Gestione della memoria e istruzioni di IJVM

Aniello Minutolo

Modello della memoria di IJVM

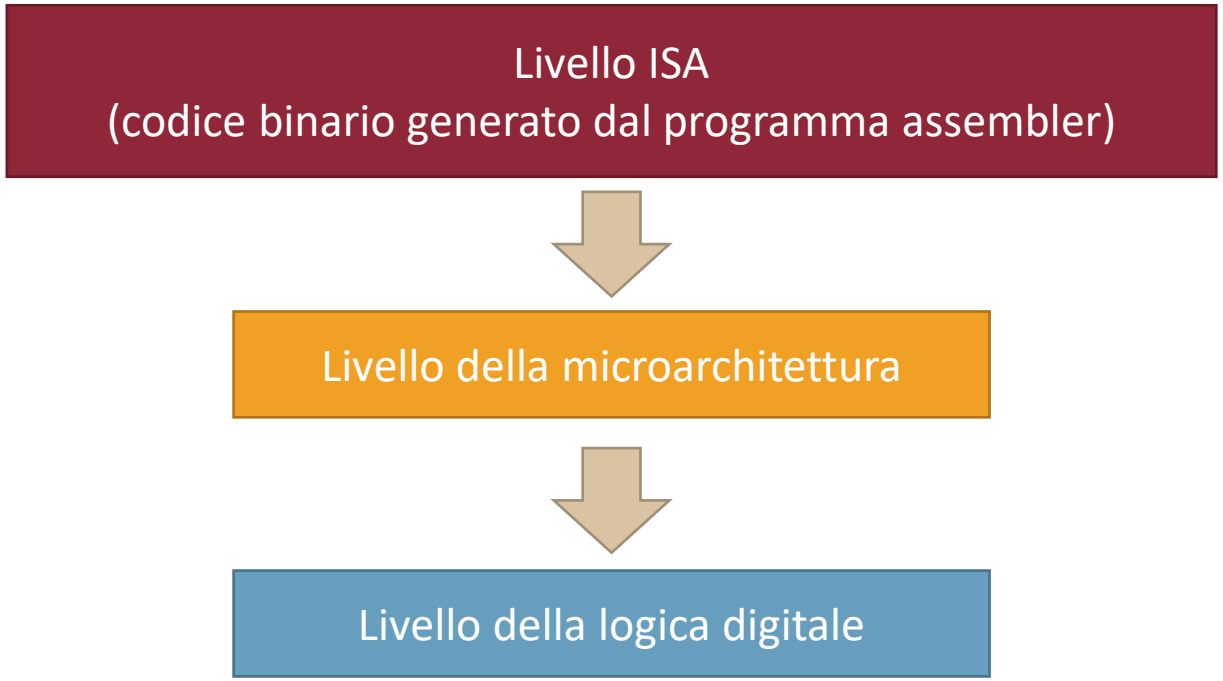
Sommario

1. Modello della memoria di IJVM
2. Gestione dello stack in IJVM
3. Le istruzioni di IJVM

Progettazione di una microarchitettura

La **microarchitettura**

- descrive il funzionamento interno di una CPU, spiegando come le istruzioni **ISA** (Instruction Set Architecture) vengono decodificate ed elaborate dall'hardware della CPU (il livello della logica digitale)



Progettazione di una microarchitettura

Come studiare la progettazione di una microarchitettura

- Tramite analisi di un caso semplice ma sufficientemente generale da poter essere ampliato e reso più complesso secondo necessità.

Caso di studio

- Microarchitettura di una CPU in grado di eseguire le istruzioni di IJVM (Integer JVM), cioè di avere IJVM come ISA.

IJVM

- Contiene solo istruzioni su numeri interi (e non floating point).

Architettura di IJVM

- Introduciamo IJVM, il livello ISA (architettura dell'insieme d'istruzioni) della macchina che sarà interpretato dal microprogramma eseguito nella microarchitettura Mic-1.
- Per praticità a volte ci si riferisce all'ISA con il termine di macroarchitettura, in contrapposizione con la microarchitettura.

Modello della memoria di IJVM

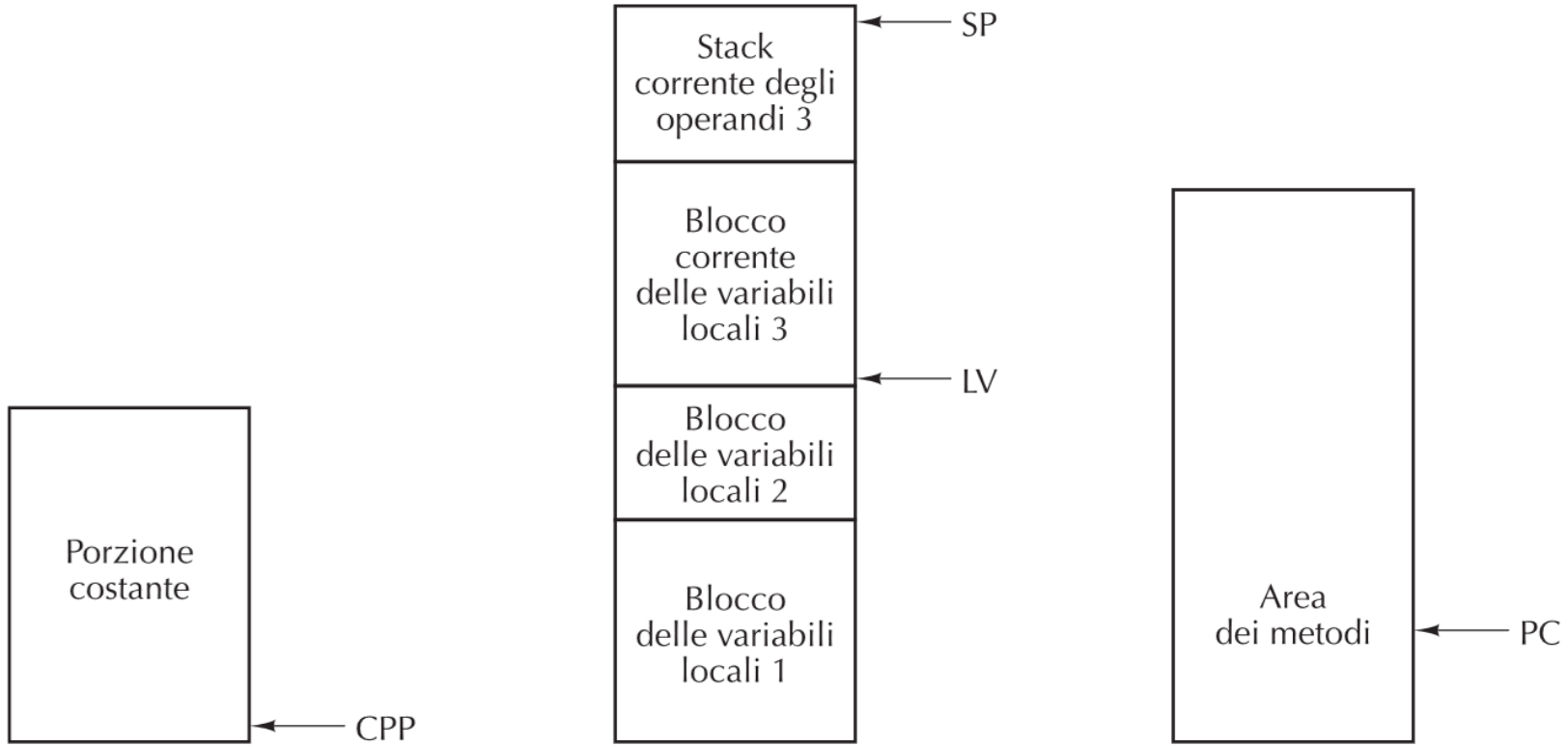
L’architettura di IJVM consiste in una memoria concepibile in due modi distinti: come un array di 4.294.967.296 byte (4 GB) oppure come un array di 1.073.741.824 parole da 4 byte.

La Java Virtual Machine non rende direttamente visibili a livello ISA gli indirizzi di memoria assoluti, ma utilizza degli indirizzi impliciti che forniscono la base per l’uso di puntatori

- l’unico modo che le istruzioni IJVM hanno per accedere alla memoria è quello di indicizzarla utilizzando questi puntatori.

Modello della memoria di IJVM

- In ogni momento sono definite le seguenti aree di memoria.
- La memoria RAM è fisicamente un'unica unità; la suddivisione in più regioni è solo una semplificazione.
- Il blocco dello stack non può superare una certa dimensione, stabilita in anticipo dal compilatore Java.

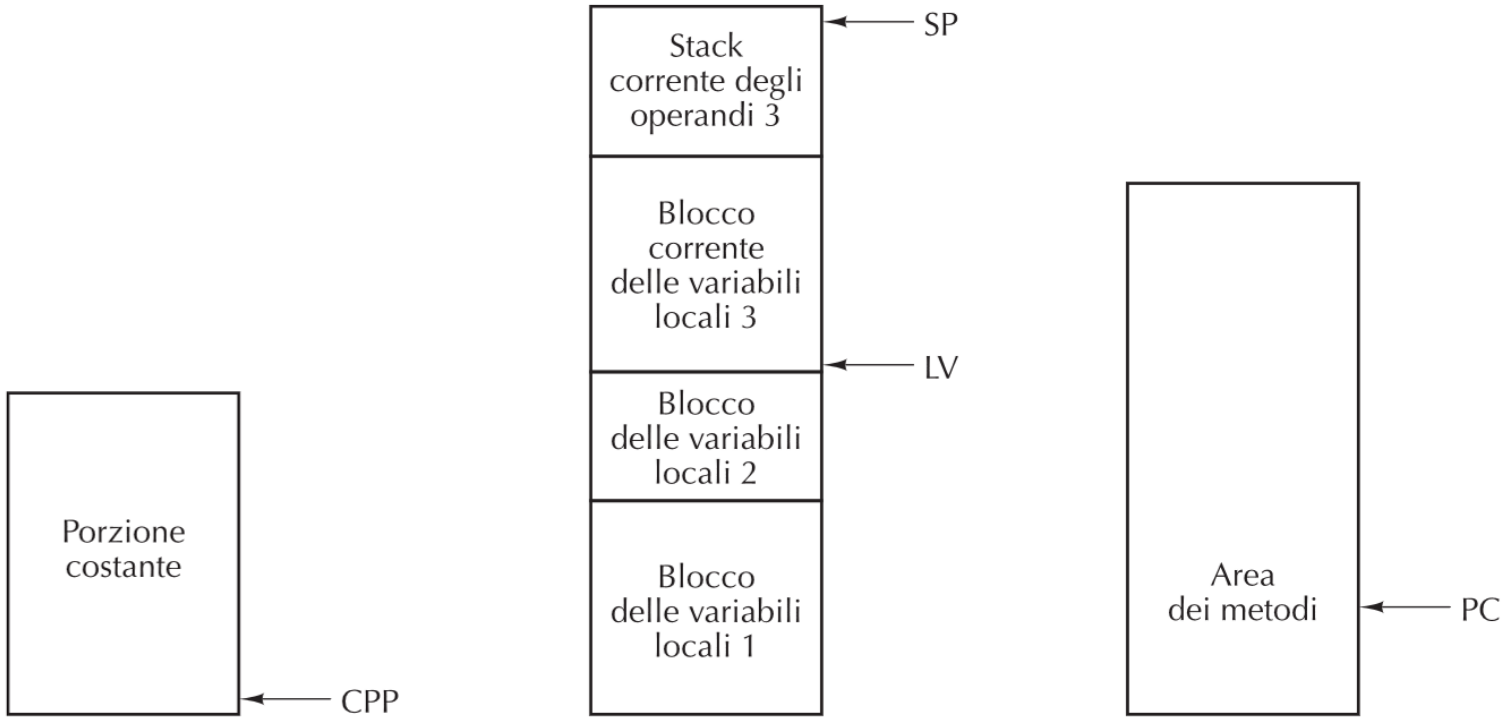


blocco dello stack

Modello della memoria di IJVM

Porzione costante di memoria

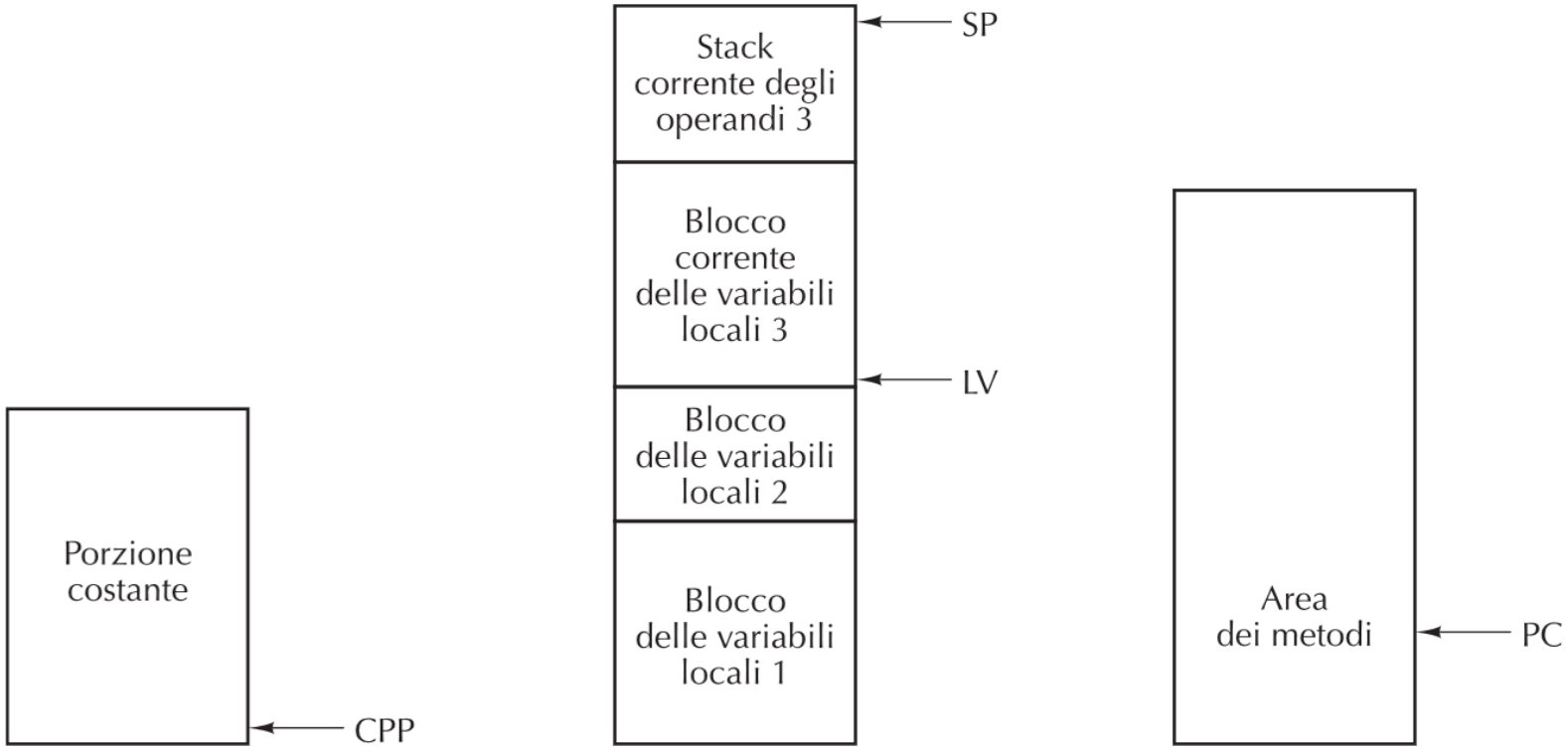
- i programmi IJVM non possono scrivere in quest'area che contiene costanti, stringhe e puntatori ad altre aree di memoria;
- è caricata quando un programma IJVM è portato in memoria e in seguito non viene modificata;
- il registro CPP contiene l'indirizzo della prima parola della porzione.



blocco dello stack

Blocco delle variabili locali

- per ogni invocazione di un metodo viene allocata un'area di memoria dedicata alla memorizzazione delle variabili locali durante l'intero ciclo di vita dell'invocazione;
- il registro LV contiene l'indirizzo della prima parola all'interno del blocco corrente delle variabili locali.

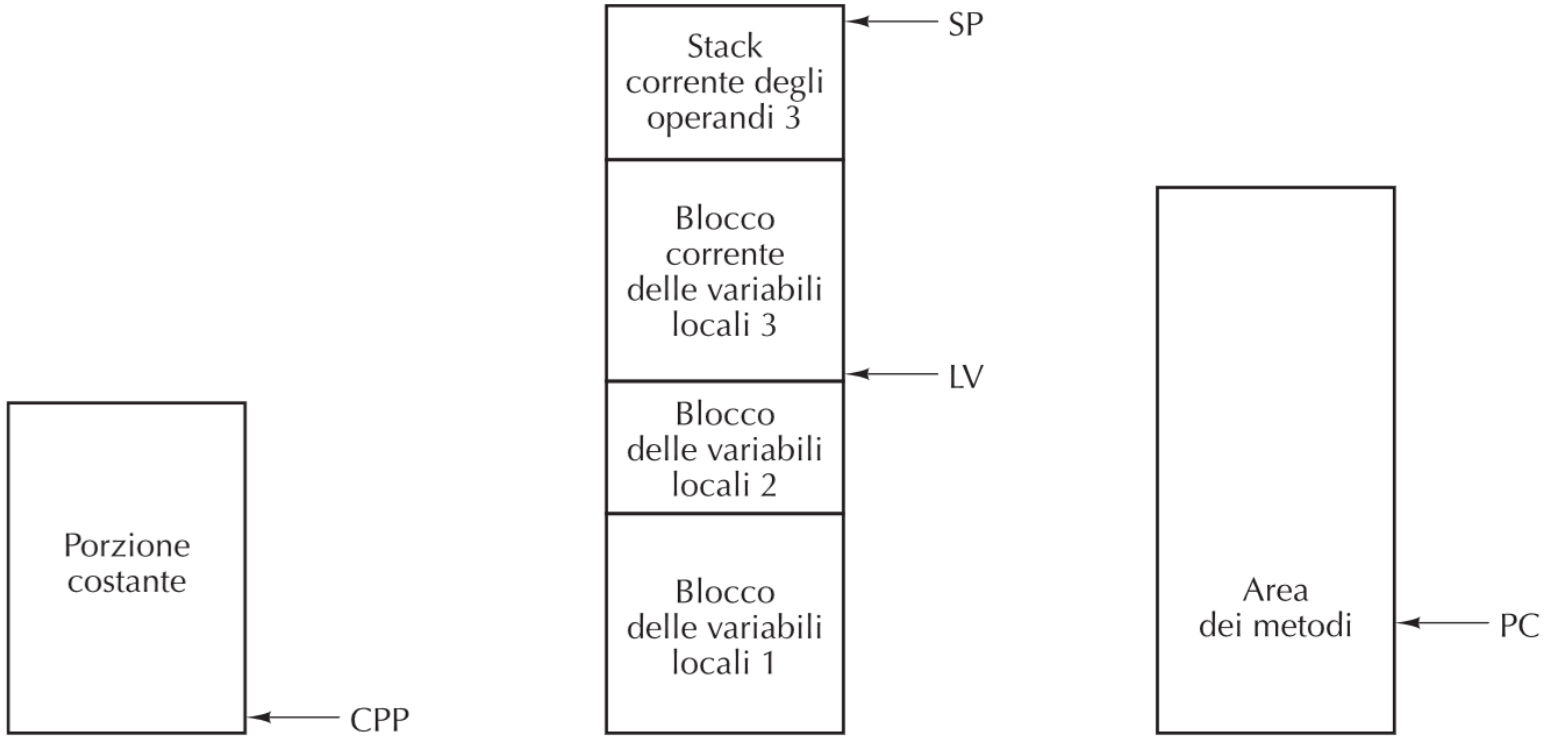


blocco dello stack

Modello della memoria di IJVM

Stack degli operandi

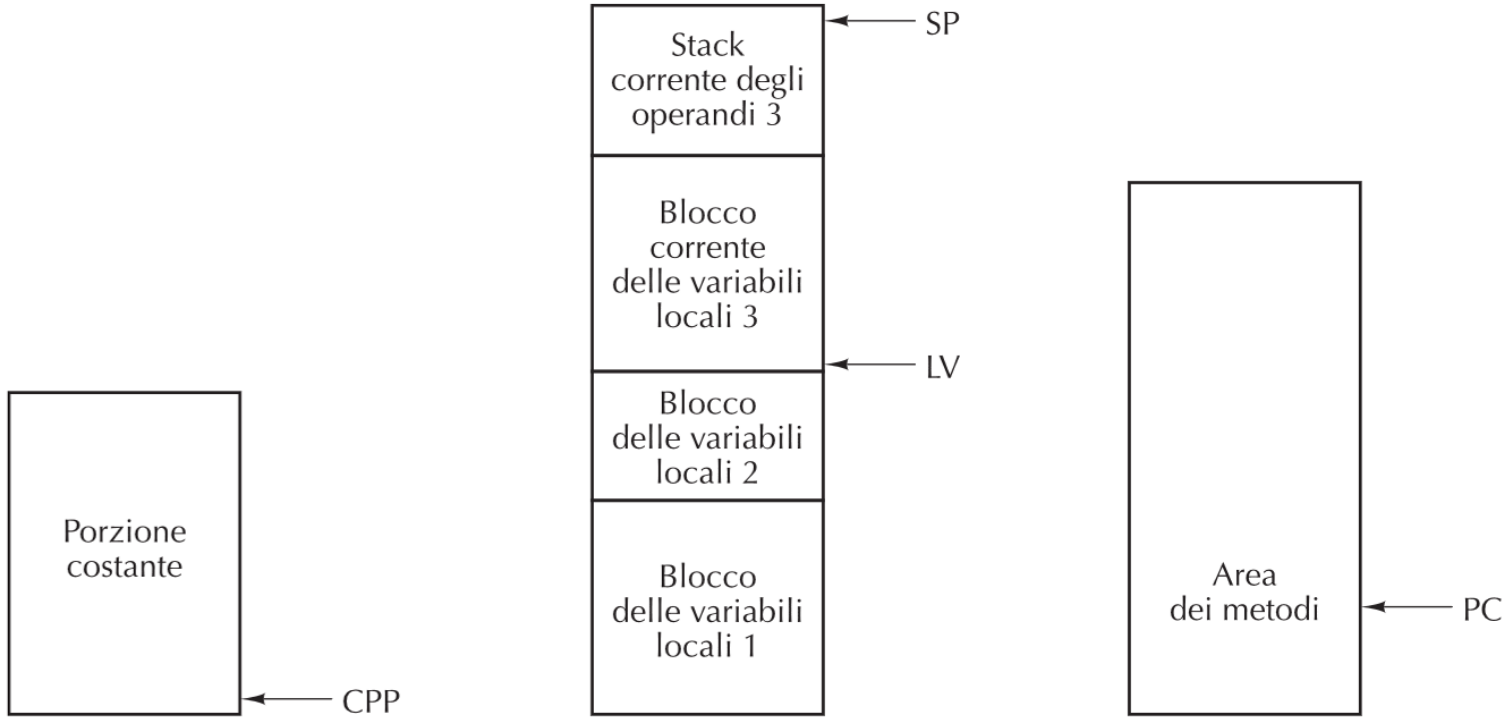
- è allocato direttamente sopra il blocco delle variabili locali;
- il registro SP contiene l'indirizzo della parola in cima allo stack;
- SP può cambiare durante l'esecuzione di un metodo a seconda che gli operandi siano inseriti nello stack o rimossi da quest'ultimo.



Modello della memoria di IJVM

Area dei metodi

- regione di memoria in cui risiede il programma;
- diversamente dalle altre regioni di memoria, l'area dei metodi è trattata come un array di byte;
- il registro PC contiene l'indirizzo della successiva istruzione da prelevare.



blocco dello stack

Gestione dello stack in IJVM

Gestione dello stack in IJVM

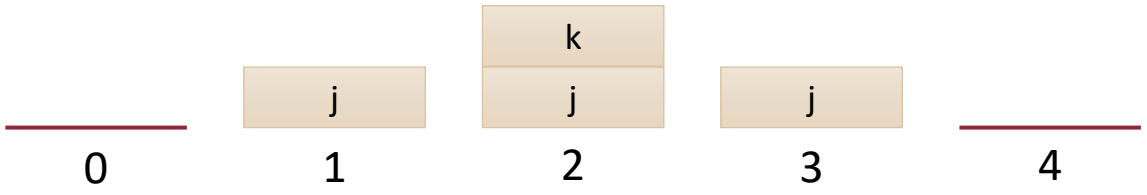
IJVM gestisce la memoria in modo peculiare, basandosi principalmente sull'utilizzo dello stack.

Lo stack è una regione di memoria in cui i dati vengono organizzati in una struttura a pila, con accesso consentito solo dalla cima:

- un'operazione di scrittura (Push) incrementa l'altezza della pila;
- un'operazione di lettura (Pop) ne riduce le dimensioni.

Nell'esempio illustrato, le operazioni vengono eseguite nell'ordine:

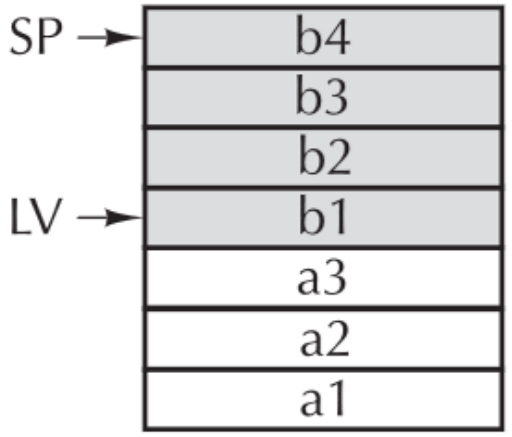
- push j; push k; pop k; pop j.



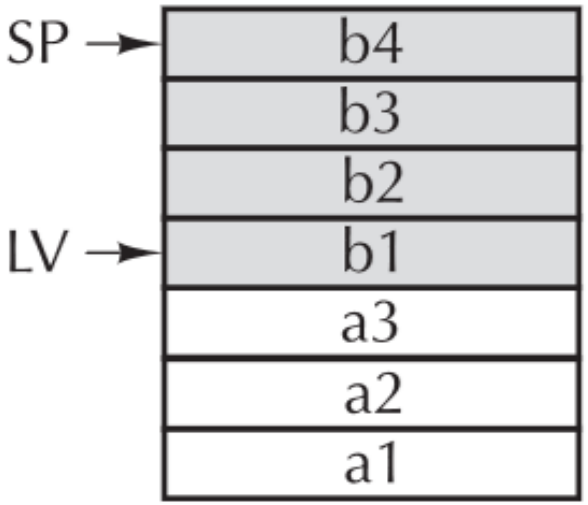
Gestione delle variabili locali nello stack

IJVM utilizza due registri per gestire all'interno dello stack le variabili locali delle procedure invocate

- **LV (Local Variable)**: punta alla parola che si trova nella locazione **più bassa** all'interno dello stack delle variabili locali della procedura corrente;
- **SP (Stack Pointer)**: punta alla parola che si trova nella locazione **più alta** all'interno dello stack delle variabili locali della procedura corrente.



Gestione delle variabili locali nello stack



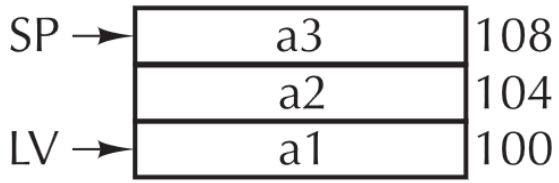
Blocco delle variabili locali della procedura corrente

- è la struttura dati compresa tra LV e SP (considerando anche le parole puntate dai due registri).

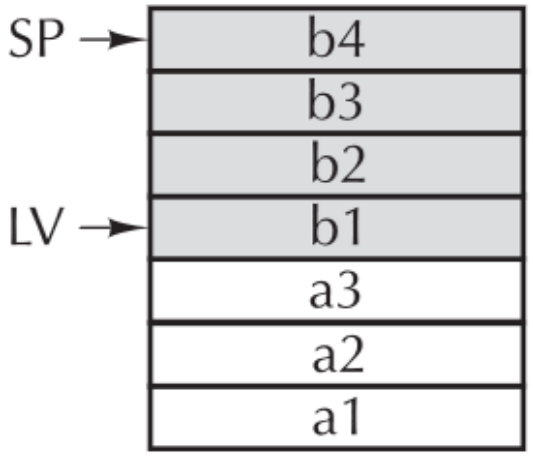
Gestione delle variabili locali nello stack

Esempio

- è stata invocata una procedura *A(a1, a2, a3)*, avente tre variabili locali;
- per memorizzare *a1, a2 e a3* viene riservato uno spazio di memoria nello stack a partire dalla locazione puntata da LV;
- se LV vale 100 e le parole sono di 4 byte, il valore di SP sarà 108
- è possibile far riferimento alle variabili locali di A fornendo il loro spiazzamento rispetto al valore di LV.



Gestione delle variabili locali nello stack



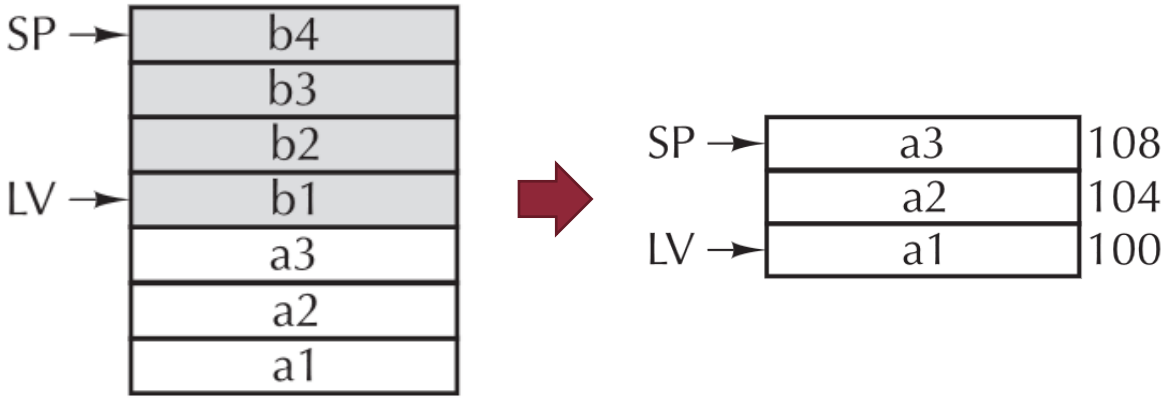
Esempio

- che cosa succede se *A* richiama la procedura *B(b1, b2, b3, b4)*?
- le quattro variabili locali (b1, b2, b3, b4) di *B* vengono memorizzate nello stack sopra il blocco delle variabili locali di *A.*
- LV viene modificato dalla chiamata alla procedura in modo da puntare alle variabili locali di *B* invece che a quelle di *A.*
- Il nuovo valore di SP sarà LV + 4 * (numero delle variabili locali – 1).
- Per accedere alle variabili locali di *B* si indica lo spiazzamento rispetto a LV.

Gestione delle variabili locali nello stack

Esempio

- che cosa succede quando la procedura *B* termina?
- la procedura *A* diventa nuovamente attiva;
- i valori di LV e SP vengono modificati in modo da riportare lo stack allo stato caratterizzante il blocco delle variabili locali di *A;*
- per accedere alle variabili locali di *A* si indica lo spiazzamento rispetto a LV.



Gestione degli operandi nello stack

Oltre a memorizzare le variabili locali nello stack, IJVM utilizza lo stack anche per memorizzare gli operandi durante il calcolo di un'espressione aritmetica:

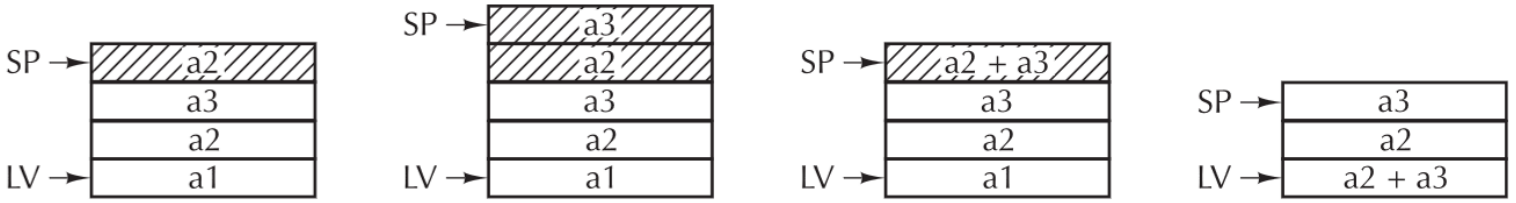
- non tutte le macchine lo fanno.

Quando uno stack è utilizzato in questo modo ci si riferisce a esso con il termine di **stack degli operandi.**

Gestione degli operandi nello stack

Esempio

- è stata invocata *A(a1, a2, a3)*, che deve eseguire il calcolo *a1 = a2 + a3.*
- per fare il calcolo si possono porre gli operandi *a1* e *a2* in cima allo stack;
- SP viene incrementato in base al numero di operandi e al numero di byte che formano una parola, in modo da puntare in cima allo stack;
- viene eseguita un'istruzione che preleva due parole dallo stack, le somma e inserisce il risultato nuovamente nello stack, ed aggiorna SP;
- la parola contenente il risultato può essere rimossa e memorizzata nella variabile locale a1, ed SP viene aggiornato tornando allo stato iniziale.



Gestione dello stack in IJVM

Il blocco delle variabili locali e lo stack degli operandi possono essere mischiati tra loro.

Esempio: calcolo di un'espressione come *x^2 + f(x)*

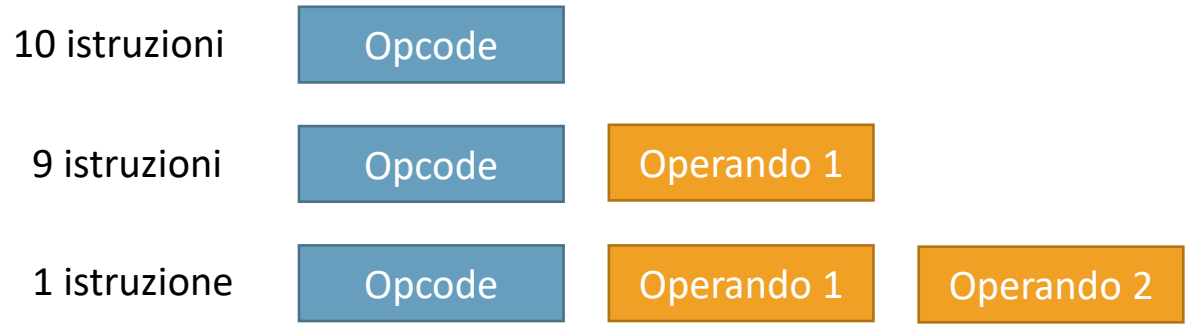
- una parte dell'espressione (per esempio, *x^2*) può trovarsi nello stack degli operandi nel momento in cui viene invocata *f;*
- il risultato della funzione *f* viene lasciato nello stack, al di sopra di *x^2*, in modo che l'istruzione successiva li possa sommare.

Le istruzioni di IJVM

Le istruzioni di IJVM

L'insieme d'istruzioni IJVM è composto da 20 istruzioni, dove ogni istruzione è composta da un codice operativo e in alcuni casi da un operando, che può essere uno spiazzamento o una costante:

- ogni istruzione è composta da un codice operativo (**opcode**) che identifica l'istruzione (ADD, BRANCH o altre operazioni);
- diverse istruzioni presentano operando, che può essere uno spiazzamento o una costante.



Le istruzioni di IJVM

L'IJVM è caratterizzato da un set di istruzioni minimalista, comprendente **20 istruzioni:**

- **3 istruzioni** per inserire dati in cima allo stack da diverse sorgenti (*BIPUSH, ILOAD, LDCW*);
- **1 istruzione** per leggere il valore in cima allo stack e salvarlo nell'area delle variabili locali (*ISTORE*);
- **3 istruzioni** per la gestione dello stack: scambio, duplicazione e rimozione dell'elemento in cima (*SWAP, DUP, POP*);
- **4 operazioni** tra cui 2 aritmetiche (*IADD, ISUB*) e 2 logiche (*IAND, IOR*);
- **1 istruzione** per incrementare il valore di una variabile locale (*IINC*);
- **4 istruzioni di salto**, di cui 1 incondizionata (*GOTO*) e 3 condizionate (*IFEQ, IFLT, IFICMPEQ*);
- **1 istruzione** per estendere a 16 bit l'indice di un operando (*WIDE*);
- **2 istruzioni** per la gestione dei metodi (*INVOKEVIRTUAL, IRETURN*);
- **1 istruzione** senza effetto (*NOP*).

La tabella mostra le istruzioni di IJVM

- gli operandi *byte*, *const* e *varnum* sono lunghi 1 byte;
- gli operandi *disp*, *index* e *offset* sono lunghi 2 byte.

Codifica Esadecimale	Nome mnemonico in linguaggio assemblativo	Descrizione del significato
0x10	BIPUSH *byte*	Scrive un byte in cima allo stack
0x59	DUP	Legge la parola in cima allo stack e la inserisce in cima allo stack
0xA7	GOTO *offset*	Diramazione incondizionata
0x60	IADD	Sostituisce le due parole in cima allo stack con la loro somma
0x7E	IAND	Sostituisce le due parole in cima allo stack con il loro AND
0x99	IFEQ *offset*	Estrae una parola dalla cima dello stack ed effettua una diramazione se ha valore zero
0x9B	IFLT *offset*	Estrae una parola dalla cima dello stack ed effettua una diramazione se ha valore negativo
0x9F	IF_ICMPEQ *offset*	Estrae le due parole in cima allo stack ed effettua una diramazione se sono uguali
0x84	IINC *varnum const*	Aggiunge una costante a una variabile locale
0x15	ILOAD *varnum*	Scrive una variabile locale in cima allo stack

La tabella mostra le istruzioni di IJVM

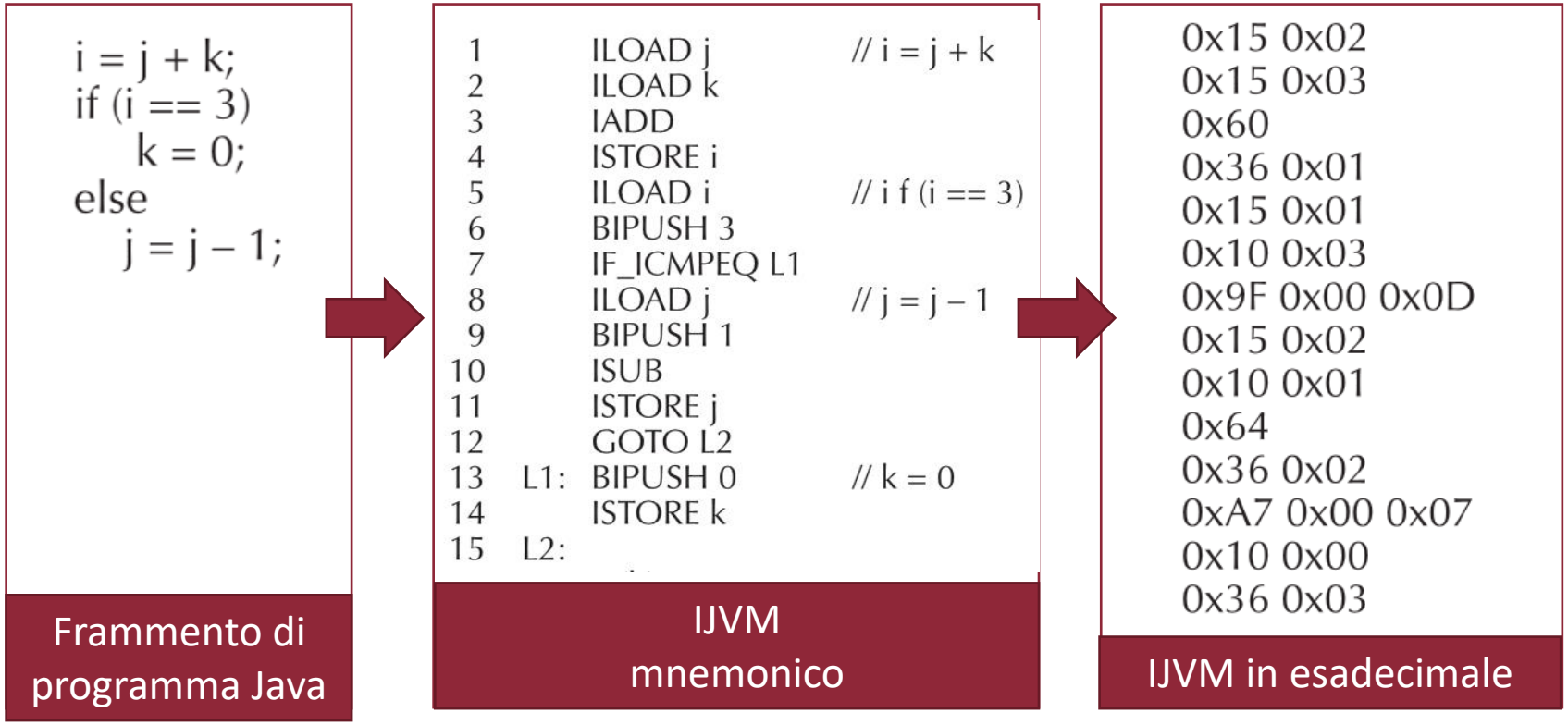
- gli operandi *byte*, *const* e *varnum* sono lunghi 1 byte
- gli operandi *disp*, *index* e *offset* sono lunghi 2 byte

Codifica Esadecimale	Nome mnemonico in linguaggio assemblativo	Descrizione del significato
0xB6	INVOKEVIRTUAL *disp*	Invoca un metodo
0x80	IOR	Sostituisce le due parole in cima allo stack con il loro OR
0xAC	IRETURN	Termina un metodo restituendo un valore intero
0x36	ISTORE *varnum*	Preleva una parola dalla cima dello stack e la memorizza in una variabile locale
0x64	ISUB	Sostituisce le due parole in cima allo stack con la loro differenza
0x13	LDC_W *index*	Scrive in cima allo stack una costante proveniente dalla porzione costante di memoria
0x00	NOP	Non esegue nulla
0x57	POP	Rimuove la cima allo stack
0x5F	SWAP	Scambia le due parole in cima allo stack
0xC4	WIDE	Istruzione prefisso: l'istruzione successiva ha un indice a 16 bit

Compilazione da Java a IJVM

Relazione tra Java e IJVM

- un compilatore Java, che riceve in ingresso un codice Java, dovrebbe generare il codice IJVM mnemonico equivalente, e in seguito tradurre il programma assemblativo nel codice IJVM binario in versione.



Bibliografia

Libro di testo

- Andrew S. Tanenbaum e Todd Austin. Architettura dei calcolatori. Un approccio strutturale. 6/ED. Anno 2013. Pearson Italia spa. (Disponibile nella sezione “Biblioteca”)

Fonte argomenti e immagini

- Capitolo 4, Paragrafo 4.2, pp. 264-274