di funzionamento. In entrambi i casi è basata sulla Java Virtual Machine (JVM). Per l'instruction set è stato preso un sottoinsieme di quello della JVM, percui un programma funzionante su questa macchina astratta funziona anche sulla JVM, traducendo però gli mnemonici in bytecode con un assembler come, per esempio, l'Oolong scritto da Joshua Engel. La struttura, in modo simile alla JVM, è composta da uno stack di sistema, in cui vengono messi i record di attivazione (RdA) di ogni funzione che viene eseguita, e da una area-programma dove vengono memorizzate le istruzioni da eseguire. Ogni RdA è quindi composto da un program counter (PC) che punta all'istruzione da eseguire, da uno stack degli operandi e da un array delle variabili locali.

2 Direttive

Le direttive vengono utilizzate per definire l'inizio e la fine del programma, per dichirare le variabili globali e le funzioni.

2.1 Tabella direttive

Direttiva	Descrizione
`.class public Main`	Inizio del programma
`.super java/lang/Object`	Inizio del programma (da mettere subito dopo `.class public Main`)
`.end class`	Fine del programma
`.field public static` *f d*	Variabile globale *f*
`.method public static` *m d*	Inizio della funzione *m*
`.end method`	Fine della funzione

2.2 Inizio e fine programma

Un programma ha inizio dopo le due seguenti direttive

```
.class public Main
.super java/lang/Object
```

e finisce con la direttiva

```
.end class
```

Fra queste direttive di inizio e fine del programma vengono definite le funzioni e le variabili globali. Ogni scritta prima di `.class public Main` e dopo `.end class` è considerata errore.

Esempio

```
.class public Main
.super java/lang/Object
  ; funzioni e variabili globali
.end class
```

2.3 Variabili globali

Le variabili globali si definiscono con la direttiva

```
.field public static fieldname descriptor
```

dove *`fieldname`* è il nome della variabile e *`descriptor`* è il tipo della variabile (vedere tabella dei tipi: 3.1).

Esempio

Variabile globale di nome `numberOfElements` e di tipo `int`:

```
.field public static numberOfElements I
```

2.4 Funzioni

Per la dichiarazione di una funzione e per indicarne l'inizio si usa la direttiva

```
.method public static methodname descriptor
```

dove *`methodname`* è il nome della funzione e *`decriptor`* descrive gli argomenti e il tipo di ritorno della funzione (vedere la tabella dei tipi: 3.1).
Per indicare la fine della definizione della funzione si usa la direttiva

```
.end method
```

Fra le direttive `.method` e `.end method` vi sono le istruzioni della funzione (vedere le tabelle delle istruzioni nel paragrafo 4.1).

Esempio

Funzione di nome `icalc` che prende come argomenti quattro interi e ritorna un intero

```
.method public static icalc(IIII)I
  ; serie di istruzioni
.end method
```

2.5 Funzione inizializzatore

Se esiste è la prima funzione ad essere eseguita ed è utilizzata per inizializzare le variabili globali prima di eseguire la funzione principale; ha nome `<clinit>`, non prende argomenti e ha tipo di ritorno void:

```
.method public static <clinit> ()V
```

Se nel testo del programma non c'è la funzione `<clinit>` viene eseguita direttamente la funzione principale.

Esempio

```
.method public static <clinit> ()V
  ; serie di istruzioni per inizializzare
  ; le variabili globali
  return
.end method
```

2.6 Funzione principale

È la funzione principale, la prima ad essere eseguita (se non esiste la funzione inizializzatore) e da cui inizia il programma, ha nome `main`:

```
.method public static main([Ljava/lang/String;)V
```

Se nel testo del programma non c'è la funzione `main` viene segnalato un errore e il programma non può essere eseguito.

Esempio

```
.method public static main([Ljava/lang/String;)V
  return
.end method
```

3 Tipi

Nella macchina astratta sono implementati i seguenti tipi:

- `int`
- `long`
- `short`
- `char`

Si possono usare inoltre le "costanti stringa" (`String`) con le seguenti caratteristiche: sono racchiuse da virgolette ("), con l'istruzione `ldc_w "stringa"` si mette la costante stringa in cima allo stack, si possono stampare su output con la funzione di stampa e argomento `Ljava/lang/String;`, si possono leggere da standard input terminate da un a-capo, **non** vi sono altre istruzioni di alcun tipo sulle stringhe, **non** si possono assegnare a variabili globali o locali, **non** possono essere passate come parametro a funzioni o usate come valore di ritorno.

3.1 Tabella tipi

Qui sotto la tabella contenente i tipi disponibili nel linguaggio della macchina astratta, con il descrittore corrispondente, cioè il simbolo che identifica il tipo, da mettere nelle funzioni (per gli argomenti e il tipo di ritorno) e nelle variabili globali (per il tipo di variabile).

Tipo	Descrittore
`char`	`C`
`int`	`I`
`long`	`J`
`short`	`S`
`void`	`V`

3.2 Rappresentazioni e conversioni

Le conversioni da un tipo più piccolo ad uno più grande (promozioni), per esempio da `int` a `long`, sono fatte normalmente, mantenendo inalterato il valore. Le conversioni da un tipo più grande, diciamo `T`, ad uno più piccolo, diciamo `t`, per esempio da `int` a `short`, vengono fatte nel seguente modo: se il valore di `T` è compreso tra i valori possibili di `t`, allora la conversione avviene normalmente; se il valore di `T` è maggiore del più grande valore di `t`, allora si ricomincia (in modo ciclico) dal valore più piccolo di `t`, quindi se, ad esempio, si vuol convertire l'`int` +32769 in un `short`, prenderà il valore -32767, allo stesso modo l'`int` 98304 prenderà il valore -32768; se il valore di `T` è minore del più piccolo valore di `t`, allora si ricomincia (in modo ciclico) dal valore più grande di `t`, quindi se, ad esempio, si vuol convertire l'`int` -32769 in un `short`, prenderà il valore +32767.

Tabella rappresentazioni

Qui sotto la tabella con la rappresentazione binaria ed i valori che può assumere ogni tipo.

Tipo	Rappresentazione	da	a
`int`	32 bit, con segno	-2.147.483.648	+2.147.483.647
`short`	16 bit, con segno	-32.768	+32.767
`long`	64 bit, con segno	-2^{63}	$+2^{63}-1$
`char`	16 bit, senza segno	0	+65.535

I tipi `int`, `short` e `char` occupano uno slot sullo stack ed un solo registro nelle variabili locali, il tipo `long` occupa due slot sullo stack e due registri (consecutivi) nelle variabili locali.

4 Istruzioni

Chiavi di lettura per le istruzioni:

`a`	Slot in cima allo stack (top). Può contenere `int` o costanti stringa
`b`	Secondo slot dello stack. Può contenere `int` o costanti stringa
`c`	Terzo slot dello stack. Può contenere `int` o costanti stringa
`d`	Quarto slot dello stack. Può contenere `int` o costanti stringa
`ab`	Primo e secondo slot dello stack. Usato per tipi `long`
`cd`	Terzo e quarto slot dello stack. Usato per tipi `long`

4.1 Tabelle istruzioni

Di seguito tutte le istruzioni divise per categoria con una breve descrizione. Per maggiori dettagli si rimanda al paragrafo 4.2.

Aritmetiche

Istruzione	Argomenti	Descrizione
`iadd`		Somma `int` (`b+a`)
`idiv`		Divisione `int` (`b/a`)
`imul`		Moltiplicazione `int` (`b*a`)
`ineg`		Negazione `int` (`-a`)
`irem`		Resto divisione `int` (`b%a`)
`ishl`		Shift `int` a sinistra (`b « a`)
`ishr`		Shift `int` a destra (`b » a`)
`isub`		Sottrazione `int` (`b-a`)
`ladd`		Somma `long` (`cd+ab`)
`ldiv`		Divisione `long` (`cd/ab`)
`lmul`		Moltiplicazione `long` (`cd*ab`)
`lneg`		Negazione `long` (`-ab`)
`lrem`		Resto divisione `long` (`cd%ab`)
`lshl`		Shift `long` a sinistra (`bc « a`)
`lshr`		Shift `long` a destra (`bc » a`)
`lsub`		Sottrazione `long` (`cd-ab`)

Costanti

Istruzione	Argomenti	Descrizione
`ldc_w`	`x`	Mette `x` in cima allo stack (`int` o `String`)
`ldc2_w`	`x`	Mette `x` in cima allo stack (`long`)
`sipush`	`n`	Mette `n` in cima allo stack (`short`)

Controllo

Istruzione	Argomenti	Descrizione
`goto`	`label`	Salta all'istruzione contrassegnata con `label`
`if_icmpeq`	`label`	Salta a `label` se `b == a`
`if_icmpge`	`label`	Salta a `label` se `b >= a`
`if_icmpgt`	`label`	Salta a `label` se `b > a`
`if_icmple`	`label`	Salta a `label` se `b <= a`
`if_icmplt`	`label`	Salta a `label` se `b < a`
`if_icmpne`	`label`	Salta a `label` se `b != a`
`ifeq`	`label`	Salta a `label` se `a == 0`
`ifge`	`label`	Salta a `label` se `a >= 0`
`ifgt`	`label`	Salta a `label` se `a > 0`
`ifle`	`label`	Salta a `label` se `a <= 0`
`iflt`	`label`	Salta a `label` se `a < 0`
`ifne`	`label`	Salta a `label` se `a != 0`
`ireturn`		Ritorna un `int` alla funzione chiamante
`lcmp`		Confronto tra `long`
`lreturn`		Ritorna un `long` alla funzione chiamante
`nop`		Non fa' niente
`return`		Ritorna al metodo chiamante

Conversione di tipi

Istruzione	Argomenti	Descrizione
`i2c`		Converte l'`int` in `a` in `char`
`i2s`		Converte l'`int` in `a` in `short`
`i2l`		Converte l'`int` in `a` in `long`
`l2i`		Converte il `long` in `ab` in `int`

Operazioni sullo stack

Istruzione	Argomenti	Descrizione
`dup`		Duplica `a`
`dup2`		Duplica `ab`
`pop`		Rimuove `a`
`pop2`		Rimuove `ab`
`swap`		Scambia `a` con `b`

Funzioni e variabili globali

Istruzione	Argomenti	Descrizione
`getstatic`	`Main/`*`f`* *`d`*	Mette sullo stack *`f`* *`d`* (variabile globale)
`invokestatic`	`Main/`*`m`* *`d`*	Chiama la funzione *`m`* *`d`*
`putstatic`	`Main/`*`f`* *`d`*	Memorizza in *`f`* *`d`* (variabile globale)

Variabili

Istruzione	Argomenti	Descrizione
`iload`	`n`	Mette sullo stack la variabile locale `n` (`int`)
`istore`	`n`	Mette `a` nella variabile locale `n`
`lload`	`n`	Mette sullo stack la variabile locale `n` (`long`)
`lstore`	`n`	Mette `ab` nella variabile locale `n`

4.2 Dettagli istruzioni

`dup`: duplica e mette sullo stack il valore di tipo `int` in cima allo stack (`a`).

`dup2`: duplica e mette sullo stack il valore di tipo `long` in cima allo stack (`ab`).

`getstatic Main/`*`f`* *`d`*: mette in cima allo stack il valore della variabile globale di nome *`f`* e tipo *`d`* (vedere la tabella dei tipi: 3.1).

`goto label`: salta all'istruzione etichettata con `label`.

`i2c`: preleva l'`int` in cima allo stack (`a`), lo converte in `char` e mette il risultato in cima allo stack.

`i2l`: preleva l'`int` in cima allo stack (`a`), lo converte in `long` e mette il risultato in cima allo stack.

`i2s`: preleva l'`int` in cima allo stack (`a`), lo converte in `short` e mette il risultato in cima allo stack.

`iadd`: preleva i primi due `int` in cima allo stack (`a` e `b`), esegue la somma tra il secondo e il primo (`b+a`), mette il risultato (`int`) sullo stack.

`idiv`: preleva i primi due `int` in cima allo stack (`a` e `b`), esegue la divisione tra il secondo e il primo (`b/a`), mette il risultato (`int`) sullo stack.

`if_icmpeq label`: preleva i primi due `int` in cima allo stack (`a` e `b`), se il secondo è uguale al primo (`b == a`) salta all'istruzione etichettata con `label`, altrimenti esegue l'istruzione successiva.

`if_icmpge label`: preleva i primi due `int` in cima allo stack (`a` e `b`), se il secondo è maggiore o uguale al primo (`b >= a`) salta all'istruzione etichettata con `label`, altrimenti esegue l'istruzione successiva.

`if_icmpgt label`: preleva i primi due `int` in cima allo stack (`a` e `b`), se il secondo è maggiore del primo (`b > a`) salta all'istruzione etichettata con `label`, altrimenti esegue l'istruzione successiva.

`if_icmple label`: preleva i primi due `int` in cima allo stack (`a` e `b`), se il secondo è minore o uguale al primo (`b <= a`) salta all'istruzione etichettata con `label`, altrimenti esegue l'istruzione successiva.

`if_icmplt label`: preleva i primi due `int` in cima allo stack (`a` e `b`), se il secondo è minore del primo (`b < a`) salta all'istruzione etichettata con `label`, altrimenti esegue l'istruzione successiva.

`if_icmpne label`: preleva i primi due `int` in cima allo stack (`a` e `b`), se il secondo è diverso dal primo (`b != a`) salta all'istruzione etichettata con `label`, altrimenti esegue l'istruzione successiva.

`ifeq label`: preleva l'`int` in cima allo stack (`a`), se è uguale a zero (`a == 0`) salta all'istruzione etichettata con `label`, altrimenti esegue l'istruzione successiva.

`ifge label`: preleva l'`int` in cima allo stack (`a`), se è maggiore o uguale a zero (`a >= 0`) salta all'istruzione etichettata con `label`, altrimenti esegue l'istruzione successiva.

`ifgt label`: preleva l'`int` in cima allo stack (`a`), se è maggiore di zero (`a > 0`) salta all'istruzione etichettata con `label`, altrimenti esegue l'istruzione successiva.

`ifle label`: preleva l'`int` in cima allo stack (`a`), se è minore o uguale a zero (`a <= 0`) salta all'istruzione etichettata con `label`, altrimenti esegue l'istruzione successiva.

`iflt label`: preleva l'`int` in cima allo stack (`a`), se è minore di zero (`a < 0`) salta all'istruzione etichettata con `label`, altrimenti esegue l'istruzione successiva.

`ifne label`: preleva l'`int` in cima allo stack (`a`), se è diverso da zero (`a != 0`) salta all'istruzione etichettata con `label`, altrimenti esegue l'istruzione successiva.

`iload n`: mette in cima allo stack l'`int` memorizzato nella variabile locale `n`.

`imul`: preleva i primi due `int` in cima allo stack (`a` e `b`), esegue il prodotto tra il secondo e il primo (`b*a`), mette il risultato (`int`) sullo stack.

`ineg`: preleva l'`int` in cima allo stack (`a`), lo nega (`-a`) e mette il risultato sullo stack.

`invokestatic Main/m d`: chiama la funzione di nome *m* con descrittore *d* (il descrittore contiene i tipi degli argomenti e il tipo di ritorno della funzione, vedere la tabella dei tipi: 3.1) passandogli come parametri (se previsti dalla funzione) i valori sullo stack, in modo che il valore in cima allo stack corrisponde all'ultimo parametro richiesto dalla funzione, il secondo nello stack corrisponde al penultimo parametro, e così via. Nel corpo della funzione chiamata, i parametri passati vengono inseriti nelle prime variabili locali, quindi il primo parametro sarà nella variabile locale `0`, il secondo in `1` (supponendo che il primo occupava un solo posto nelle variabili), e così via. Al termine dell'esecuzione della funzione, se ha un valore di ritorno viene messo in cima allo stack del chiamante.

`irem`: preleva i primi due `int` in cima allo stack (`a` e `b`), calcola il resto della divisione tra il secondo e il primo (`b%a`) e mette il risultato (`int`) sullo stack.

`ireturn`: preleva l'`int` in cima allo stack (`a`), termina l'esecuzione della funzione, torna il controllo alla funzione chiamante e mette l'`int` in cima allo stack. Da utilizzare nelle funzioni che hanno come tipo di ritorno `I`, `S` o `C`.

`ishl`: preleva i primi due `int` in cima allo stack (`a` e `b`), calcola lo shift a sinistra del secondo valore con il primo (`b « a`) e mette il risultato (`int`) sullo stack.

`ishr`: preleva i primi due `int` in cima allo stack (`a` e `b`), calcola lo shift a destra del secondo valore con il primo (`b » a`) e mette il risultato (`int`) sullo stack.

`istore n`: preleva l'`int` in cima allo stack (`a`), e memorizza il valore nella variabile locale `n`.

`isub`: preleva i primi due `int` in cima allo stack (`a` e `b`), esegue la sottrazione tra il secondo e il primo (`b-a`), mette il risultato (`int`) sullo stack.

`l2i`: preleva il `long` in cima allo stack `ab`, lo converte in `int` e mette il risultato in cima allo stack.

`ladd`: preleva i primi due `long` in cima allo stack (`ab` e `cd`), esegue la somma tra il primo e il secondo (`ab+cd`) e mette il risultato (`long`) sullo stack.

`lcmp`: preleva i primi due `long` in cima allo stack (`ab` e `cd`), li confronta, e mette sullo stack il numero 1 (di tipo `int`) se il secondo valore è maggiore del primo (`cd > ab`), -1 se il secondo è minore del primo (`cd < ab`), 0 se i due valori sono uguali (`cd == ab`).

`ldc_w x`: mette in cima allo stack la costante `x` di tipo `int` o di tipo `String` (stringa tra virgolette). Se `x` è di tipo `int` dev'essere un numero intero compreso tra i valori rappresentati nella tabella dei tipi 3.2 a pagina 6 (nella riga `int`). Se `x` è di tipo `String` dev'essere una stringa valida racchiusa tra virgolette.

`ldc2_w x`: mette in cima allo stack la costante `x` di tipo `long`. La costante `x` dev'essere un numero intero compreso tra i valori rappresentati nella tabella dei tipi 3.2 a pagina 6 (nella riga `long`).

`ldiv`: preleva i primi due `long` in cima allo stack (`ab` e `cd`), esegue la divisione tra il secondo e il primo (`cd/ab`) e mette il risultato (`long`) sullo stack.

`lload n`: mette in cima allo stack il `long` memorizzato nella variabile locale `n`.

`lmul`: preleva i primi due `long` in cima allo stack (`ab` e `cd`), esegue il prodotto tra il secondo e il primo (`cd*ab`) e mette il risultato (`long`) sullo stack.

`lneg`: preleva il `long` in cima allo stack (`ab`), lo nega (`-ab`) e mette il risultato sullo stack.

`lrem`: preleva i primi due `long` in cima allo stack (`ab` e `cd`), calcola il resto della divisione tra il secondo e il primo (`cd%ab`) e mette il risultato (`long`) sullo stack.

`lreturn`: preleva il `long` in cima allo stack (`ab`), termina l'esecuzione della funzione, torna il controllo alla funzione chiamante e mette il `long` in cima allo stack. Da utilizzare nelle funzioni che hanno come tipo di ritorno `J`.

`lshl`: preleva un `int` e un `long` dallo stack (`a` e `bc`), calcola lo shift a sinistra del valore di tipo `long` (`bc « a`) e mette il risultato (`long`) sullo stack.

`lshr` preleva un `int` e un `long` dallo stack (`a` e `bc`), calcola lo shift a destra del valore di tipo `long` (`bc » a`) e mette il risultato (`long`) sullo stack.

`lstore n`: preleva il `long` in cima allo stack (`ab`), e memorizza il valore nella variabile locale `n` (**nota**: un long occupa due posizioni nelle variabili locali).

`lsub`: preleva i primi due `long` in cima allo stack (`ab` e `cd`), esegue la sottrazione tra il secondo e il primo (`cd-ab`) e mette il risultato (`long`) sullo stack.

`nop`: non fa niente.

`pop`: toglie il valore nel primo slot in cima allo stack (`a`).

`pop2`: toglie i valori nei primi due slot in cima allo stack (`ab`).

`putstatic Main/`*`f`* *`d`*`:` mette nella variabile globale di nome *`f`* e tipo *`d`* il valore in cima allo stack.

`return`: interrompe l'esecuzione della funzione e torna il controllo alla funzione chiamante. Da utilizzare nelle funzioni che hanno come tipo di ritorno `V`.

`sipush n`: converte la costante numerica `n` in `short` (valore da -32.768 a 32767) e la mette in cima allo stack.

`swap`: scambia il valore del primo slot (`a`) sullo stack con il secondo (`b`).

4.3 Etichette

Le etichette vengono utilizzate dalle istruzioni di controllo per "saltare", eventualmente dopo la verifica di una certa condizione, ad una determinata istruzione. Le etichette hanno le seguenti caratteristiche:

- Sono nella forma *<nome>*`:`.
- Devono essere all'inizio della riga, con un'istruzione di seguito o nella riga successiva.
- Il nome dev'essere unico all'interno di una funzione.
- Il nome è composto da caratteri alfanumerici.
- Non possono avere il nome di un'istruzione.

Esempio

```
goto label
label: ldc_w 1
```

5 Stampa e lettura

È possibile stampare su standard output e leggere da standard input `int`, `long`, `char` e `String` (costanti stringa).
Di seguito vengono illustrate le istruzioni per stampare e leggere, da notare che ogni istruzioni dev'essere su una riga, terminata da a-capo, nel testo seguente vengono spezzate su più righe per questioni di spazio.

5.1 Stampa

Con il seguente codice si può stampare un elemento di tipo `T` su standard output:

```
getstatic java/lang/System/out Ljava/io/PrintStream;
; serie di istruzioni
invokevirtual java/io/PrintStream/print(T)V
```

Il tipo `T` può essere: `C` per `char`, `I` per `int`, `J` per `long` oppure `Ljava/lang/String;` per le stringhe. La prima istruzione mette sullo stack degli operandi l'elemento `out`, la seconda (dopo una serie qualunque di istruzioni) chiama la funzione di stampa su standard output. Quando viene eseguita la seconda istruzione, in cima allo stack degli operandi dev'essere presente un elemento di tipo `T` (che verrà stampato), seguito immediatamente dall'elemento `out`.

Esempio

Stampa della stringa "Hello, World!!!" su std out:

```
getstatic java/lang/System/out Ljava/io/PrintStream;
ldc_w "Hello, World!!!"
invokevirtual java/io/PrintStream/print(Ljava/lang/String;)V
```

5.2 Lettura

Per leggere da input si utilizzano le seguenti serie di istruzioni (rispettivamente per `char`, `String`, `int` e `long`).

Char

Legge da input un carattere e lo mette in cima allo stack:

```
new java/io/BufferedReader
dup
new java/io/InputStreamReader
dup
getstatic java/lang/System/in Ljava/io/InputStream;
invokespecial
    java/io/InputStreamReader/<init> (Ljava/io/InputStream;)V
invokespecial
    java/io/BufferedReader/<init> (Ljava/io/Reader;)V
invokevirtual java/io/BufferedReader/read ()I
```

String

Legge da input una stringa terminata da a-capo e la mette in cima allo stack:

```
new java/io/BufferedReader
dup
new java/io/InputStreamReader
dup
getstatic java/lang/System/in Ljava/io/InputStream;
invokespecial
    java/io/InputStreamReader/<init> (Ljava/io/InputStream;)V
invokespecial
    java/io/BufferedReader/<init> (Ljava/io/Reader;)V
invokevirtual
    java/io/BufferedReader/readLine ()Ljava/lang/String;
```

Int

Legge da input un `int` e lo mette in cima allo stack, se l'input non è un numero intero viene messo il valore 0 in cima allo stack, se il numero è troppo grande viene troncato al valore massimo del tipo `int`:

```
new java/io/BufferedReader
dup
new java/io/InputStreamReader
dup
getstatic java/lang/System/in Ljava/io/InputStream;
invokespecial
    java/io/InputStreamReader/<init> (Ljava/io/InputStream;)V
invokespecial
    java/io/BufferedReader/<init> (Ljava/io/Reader;)V
invokevirtual
    java/io/BufferedReader/readLine ()Ljava/lang/String;
invokestatic
    java/lang/Integer/parseInt (Ljava/lang/String;)I
```

Long

Legge da input un `long` e lo mette in cima allo stack, se l'input non è un numero intero viene messo il valore 0 in cima allo stack, se il numero è troppo grande viene troncato al valore massimo del tipo `long`:

```
new java/io/BufferedReader
dup
new java/io/InputStreamReader
dup
```

```
getstatic java/lang/System/in Ljava/io/InputStream;
invokespecial
    java/io/InputStreamReader/<init> (Ljava/io/InputStream;)V
invokespecial
    java/io/BufferedReader/<init> (Ljava/io/Reader;)V
invokevirtual
    java/io/BufferedReader/readLine ()Ljava/lang/String;
invokestatic
    java/lang/Long/parseLong (Ljava/lang/String;)J
```

6 Ipotesi iniziali

Per l'implementazione della macchina astratta sono state fatte le seguenti ipotesi iniziali:

1. Il programma è sintatticamente corretto, percui:
    - Tutte le direttive sono scritte in modo corretto e sono nella tabella del paragrafo 2.1.
    - Tutte le istruzioni sono scritte in modo corretto e sono tra le tabelle del paragrafo 4.1.
    - Tutte le funzioni sono scritte in modo corretto e terminano con un'istruzione "return".
    - Le lettere dei tipi degli argomenti delle funzioni sono senza spazi fra loro (eventualmente con spazi tra gli argomenti e le parentesi, e tra le parentesi e il nome della funzione e il tipo di ritorno).
    - Ogni istruzione (con eventualmente un'etichetta prima dell'istruzione) ed ogni direttiva, sono su una riga e terminano con un a-capo.
    - Le etichette hanno nomi univoci all'interno delle funzioni.
    - Le etichette non hanno nomi di istruzioni.
    - Se è presente una etichetta è all'inizio della riga.
    - Gli argomenti delle istruzioni `ldc_w` e `ldc2_w` (per mettere elementi sullo stack degli operandi) sono corretti, ovvero sono costanti numeriche comprese tra i range dei tipi `int` e `long` rispettivamente, oppure una costante stringa corretta nel caso dell'istruzione `ldc_w`.
2. Il programma è semanticamente corretto, percui:
    - Gli accessi alle variabili locali (istruzioni "load") avvengono solo se si è sicuri che sono state inizializzate (con istruzioni "store").

- Gli accessi alle variabili locali avvengono rispettando i tipi memorizzati all'interno delle variabili; per esempio, l'istruzione `iload 2` può essere fatta solo se nella variabile locale `2` c'è memorizzato un `int`.
- La variabile globale utilizzata come argomento nelle istruzioni `putstatic` e `getstatic` (per l'accesso alle variabili globali) esiste ed è stata definita nel testo del programma.
- Le etichette utilizzate come argomenti nelle istruzioni di controllo (es. `goto label`) esistono all'interno della funzione in cui vengono eseguite.
- L'istruzione di return delle funzioni è compatibile con il tipo di ritorno della funzione; ad esempio, se il tipo di ritorno è `S` (short) l'istruzione di return dev'essere `ireturn`.

3. La stampa viene eseguita rispettando esattamente la sintassi e l'ordine delle istruzioni del paragrafo 5.1.

4. La lettura viene eseguita rispettando esattamente la sintassi e l'ordine delle istruzioni del paragrafo 5.2

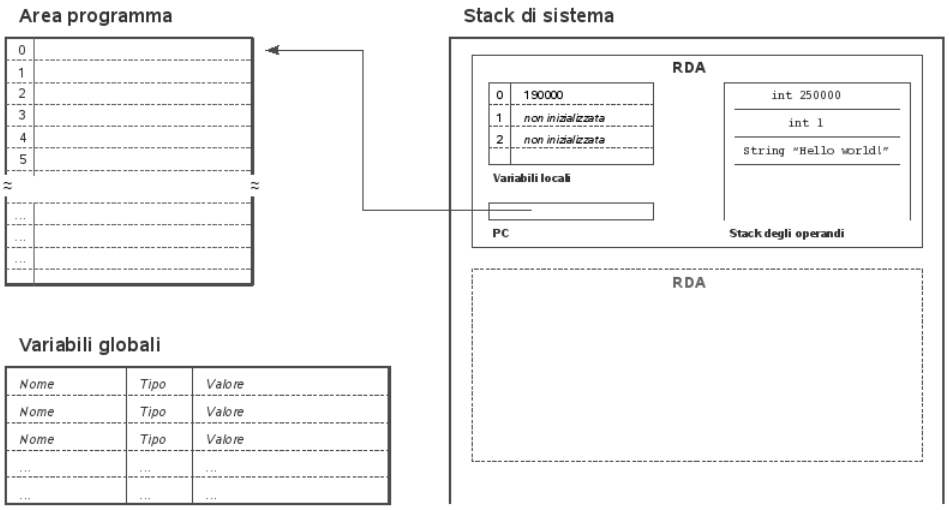
Il file contenente il programma deve assolutamente rispettare i punti sopra citati per il funzionamento della macchina astratta.

7 Struttura e funzionamento generale

Di seguito viene illustrata la struttura della macchina astratta, cioè l'insieme delle strutture dati utilizzate per eseguire il programma, e il funzionamento, cioè un insieme di punti generali che descrivono come la macchina astratta esegue il programma.

7.1 Struttura

La macchina astratta è composta da un'**area programma**, da uno spazio **variabili globali** e da uno **stack di sistema**.



Nell'**area programma** ci sono le istruzioni del programma divise in funzioni ed ogni istruzione ha un indice univoco e consecutivo all'istruzione che la precede. Nello spazio **variabili globali** ci sono tutte le variabili globali con il loro valore, distinte in base al nome e al tipo. Lo **stack di sistema** contiene i RDA (record di attivazione): ogni funzione ha un suo RDA, una chiamata a funzione comporta l'inserimento di un nuovo RDA sullo **stack di sistema** e le istruzioni "return" tolgono un RDA dallo stack. Ogni RDA ha uno **stack degli operandi**, uno spazio per le **variabili locali** e un **PC** (program counter); lo **stack degli operandi** viene utilizzato dalle varie istruzioni (vedere il funzionamento delle istruzioni in 4.2); lo spazio delle **variabili locali** contiene fino a 65536 "registri" nei quali si possono memorizzare o leggere valori, rispettivamente con le istruzioni "store" e "load"; infine il **PC** contiene l'indice dell'istruzione da eseguire dell'**area programma**.

7.2 Funzionamento

Vengono eseguiti una serie di passi iniziali per costruire e inizializzare le strutture dati della macchina astratta, dopodiché l'esecuzione passa ad una funzione "esecutore" che si occupa di eseguire le istruzioni del programma.

Inizio

1. Legge il file contenente il programma e carica le istruzioni (divise per funzioni) nella struttura dati **area programma**.

2. Ricerca le variabili globali nel programma e le mette nello spazio **variabili globali**.

3. Se esiste la funzione `<clinit>` (funzione per inizializzare le variabili globali) crea un RDA vuoto nello **stack di sistema**, imposta il PC

alla prima istruzione della funzione `<clinit>`, passa il controllo alla funzione esecutore e, quando ha terminato, prosegue con il punto seguente. Se tale funzione non esiste prosegue con il punto seguente.

4. Costruisce il primo RDA nello **stack di sistema** con lo stack degli operandi vuoto, le variabili locali non inizializzate e il PC che punta alla prima istruzione della funzione `main`.

5. Passa il controllo all'**esecutore** che esegue le istruzioni puntate dal PC e termina quando lo **stack di sistema** è vuoto (l'istruzione `return` toglie un RDA dallo stack).

Esecutore

1. Controlla se lo **stack di sistema** è vuoto. Se non è vuoto continua, altrimenti termina l'esecuzione.

2. Prende il valore del PC del RDA in cima allo **stack di sistema**.

3. Legge l'istruzione puntata dal PC.

4. Incrementa il PC.

5. Esegue l'istruzione seguendo le specifiche in 4.2.

6. Ritorna al punto 1.

8 Implementazione in C++

La macchina astratta è stata implementata in C++, definendo e utilizzando le seguenti classi:

ActivationRecord: rappresenta un record di attivazione, con un PC, uno stack degli operandi e un'area per le variabili locali. Fornisce dei metodi per leggere e scrivere nelle variabili locali, per cambiare il valore del PC e per eseguire le operazioni fondamentali (leggere, togliere e mettere un elemento) sullo stack degli operandi.

SystemStack: rappresenta uno stack di sistema, praticamente una pila di ActivationRecord. Fornisce dei metodi per inserire e togliere un record di attivazione e per accedere a quello in cima alla pila.

GlobalVariablesArea: rappresenta l'area dove vengono memorizzate le variabili globali del programma ed il loro valore. Permette di aggiungere variabili globali e di scriverne e leggere il valore.

ProgramArea: in un oggetto di questo tipo viene mantenuto l'insieme delle istruzioni del programma, suddivise per funzione di appartenenza, con un indice univoco associato e consecutivo in modo da poter essere "puntate" dal PC di un RdA.

Il programma si compone di due funzioni principali di nome `main` ed `esecutore`, mantenute rispettivamente nei file *macchina-astratta.cc* e *esecutore.cc*.
Nel file *macchina-astratta.cc* vengono definiti tre oggetti di tipo `ProgramArea`, `GlobalVariablesArea` e `SystemStack` che rappresentano rispettivamente l'**area programma**, lo spazio per le **variabili globali** e lo **stack di sistema**.
La funzione `main` prende come argomento il nome del file da eseguire (contenente il programma), dopodichè esegue i seguenti passi:

1. Inizializza l'oggetto di tipo `ProgramArea` e carica le istruzioni del programma al suo interno.

2. Inizializza l'oggetto di tipo `GlobalVariablesArea` e ci mette dentro le variabili globali del programma.

3. Se all'interno del programma da eseguire esiste la funzione "<clinit> ()V" (per l'inizializzazione delle variabili globali) crea un record di attivazione vuoto nell'oggetto di tipo `SystemStack` impostando il PC alla prima istruzione di "<clinit> ()V" e chiama la funzione `esecutore`.

4. Crea un record di attivazione nell'oggetto di tipo `SystemStack` e imposta il PC alla prima istruzione della funzione "main" del programma da eseguire.

5. Passa il controllo alla funzione `esecutore` che si occupa di eseguire le istruzioni contenute nell'oggetto `ProgramArea`.

La funzione `esecutore`, finchè nell'oggetto `SystemStack` c'è almeno un record di attivazione, esegue i seguenti passi:

1. Prende il valore del PC del record di attivazione in cima all'oggetto di tipo `SystemStack`.

2. Legge l'istruzione dall'oggetto di tipo `ProgramArea` puntata dal PC.

3. Incrementa il PC.

4. Esegue l'istruzione chiamando una funzione appropriata.

8.1 Compilazione, installazione e documantazione

Per compilare e creare l'eseguibile della macchina astratta, o per creare la documentazione attraverso doxygen, è possibile dare uno dei seguenti comandi all'interno della directory "macchina-astratta" dove sono presenti i file sorgente:

- `make`: crea l'eseguibile `macchina-astratta` dentro la directory "bin".
- `make doc`: crea la documentazione del codice in formato html e pdf dentro la directory "doc".

L'eseguibile `macchina-astratta` si aspetta come argomento un file, all'interno del quale ci dovrà essere il codice del programma da eseguire.