Il Livello Microarchitettura

A cura di:
Luca Breveglieri[*] Giacomo Buonanno[#]
Roberto Negrini[*] Giuseppe Pozzi[*] Donatella Sciuto[*]

[*] DEI, PoliMI, Milano
[#] LIUC, Castellanza (VA)

breveglieri,negrini,pozzi,sciuto@elet.polimi.it
buonanno@liuc.it

L'insieme delle istruzioni del processore IJVM

(lezione 3)
(Tanenbaum cap. 4)

Contenuto

- Classi di istruzioni
- Funzionamento delle istruzioni:
  - Istruzioni aritmetico-logiche
  - Istruzioni di caricamento-memorizzazione
  - Istruzioni di salto
  - Istruzioni di manipolazione della pila
  - Istruzioni di controllo
- Esempio di programma IJVM e simulazione
- Codifica delle istruzioni IJVM

Generalità sul linguaggio IJVM

- Il linguaggio macchina IJVM (Integer JVM) è un sottoinsieme del linguaggio JVM (Java Virtual Machine) completo
- IJVM contiene le istruzioni macchina di JVM specializzate per l'elaborazione dei numeri interi (naturali e relativi)
- IJVM (come anche JVM) ha un modello di esecuzione basato sul processore JVM (realizzato tramite l'architettura Mic-1) e sulla pila di memoria

Classi di istruzioni IJVM

- Aritmetico-logica: eseguono le principali operazioni aritmetiche e logiche bit-per-bit (bitwise) su numeri interi
- Caricamento-memorizzazione: servono per leggere e scrivere in memoria
- Salto (condizionato e non): servono per controllare il flusso di esecuzione
- Manipolazione della pila: servono per scrivere e leggere in pila di memoria
- Controllo: gestione del processore

Numeri esadecimali (hex)

- La rappresentazione esadecimale (hex) è di tipo posizionale, in base 16

Tipo	Cifre elementari														
hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- $C4 = 12 \times 16 + 4 = 196$
- $1A7E = 1 \times 16^3 + 10 \times 16^2 + 7 \times 16 + 14 = 68222$
- In Java i numeri esadecimali si scrivono con il prefisso 0x; p. es. 0xC4, 0x1A7E

Ultima slide prima della piazzola di sosta

ersione binario-esadecimale

- esadecimale⇒binario: espandere ogni cifra hex in un gruppo di 4 bit
  - 1A7E hex = 1 / A / 7 / E = 0001 / 1010 / 0111 / 1110 = 0001101001111110 bin
- binario ⇒ esadecimale: compattare gruppi di 4 bit in una sola cifra hex
  - 11000011101010 bin = 11 / 0000 / 1110 / 1010 = 3 / 0 / E / A = 30EA hex
- Importante: un byte = 2 cifre hex, una parola da 32 bit = 8 cifre hex

Istruzioni aritmetico-logiche

Hex	Mnemonico	Funzionamento: operazione e risultato
0x60	IADD	Addiziona i 2 elementi in cima alla pila
0x64	ISUB	Sottrae i 2 elementi in cima alla pila
0x7E	IAND	AND bit-per-bit dei 2 elementi in cima alla pila
0x80	IOR	OR bit-per-bit dei 2 elementi in cima alla pila
0x84	INC numvar valore	Addiziona "valore" alla variabile locale "numvar"

- Le istruzioni IADD, ISUB, IAND e IOR operano sulla pila per il calcolo delle espressioni, con parole da 32 bit
- L'istruzione INC opera sulle variabili locali (32 bit), ma con "valore" da un byte (8 bit)

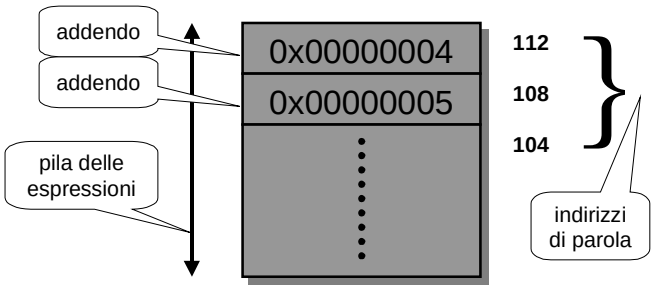
Animazione

Simulazione di IADD

Si addizionano due parole (32 bit);
gli addendi vengono tolti dalla pila

Animazione

Simulazione di IADD

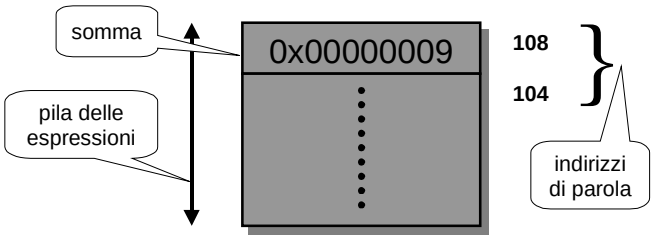


Eseguire l'istruzione: IADD

Si addizionano due parole (32 bit);
gli addendi vengono tolti dalla pila

Animazione

Simulazione di IADD

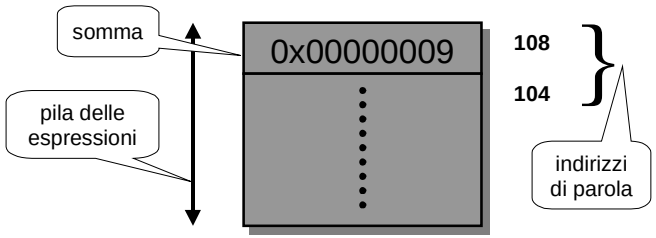


Istruzione IADD eseguita

Si addizionano due parole (32 bit);
gli addendi vengono tolti dalla pila

Animazione
Fine

Simulazione di IADD



Istruzione IADD eseguita

Si addizionano due parole (32 bit);
gli addendi vengono tolti dalla pila

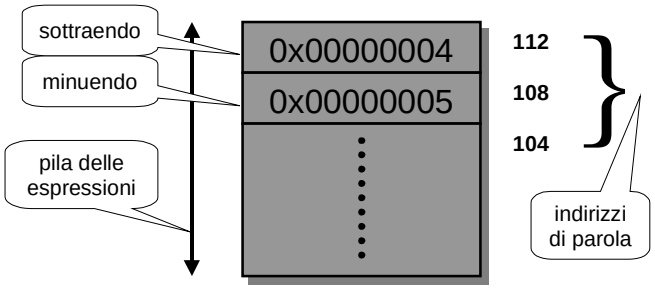
Animazione

Simulazione di ISUB

Si sottraggono due parole (32 bit); minuendo e sottraendo vengono tolti dalla pila

Animazione

Simulazione di ISUB

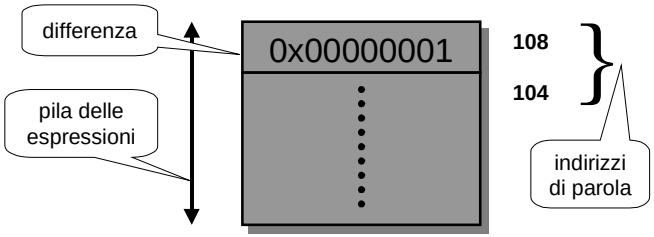


Eseguire l'istruzione: ISUB

Si sottraggono due parole (32 bit); minuendo e sottraendo vengono tolti dalla pila

Animazione

Simulazione di ISUB



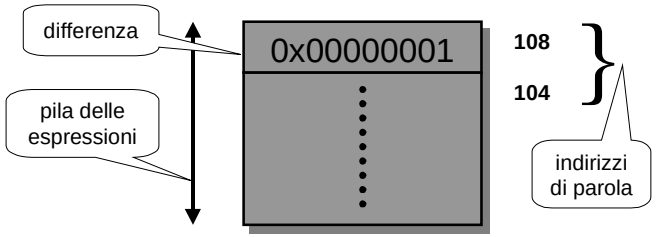
Istruzione ISUB eseguita

Si sottraggono due parole (32 bit); minuendo e sottraendo vengono tolti dalla pila

Animazione

Fine

Simulazione di ISUB



Istruzione ISUB eseguita

Si sottraggono due parole (32 bit); minuendo e sottraendo vengono tolti dalla pila

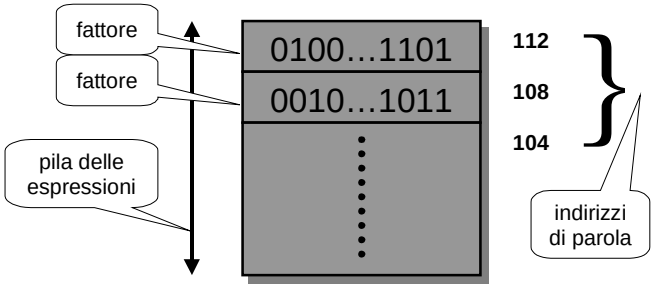
Animazione

Simulazione di IAND

Si calcola il prodotto logico bit-per-bit di due parole (32 bit); i fattori vengono tolti dalla pila

Animazione

Simulazione di IAND

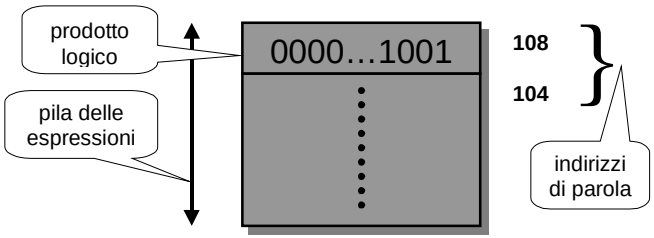


Eseguire l'istruzione: IAND

Si calcola il prodotto logico bit-per-bit di due parole (32 bit); i fattori vengono tolti dalla pila

Animazione

Simulazione di IAND



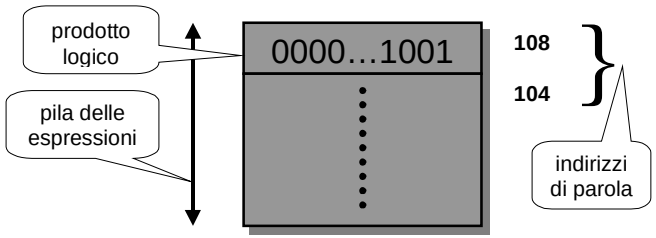
Istruzione IAND eseguita

Si calcola il prodotto logico bit-per-bit di due parole (32 bit); i fattori vengono tolti dalla pila

Animazione

Fine

Simulazione di IAND



Istruzione IAND eseguita

Si calcola il prodotto logico bit-per-bit di due parole (32 bit); i fattori vengono tolti dalla pila

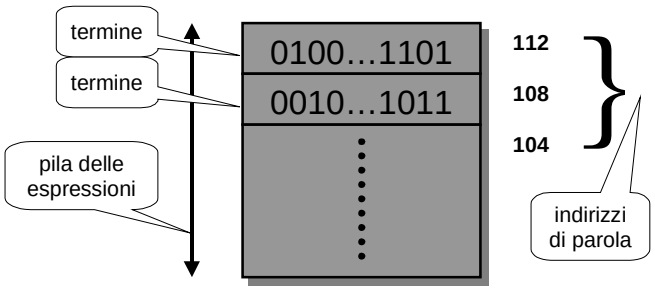
Animazione

Simulazione di IOR

Si calcola la somma logica bit-per-bit di due parole (32 bit); i termini vengono tolti dalla pila

Animazione

Simulazione di IOR

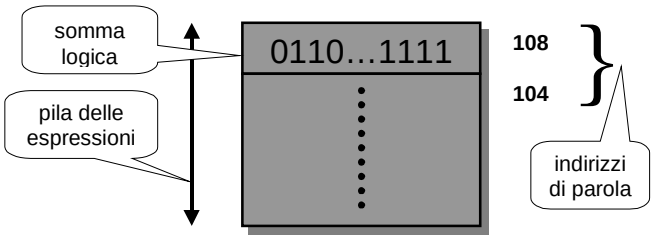


Eseguire l'istruzione: IOR

Si calcola la somma logica bit-per-bit di due parole (32 bit); i termini vengono tolti dalla pila

Animazione

Simulazione di IOR



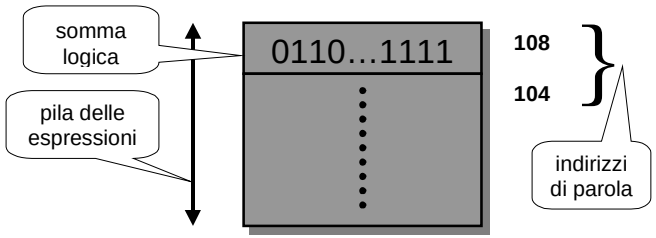
Istruzione IOR eseguita

Si calcola la somma logica bit-per-bit di due parole (32 bit); i termini vengono tolti dalla pila

Animazione

Fine

Simulazione di IOR



Istruzione IOR eseguita

Si calcola la somma logica bit-per-bit di due parole (32 bit); i termini vengono tolti dalla pila

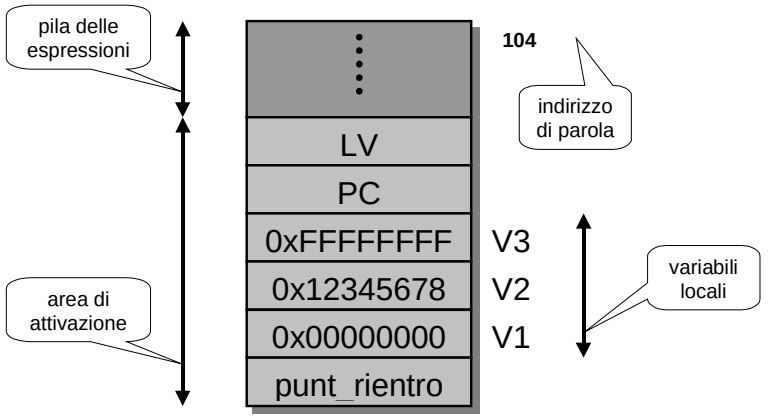
Animazione

Simulazione di INC

Costante (8 bit) sommata a una var. loc. (32 bit)

Animazione

Simulazione di INC

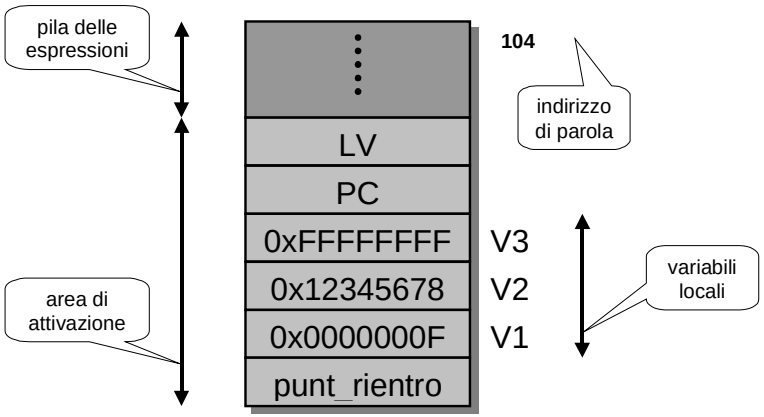


Eseguire l'istruzione: INC 1 15

Costante (8 bit) sommata a una var. loc. (32 bit)

Animazione

Simulazione di INC



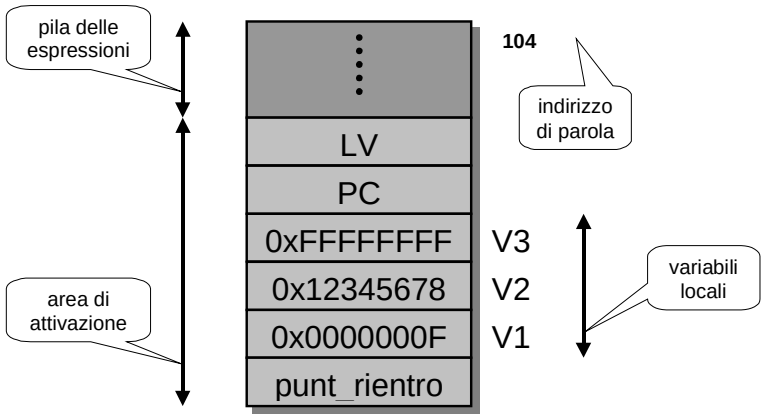
Istruzione INC 1 15 eseguita

Costante (8 bit) sommata a una var. loc. (32 bit)

Animazione

Fine

Simulazione di INC



Istruzione INC 1 15 eseguita

Costante (8 bit) sommata a una var. loc. (32 bit)

Istruzioni di caricamento-memorizzazione

Hex	Mnemonico	Funzionamento: operazione e risultato
0x15	ILOAD numvar	Push della variabile locale "numvar" sulla pila
0x36	ISTORE numvar	Pop dell'elemento sulla pila nella var. loc. "numvar"
0x1A	ILOAD_0	Push della variabile locale numero 0 sulla pila*
0x1B	ILOAD_1	Push della veriabile locale numero 1 sulla pila*
0x1C	ILOAD_2	Push delle variabile locale numero 2 sulla pila*

- Le istruzioni operano su variabili locali e pila per il calcolo delle espressioni
- Le istruzioni operano su parole da 32 bit

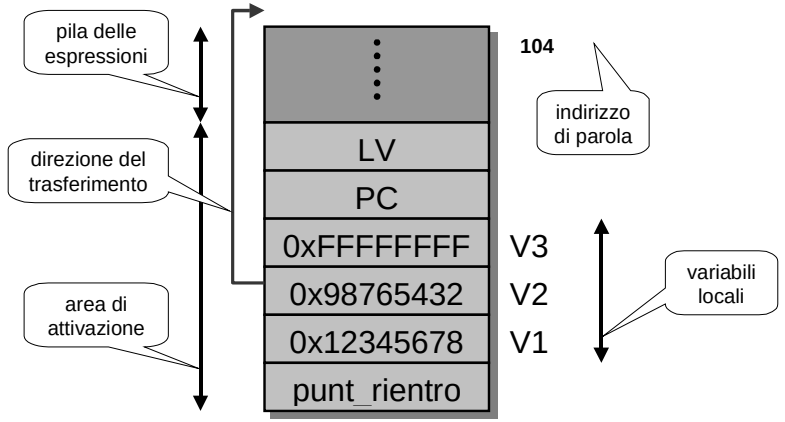
*Sono forme abbreviate dell'istruzione ILOAD

Animazione

Simulazione di ILOAD

Animazione

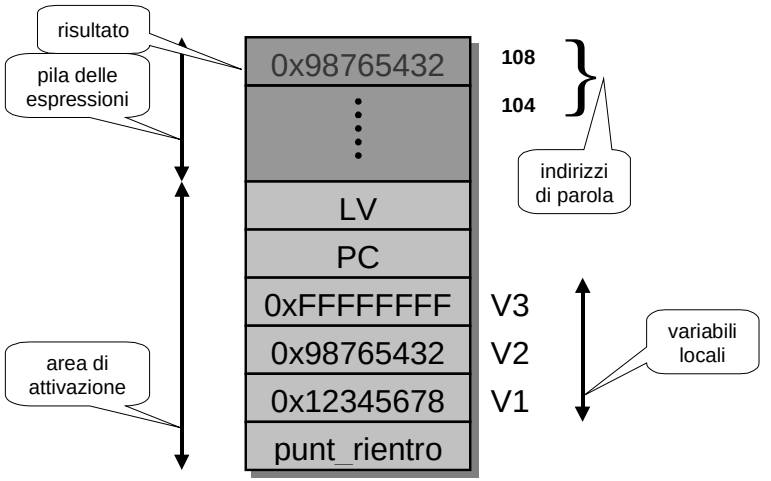
Simulazione di ILOAD



Eseguire l'istruzione: ILOAD 2

Animazione

Simulazione di ILOAD

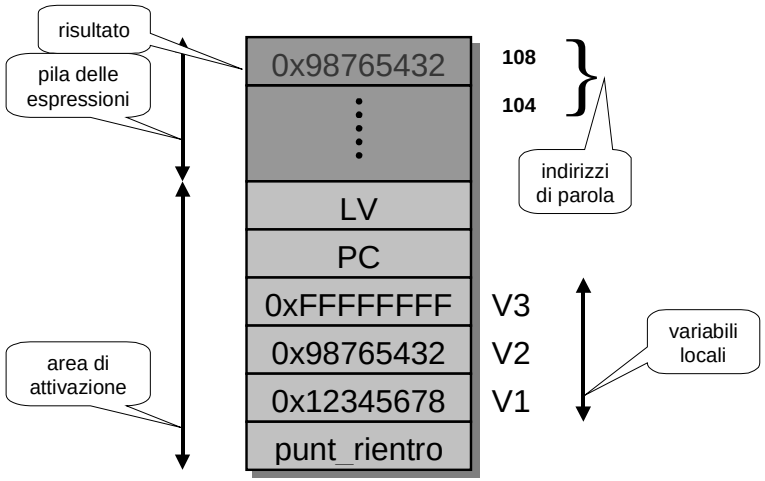


Istruzione ILOAD 2 eseguita

Simulazione di ILOAD

Animazione

Fine



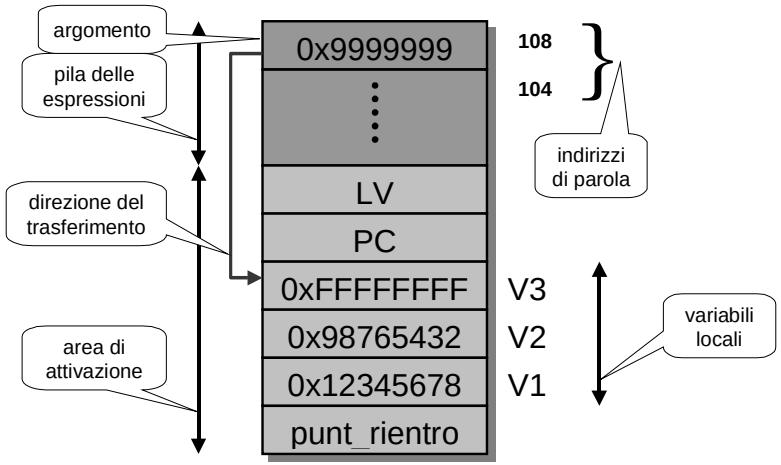
Istruzione ILOAD 2 eseguita

Simulazione di ISTORE

Animazione

Simulazione di ISTORE

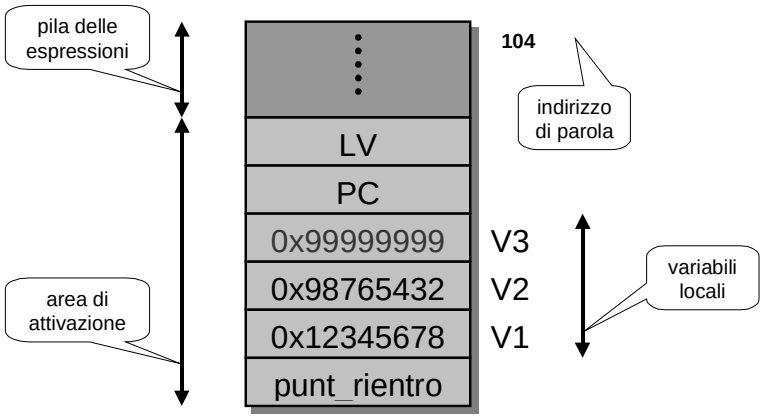
Animazione



Eseguire l'istruzione: ISTORE 3

Simulazione di ISTORE

Animazione

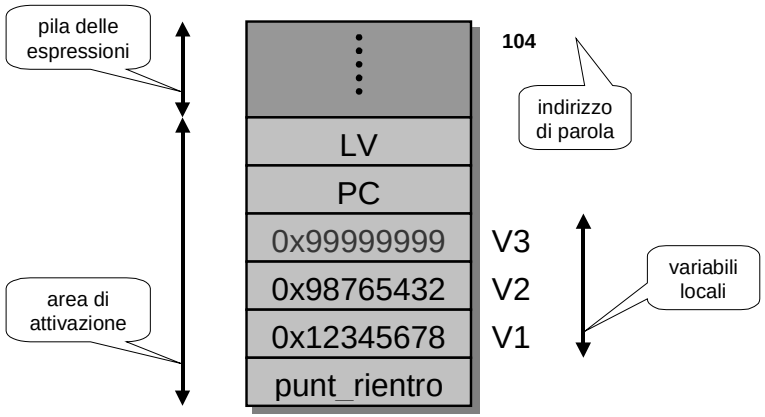


Istruzione ISTORE 3 eseguita

Simulazione di ISTORE

Animazione

Fine



Istruzione ISTORE 3 eseguita

Istruzioni di salto

Hex	Mnemonico	Funzionamento: operazione e risultato
0x99	IFEQ spiazzamento	Salta se l'elemento in cima alla pila è = 0
0x9B	IFLT spiazzamento	Salta se l'elemento in cima alla pila è < 0
0x9F	IFCMPEQ spiazzamento	Salta se i 2 elementi in cima alla pila sono =
0xA7	GOTO spiazzamento	Salta in modo incondizionato
0xAC	IRETURN	Rientra da sottoprogramma
0xB6	INVOKEVIRTUAL spiazzamento	Salta a sottoprogramma

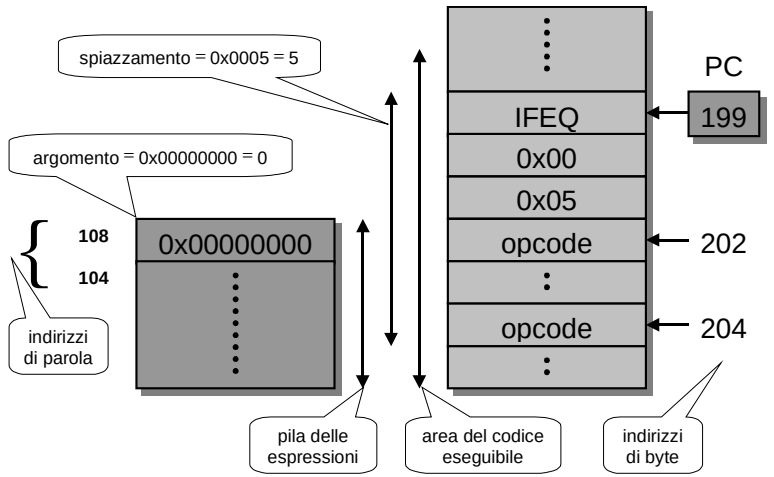
- Le istruzioni IF∗ e GOTO operano sulla pila delle espressioni, con parole da 32 bit, e sul contatore di programma PC
- Per i sottoprogrammi trattazione a parte

Simulazione di IFEQ

Animazione

Simulazione di IFEQ

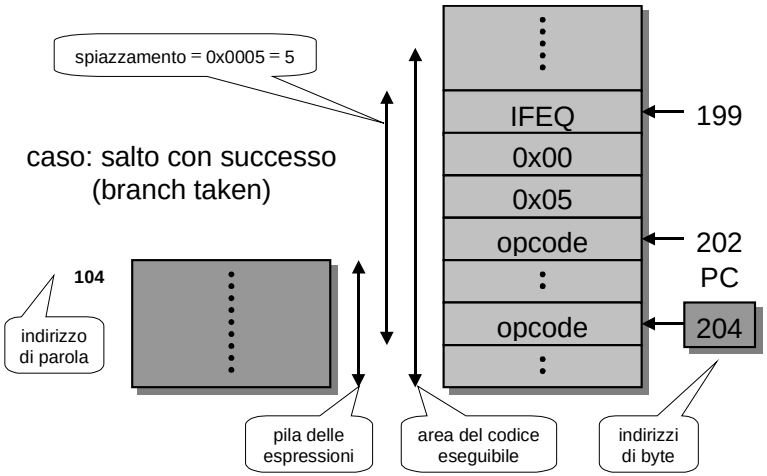
Animazione



Eseguire l'istruzione: IFEQ 5

Animazione

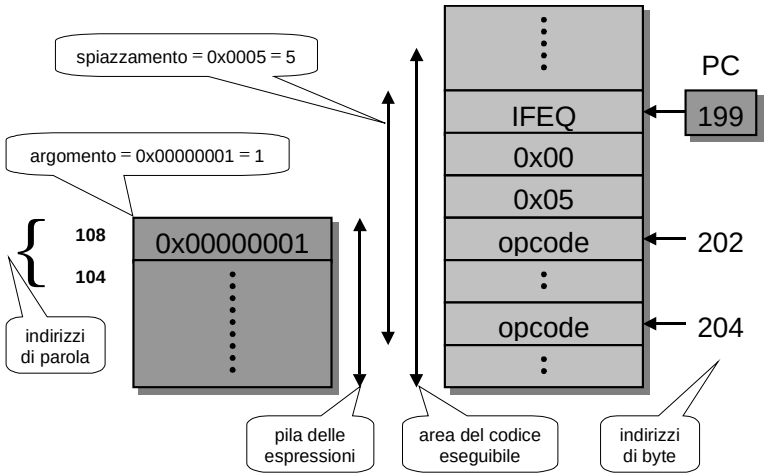
Simulazione di IFEQ



Istruzione IFEQ 5 eseguita (con successo)

Animazione

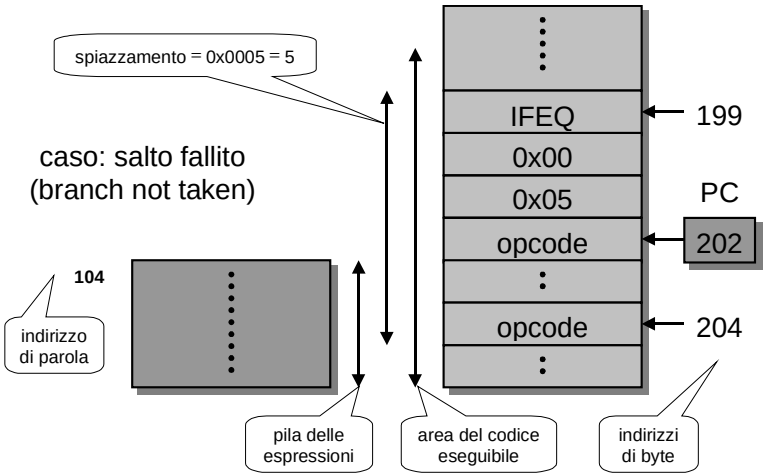
Simulazione di IFEQ



Eseguire l'istruzione: IFEQ 5

Animazione

Simulazione di IFEQ

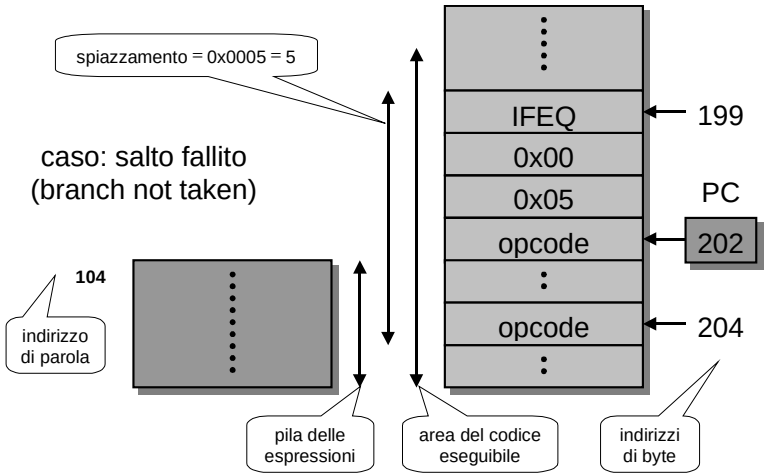


Istruzione IFEQ 5 eseguita (con fallimento)

Animazione

Fine

Simulazione di IFEQ



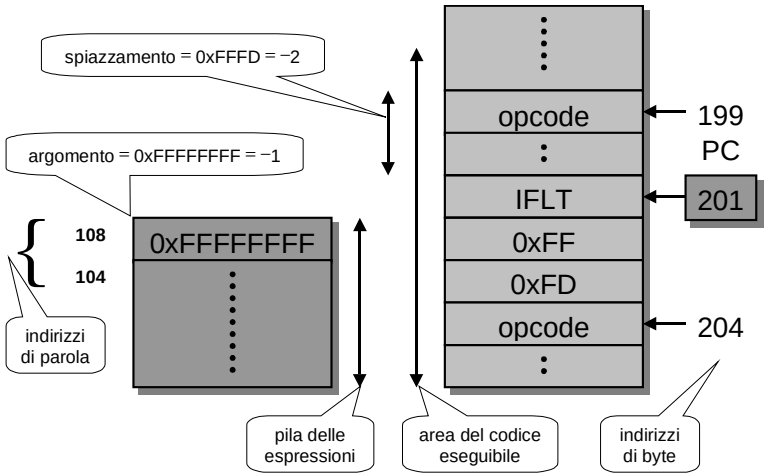
Istruzione IFEQ 5 eseguita (con fallimento)

Animazione

Simulazione di IFLT

Animazione

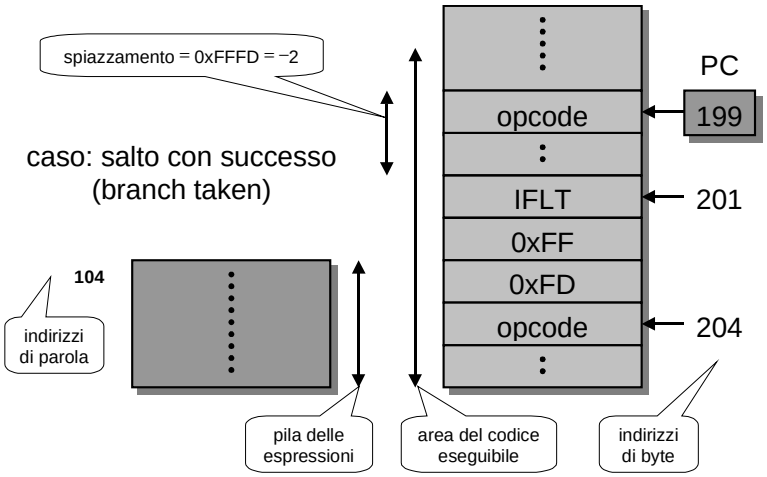
Simulazione di IFLT



Eseguire l'istruzione: IFLT −2

Animazione

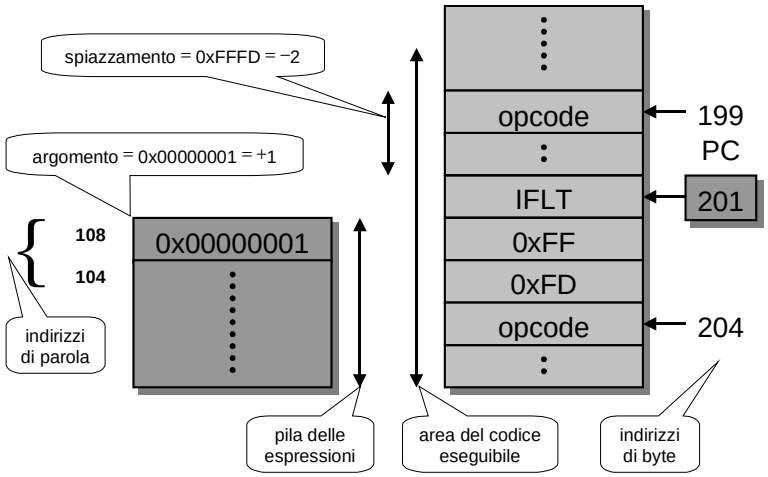
Simulazione di IFLT



Istruzione IFLT −2 eseguita (con successo)

Animazione

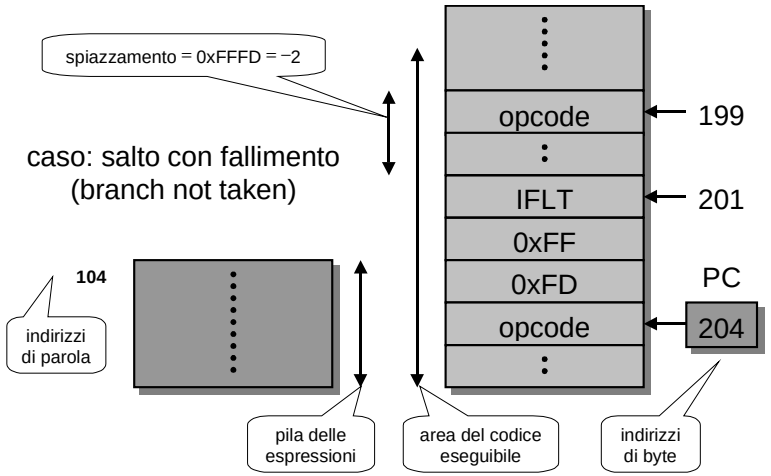
Simulazione di IFLT



Eseguire l'istruzione: IFLT −2

Animazione

Simulazione di IFLT

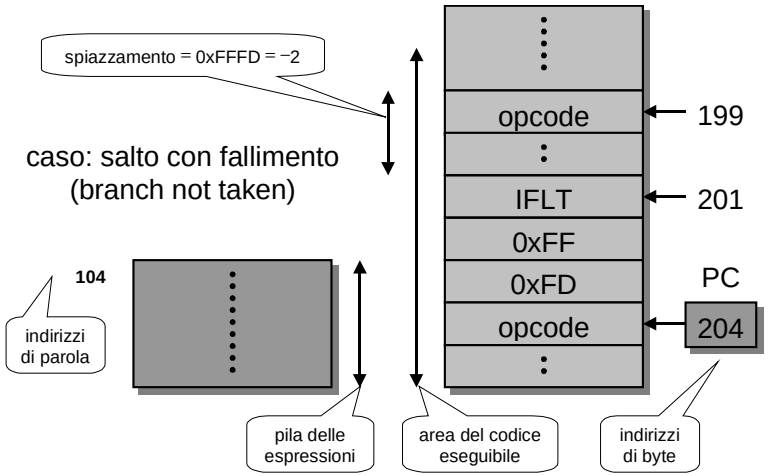


Istruzione IFLT −2 eseguita (con fallimento)

Animazione

Fine

Simulazione di IFLT


spiazzamento = 0xFFFD = −2
caso: salto con fallimento
(branch not taken)
opcode
IFLT
0xFF
0xFD
opcode
199
201
PC
204
104
indirizzi
di parola
pila delle
espressioni
area del codice
eseguibile
indirizzi
di byte

Istruzione IFLT −2 eseguita (con fallimento)

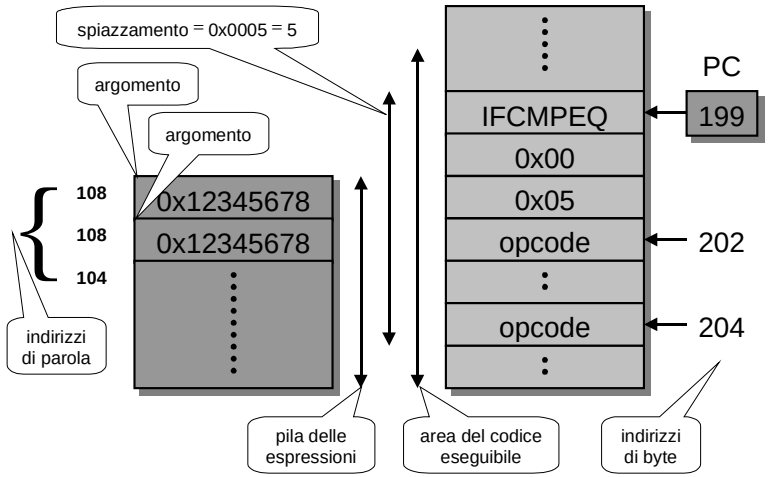
27/03/01 Informatica II - Il livello microarchitettura - lez. 3 - L. Breveglieri pp. 19

Animazione

Simulazione di IFCMPEQ

Animazione

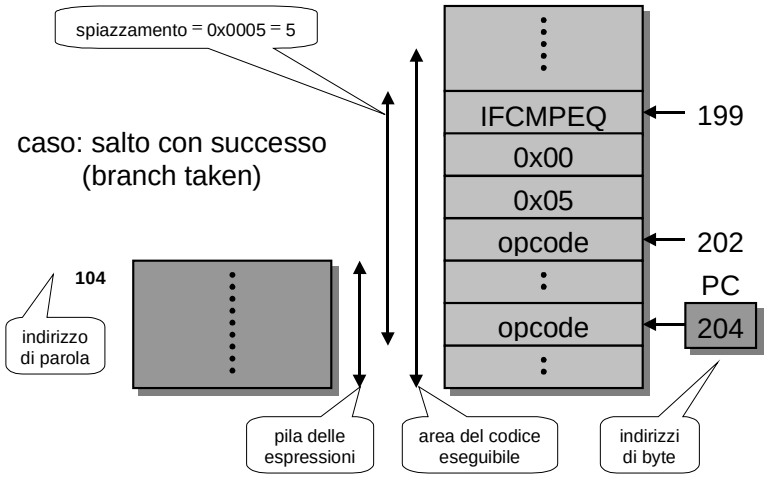
Simulazione di IFCMPEQ



Eseguire l'istruzione: IFCMPEQ 5

Animazione

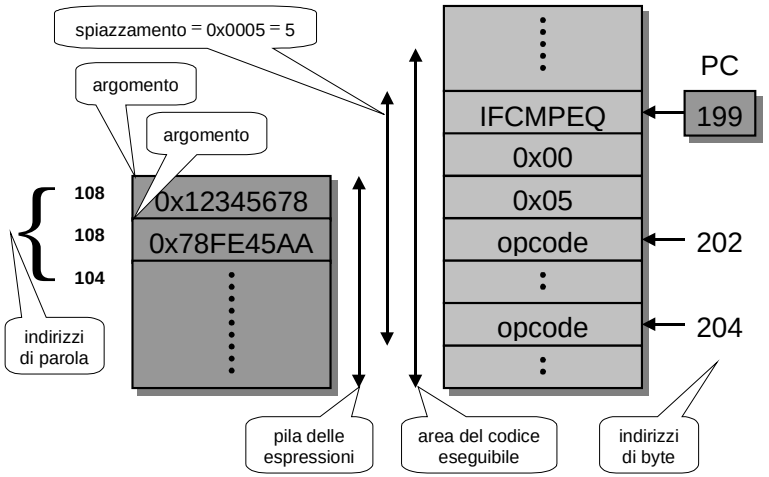
Simulazione di IFCMPEQ



Istruzione IFCMPEQ 5 eseguita (con successo)

Animazione

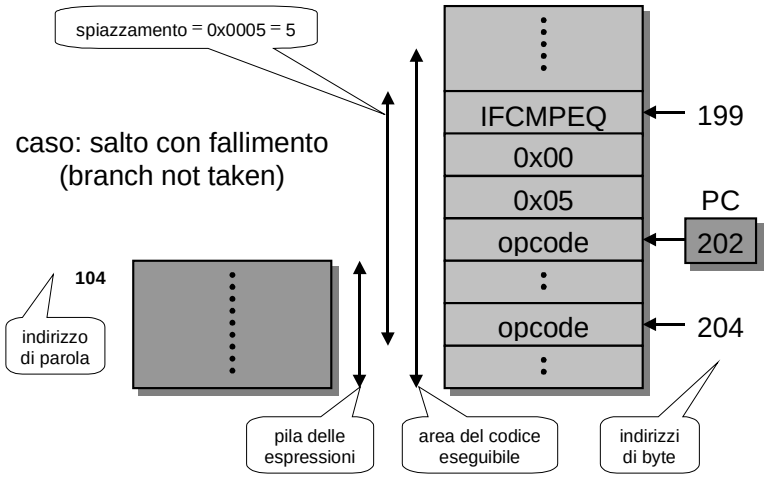
Simulazione di IFCMPEQ



Eseguire l'istruzione: IFCMPEQ 5

Animazione

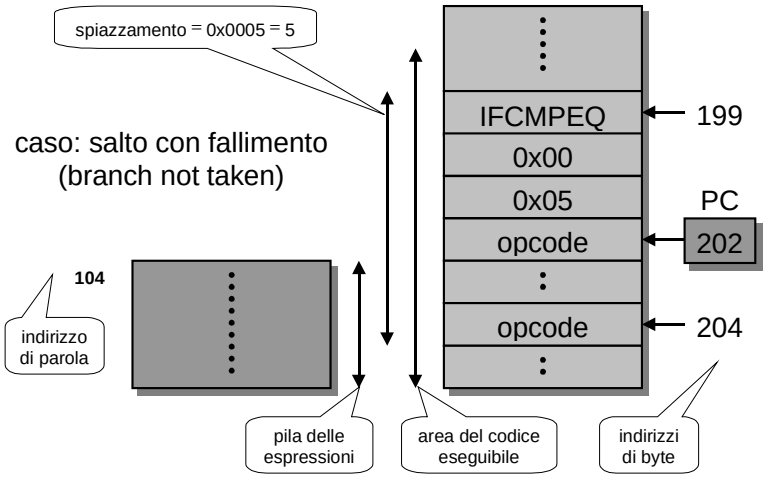
Simulazione di IFCMPEQ



Istruzione IFCMPEQ 5 eseguita (con fallimento)

Animazione
Fine

Simulazione di IFCMPEQ



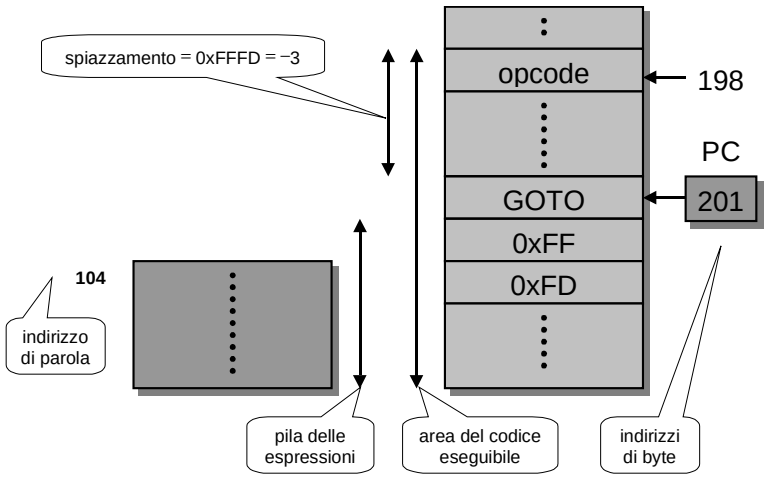
Istruzione IFCMPEQ 5 eseguita (con fallimento)

Animazione

Simulazione di GOTO

Animazione

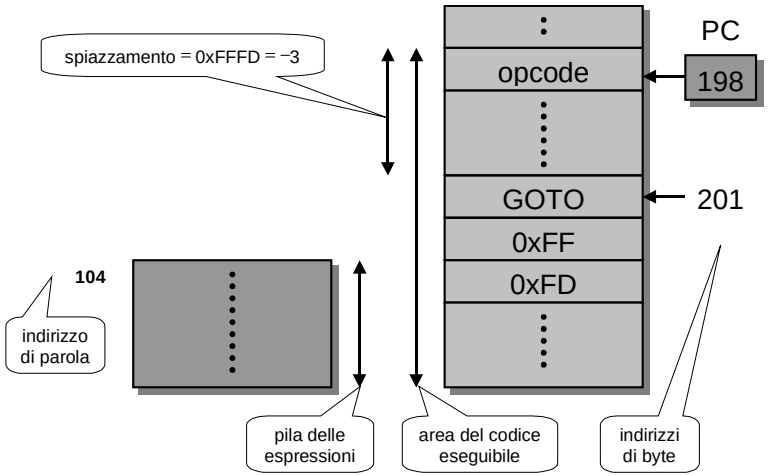
Simulazione di GOTO



Eseguire l'istruzione: GOTO −3

Animazione

Simulazione di GOTO

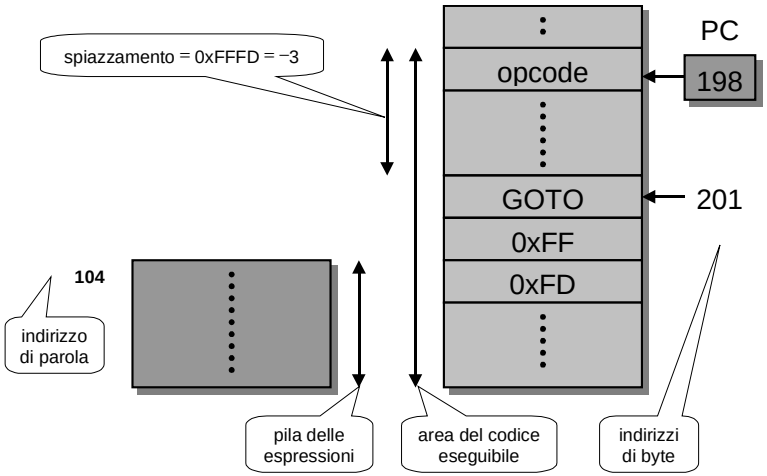


Istruzione GOTO −3 eseguita

Animazione

Fine

Simulazione di GOTO



Istruzione GOTO −3 eseguita

Istruzioni di manipolazione della pila

Hex	Mnemonico	Funzionamento: operazione e risultato
0x10	BIPUSH byte	Push del valore "byte" (8 bit) sulla cima della pila
0x13	LDCW numcost	Push della costante "numcost" sulla pila
0x57	POP	Elimina la parola (32 bit) in cima alla pila
0x59	DUP	Duplica la parola (32 bit) in cima alla pila
0x5F	SWAP	Scambia le 2 parole (32 bit) in cima alla pila

- Questo gruppo di istruzioni è misto, ma per la maggior parte contiene istruzioni che operano sulla pila e sull'area delle costanti, con parole da 32 bit

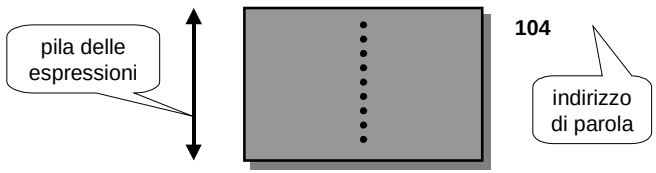
Simulazione di BIPUSH

Animazione

Il byte da scrivere viene esteso (in segno) a formare una parola da 32 bit (vedere l'estensione di segno in complemento a 2)

Simulazione di BIPUSH

Animazione

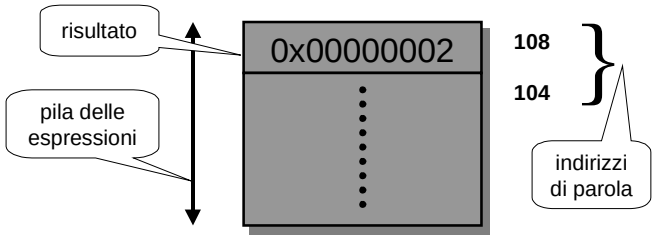


Eseguire l'istruzione: BIPUSH 0x02

Il byte da scrivere viene esteso (in segno) a formare una parola da 32 bit (vedere l'estensione di segno in complemento a 2)

Simulazione di BIPUSH

Animazione



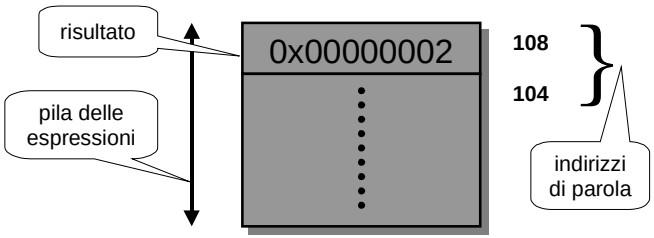
Istruzione BIPUSH 0x02 eseguita

Il byte da scrivere viene esteso (in segno) a formare una parola da 32 bit (vedere l'estensione di segno in complemento a 2)

Simulazione di BIPUSH

Animazione

Fine



Istruzione BIPUSH 0x02 eseguita

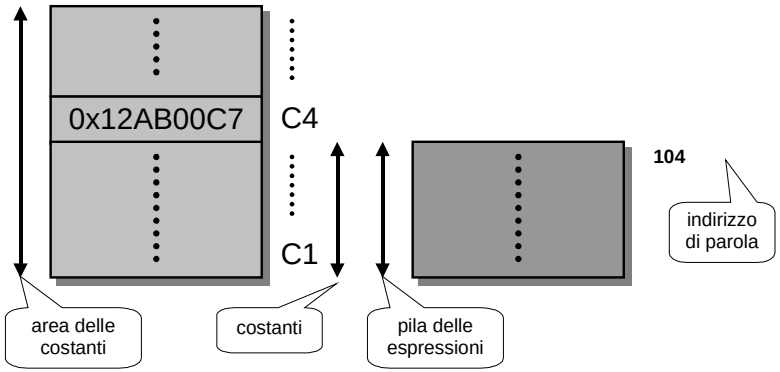
Il byte da scrivere viene esteso (in segno) a formare una parola da 32 bit (vedere l'estensione di segno in complemento a 2)

Simulazione di LDCW

Animazione

Simulazione di LDCW

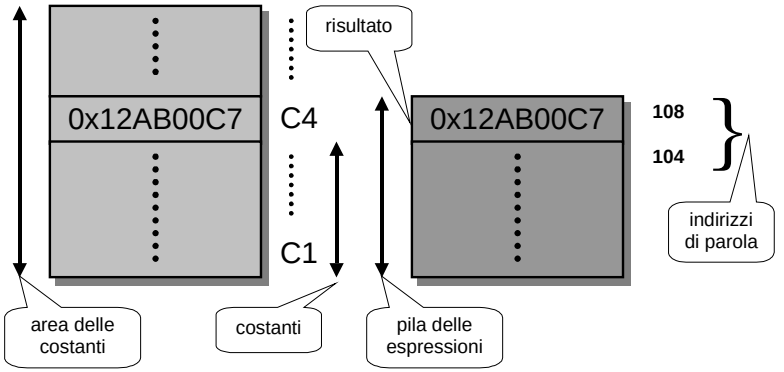
Animazione



Eseguire l'istruzione: LDCW 4

Simulazione di LDCW

Animazione

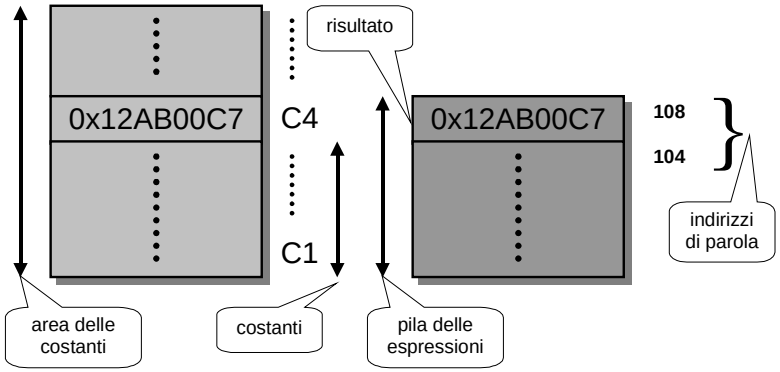


Istruzione LDCW 4 eseguita

Simulazione di LDCW

Animazione

Fine



Istruzione LDCW 4 eseguita

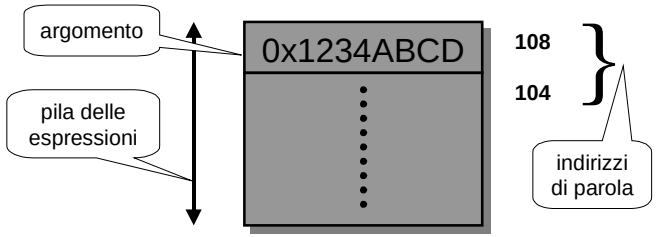
Simulazione di POP

Animazione

Si cancella una parola (32 bit)

Simulazione di POP

Animazione

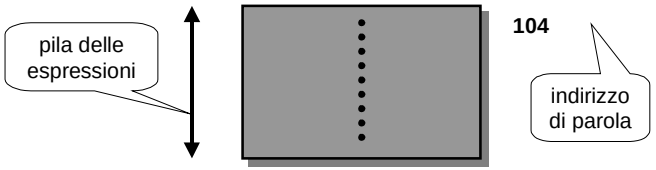


Eseguire l'istruzione POP

Si cancella una parola (32 bit)

Simulazione di POP

Animazione



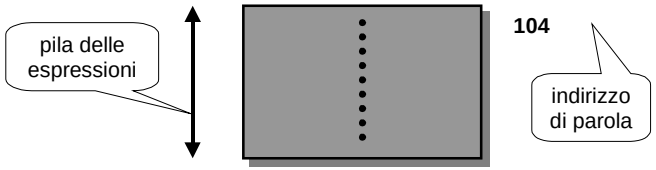
Istruzione POP eseguita

Si cancella una parola (32 bit)

Simulazione di POP

Animazione

Fine



Istruzione POP eseguita

Si cancella una parola (32 bit)

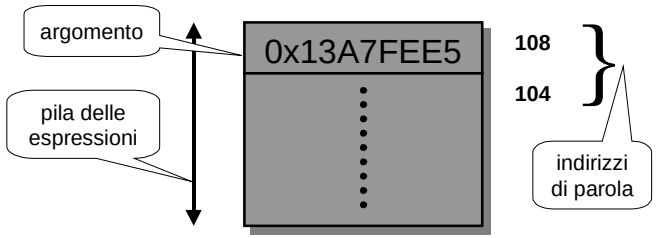
Simulazione di DUP

Animazione

Si duplica una parola (32 bit)

Simulazione di DUP

Animazione

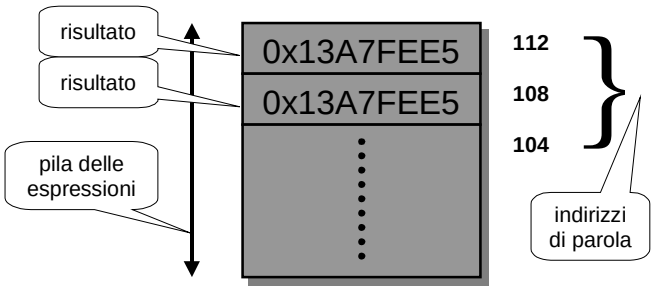


Eseguire l'istruzione DUP

Si duplica una parola (32 bit)

Simulazione di DUP

Animazione



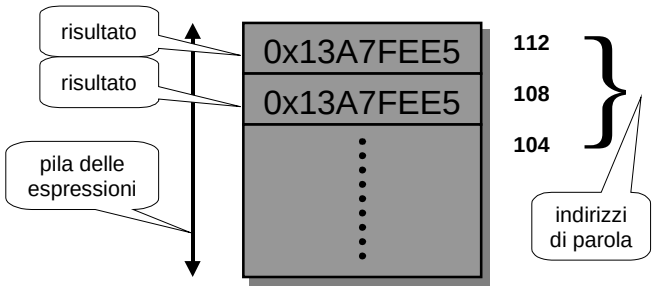
Istruzione DUP eseguita

Si duplica una parola (32 bit)

Simulazione di DUP

Animazione

Fine



Istruzione DUP eseguita

Si duplica una parola (32 bit)

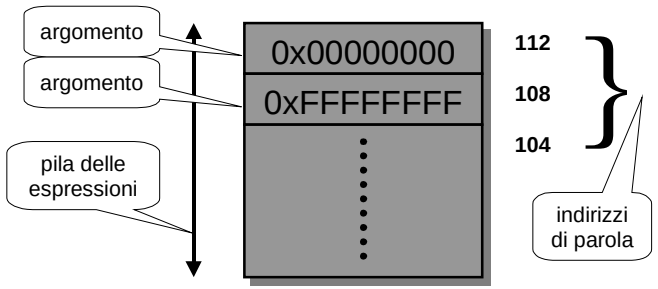
Simulazione di SWAP

Animazione

Si scambiano due parole (32 bit)

Simulazione di SWAP

Animazione

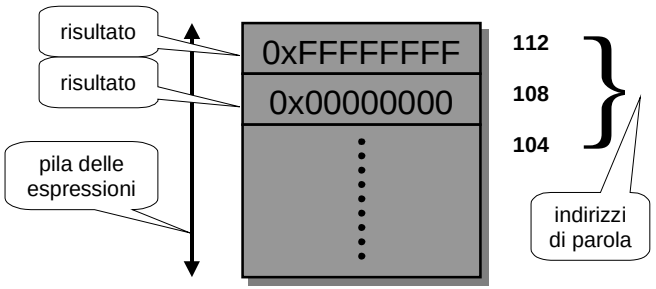


Eseguire l'istruzione SWAP

Si scambiano due parole (32 bit)

Simulazione di SWAP

Animazione



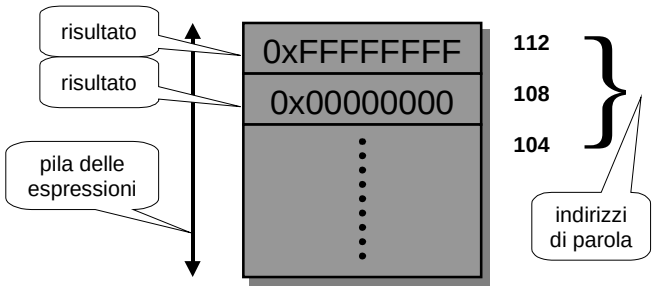
Istruzione SWAP eseguita

Si scambiano due parole (32 bit)

Simulazione di SWAP

Animazione

Fine



Istruzione SWAP eseguita

Si scambiano due parole (32 bit)

Istruzioni di controllo

Hex	Mnemonico	Funzionamento: operazione e risultato
0x00	NOP	Nessuna operazione
0xC4	WIDE	L'istruzione seguente ha un argomento di 16 bit

- Altre possibili istruzioni di controllo:
  - HALT: arresto processore
  - Controllo delle interruzioni
  - ...
- Sono comunque tutte istruzioni molto dipendenti dalla struttura del processore

Ultima slide prima della piazzola di sosta

Istruzioni di I/O

- Il processore IJVM non dispone di istruzioni macchina di ingresso/uscita
- Per comunicare con le unità funzionali di I/O (periferiche), il processore IJVM si serve di una libreria di funzioni Java predefinite, che si usano come dei normali sottoprogrammi
- Altri tipi di processore, comunque, dispongono di istruzioni di I/O, facenti parte del linguaggio macchina

Esempio di programma IJVM

- Si considera un semplice programma Java (ma valido anche in C), che esegue alcuni assegnamenti e un'istruzione condizionale "if-then-else"
- Di questo programma verranno illustrati:
  - il codice sorgente Java
  - la traduzione in linguaggio macchina IJVM
  - la simulazione dell'esecuzione
  - e la codifica del programma in binario

Il programma sorgente Java (o C)

```
I = J + K;      /* assegnamento*/
if (I == 3) {   /* condizione  */
  K = 0;        /* ramo THEN   */
} else {
  J = J – 1;    /* ramo ELSE   */
}               /* fine        */
```

espressione

espressione

espressione

- Si può immaginare questo segmento di programma come corpo di una funzione

Il programma tradotto in IJVM

# Riga	Etichetta	Opcode	Argomenti	Commento
1		ILOAD	J	`I = J + K`
2		ILOAD	K	
3		IADD		
4		ISTORE	I	
5		ILOAD	I	`if (I == 3)`
6		BIPUSH	3	
7		IFCMPEQ	RAMO_THEN	
8	RAMO_ELSE:	ILOAD	J	`J = J – 1`
9		BIPUSH	1	
10		ISUB		
11		ISTORE	J	
12		GOTO	FINE	
13	RAMO_THEN:	BIPUSH	0	`K = 0`
14		ISTORE	K	
15	FINE:	…	…	

Assegna-mento

Valutazione Condizione

Ramo ELSE

Ramo THEN

Simulazione dell'esecuzione (I)

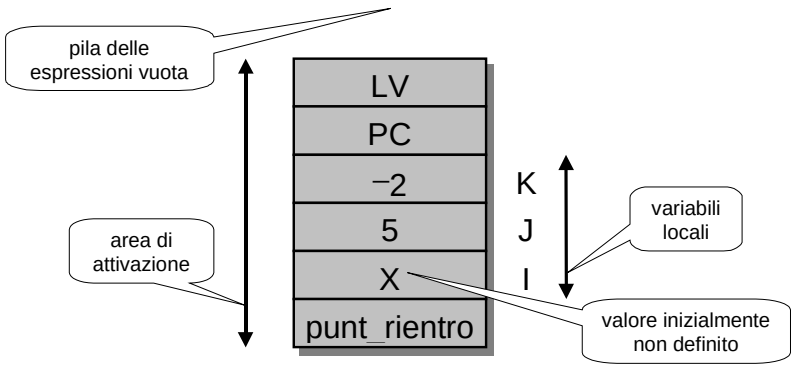
Animazione

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3

Simulazione dell'esecuzione (I)

Animazione

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3

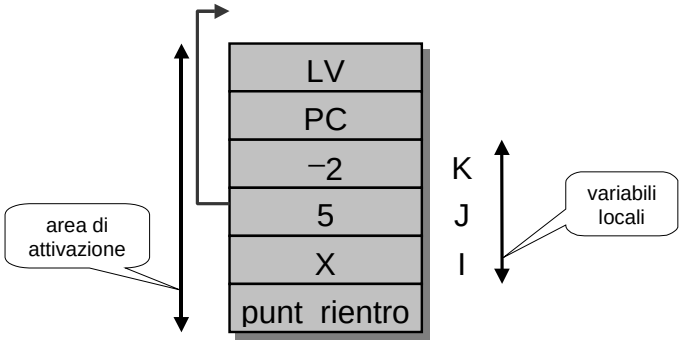


INIZIO: stato di partenza dell'area di attivazione

Animazione

Simulazione dell'esecuzione (I)

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3

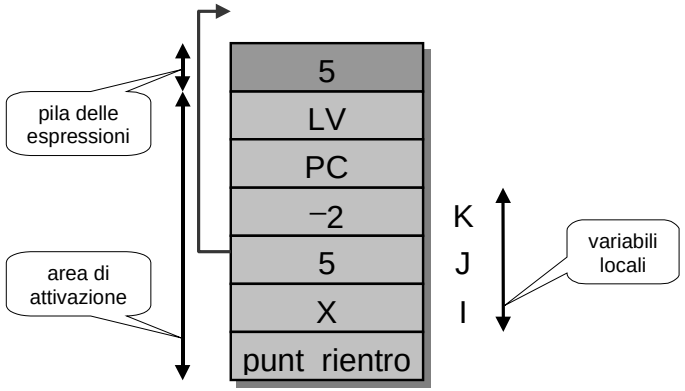


Eseguire l'istruzione: ILOAD J

Animazione

Simulazione dell'esecuzione (I)

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3

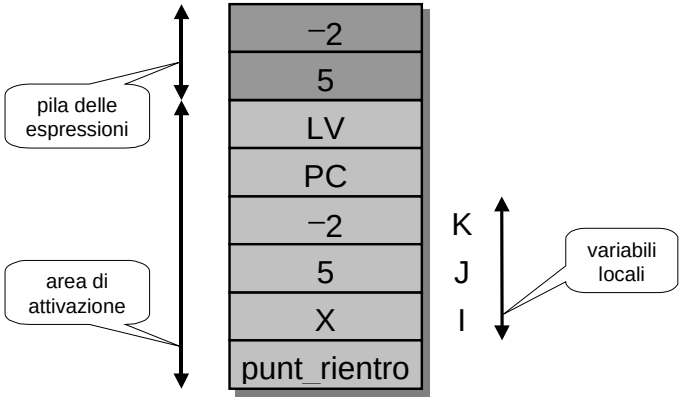


ILOAD J eseguita; eseguire l'istruzione: ILOAD K

Animazione

Simulazione dell'esecuzione (I)

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3

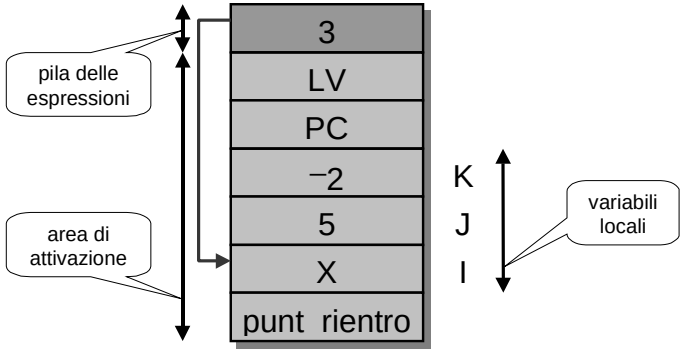


ILOAD K eseguita; eseguire l'istruzione: IADD

Animazione

Simulazione dell'esecuzione (I)

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3

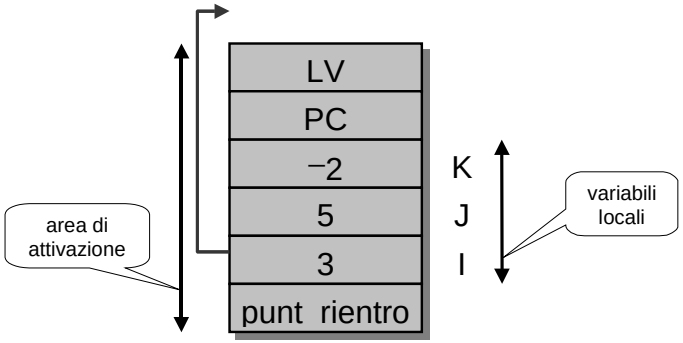


IADD eseguita; eseguire l'istruzione: ISTORE I

Animazione

Simulazione dell'esecuzione (I)

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3

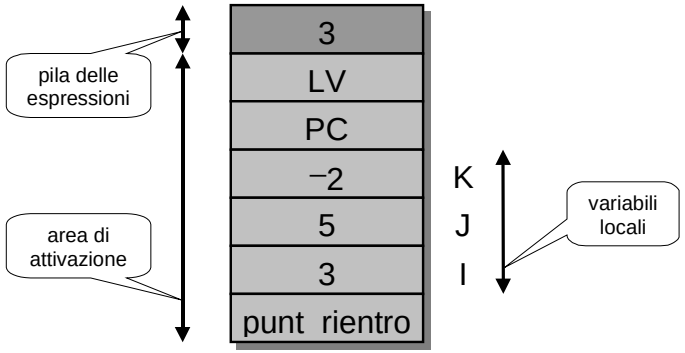


ISTORE I eseguita; eseguire l'istruzione: ILOAD I

Animazione

Simulazione dell'esecuzione (I)

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3

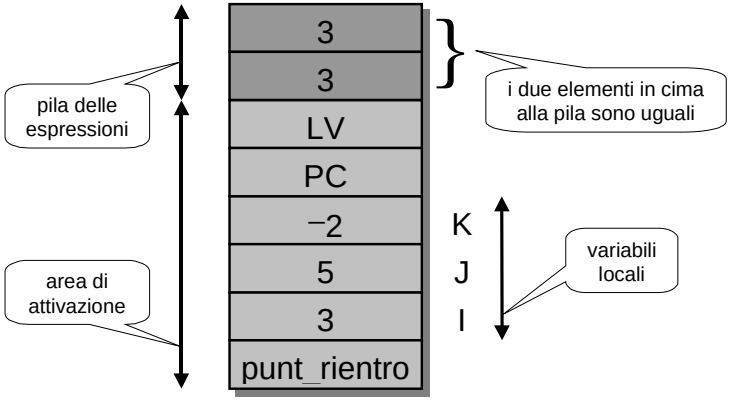


ILOAD I eseguita; eseguire l'istruzione: BIPUSH 3

Animazione

Simulazione dell'esecuzione (I)

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3

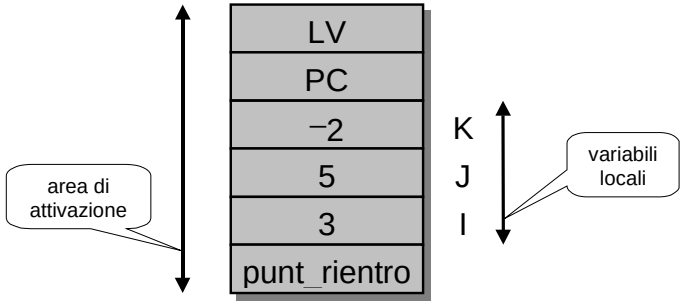


BIPUSH 3 eseguita; eseguire: IFCMPEQ RAMO_THEN

Animazione

Simulazione dell'esecuzione (I)

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3

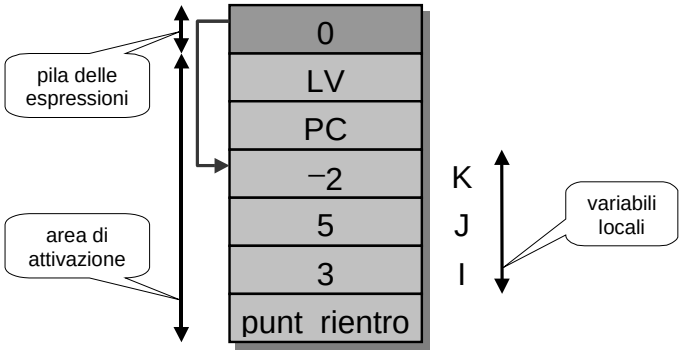


IFCMPEQ eseguita (con successo); eseguire: BIPUSH 0

Animazione

Simulazione dell'esecuzione (I)

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3

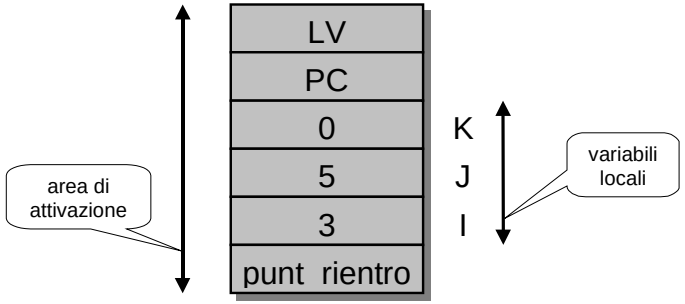


BIPUSH 0 eseguita; eseguire: ISTORE K

Animazione

Simulazione dell'esecuzione (I)

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3

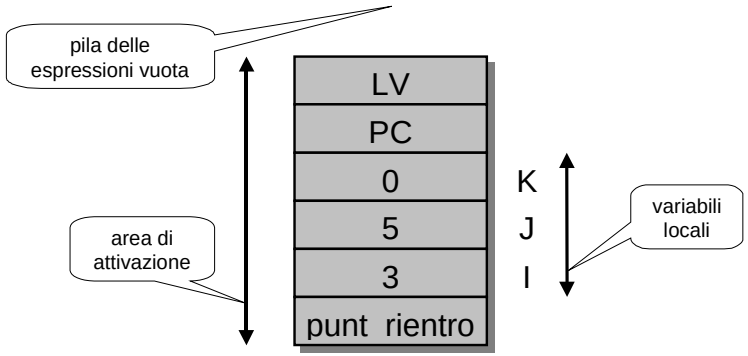


BIPUSH 0 eseguita; eseguire: FINE … (fine del prog.)

Animazione

Simulazione dell'esecuzione (I)

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3

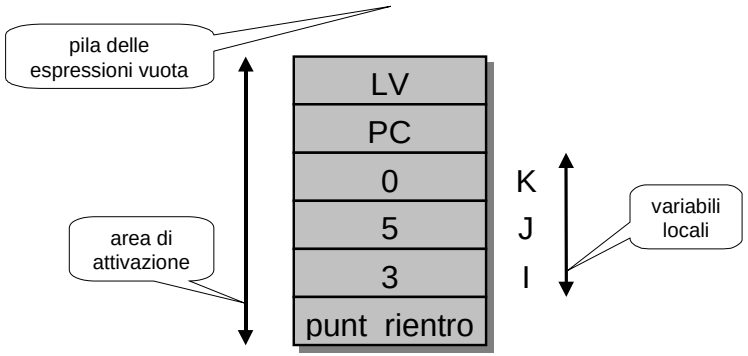


FINE: stato terminale dell'area di attivazione

Animazione

Simulazione dell'esecuzione (I)

caso: J = 5, K = – 2, pertanto I = J + K = 5 – 2 = 3



FINE: stato terminale dell'area di attivazione

Animazione

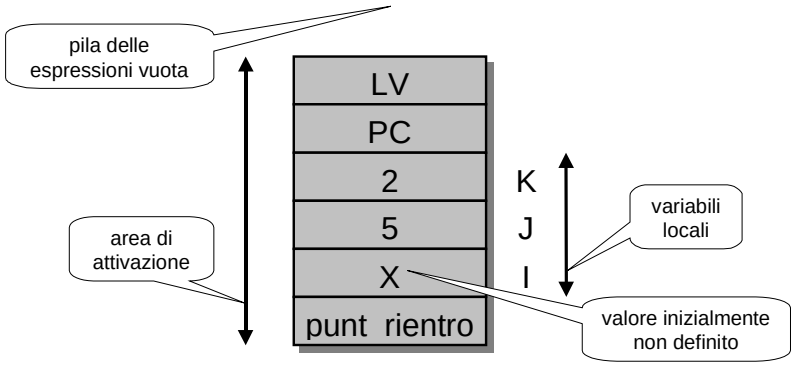
Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

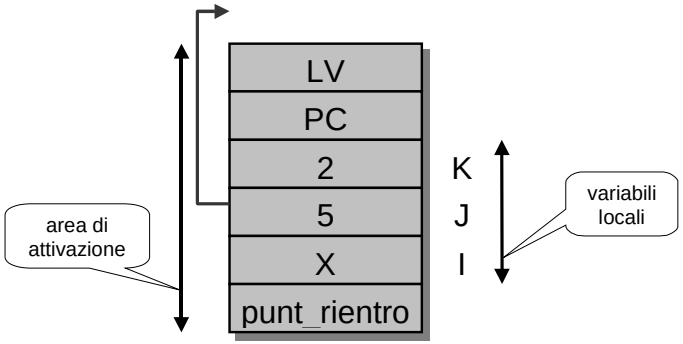


INIZIO: stato di partenza dell'area di attivazione

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

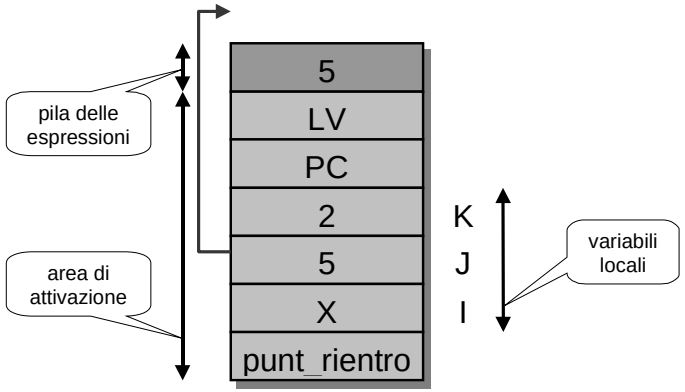


Eseguire l'istruzione: ILOAD J

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

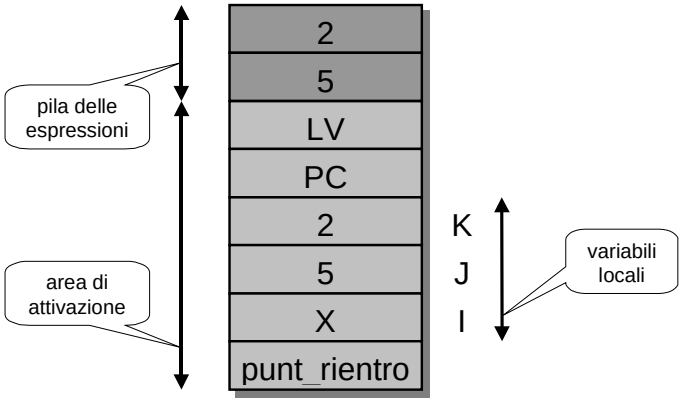


ILOAD J eseguita; eseguire l'istruzione: ILOAD K

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

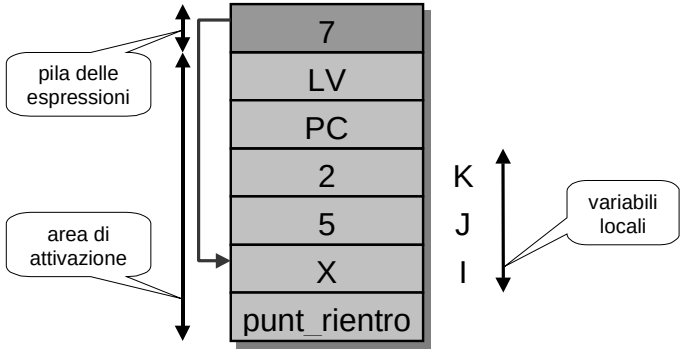


ILOAD K eseguita; eseguire l'istruzione: IADD

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

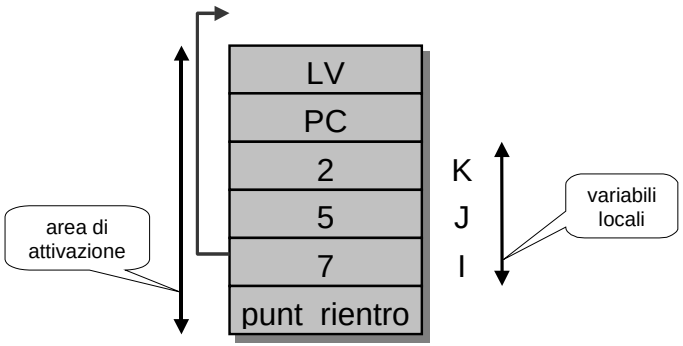


IADD eseguita; eseguire l'istruzione: ISTORE I

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

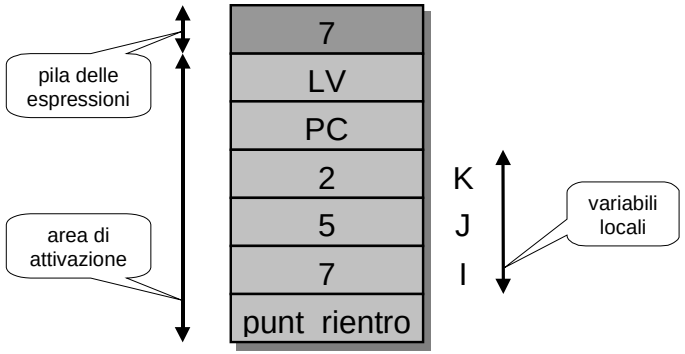


ISTORE I eseguita; eseguire l'istruzione: ILOAD I

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

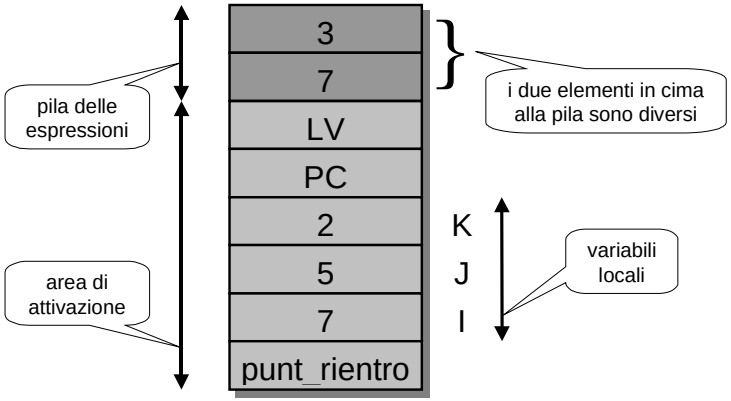


ILOAD I eseguita; eseguire l'istruzione: BIPUSH 3

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

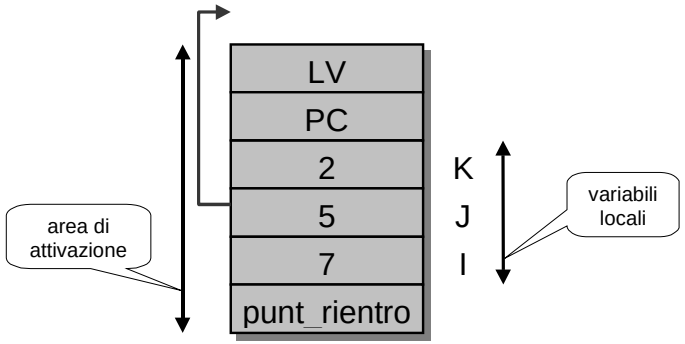


BIPUSH 3 eseguita; eseguire: IFCMPEQ RAMO_THEN

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

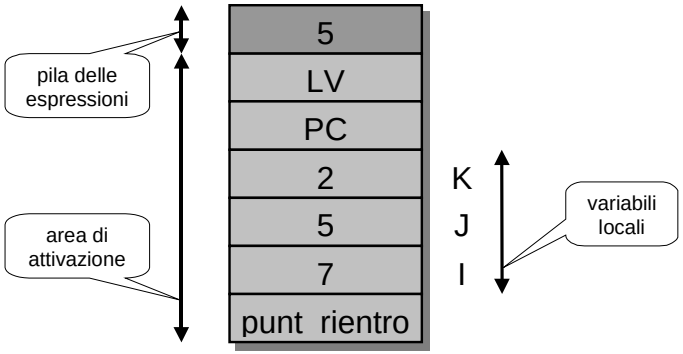


IFCMPEQ eseguita (con fallimento); eseguire: ILOAD J

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

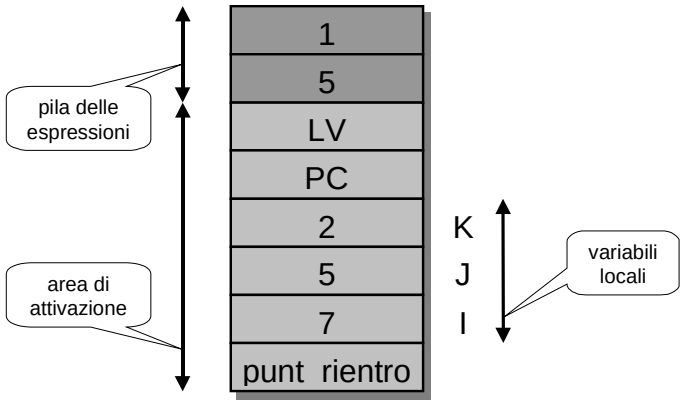


ILOAD J eseguita; eseguire l'istruzione: BIPUSH 1

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

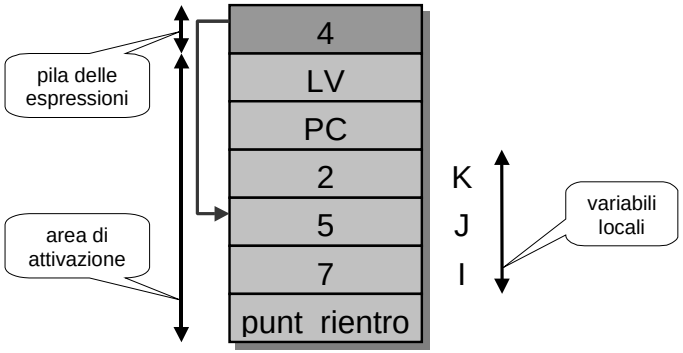


BIPUSH 1 eseguita; eseguire l'istruzione: ISUB

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

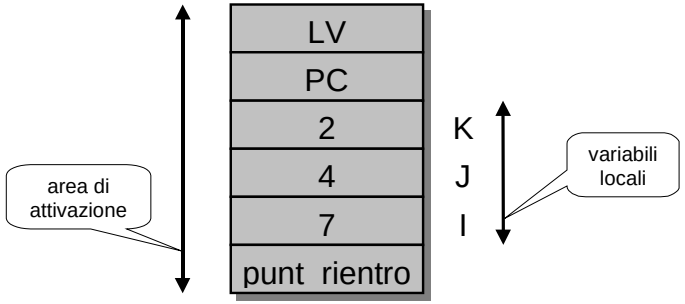


ISUB eseguita; eseguire l'istruzione: ISTORE J

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

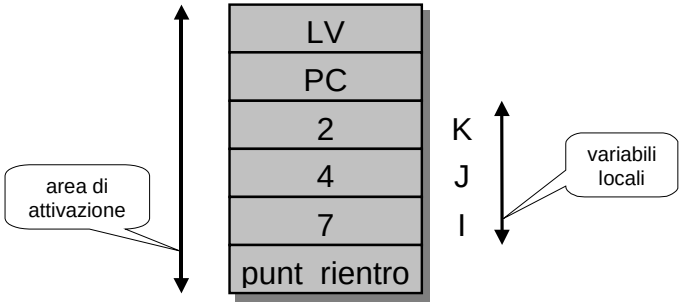


ISTORE J eseguita; eseguire l'istruzione: GOTO FINE

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

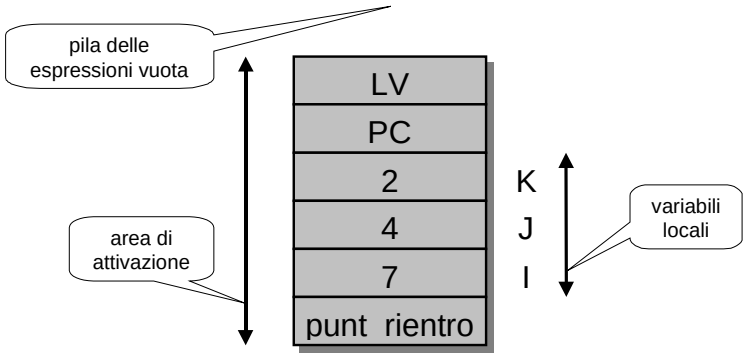


GOTO eseguita; eseguire l'istruzione: FINE … (fine prog.)

Animazione

Simulazione dell'esecuzione (II)

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7



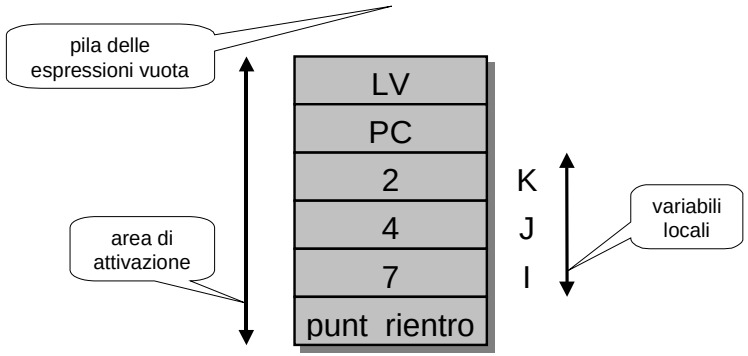
FINE: stato terminale dell'area di attivazione

Simulazione dell'esecuzione (II)

Animazione

Fine

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7



FINE: stato terminale dell'area di attivazione

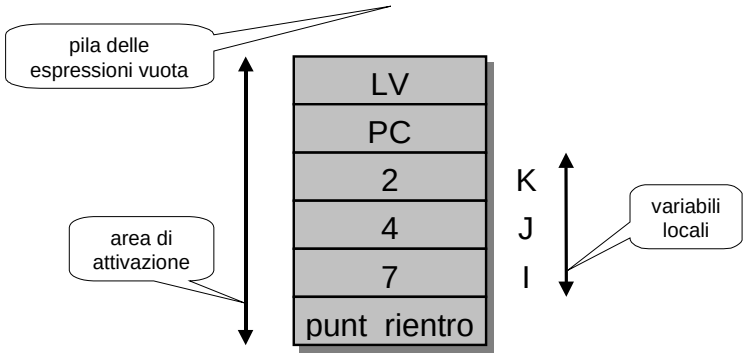
Simulazione dell'esecuzione (II)

Ultima slide prima della piazzola di sosta

Animazione

Fine

caso: J = 5, K = 2, pertanto I = J + K = 5 + 2 = 7

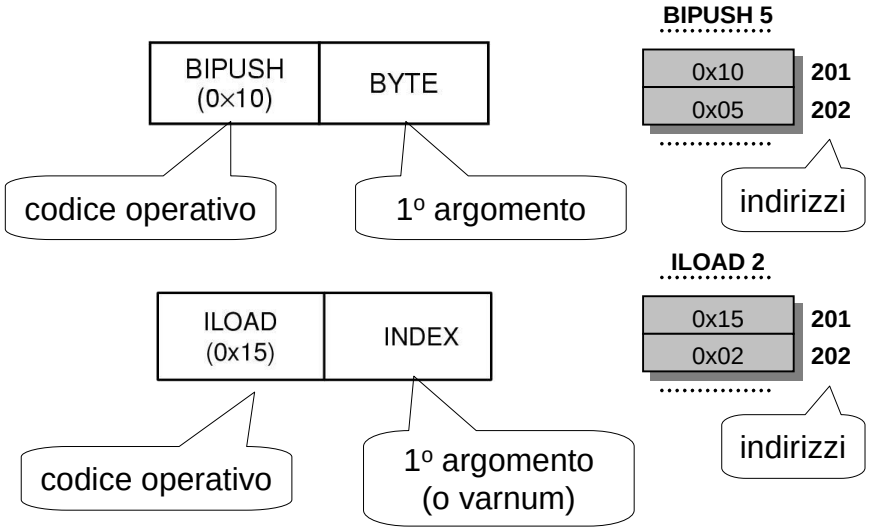


FINE: stato terminale dell'area di attivazione

Codifica delle istruzioni IJVM

- Un'istruzione macchina è formata da:
  - codice operativo (mnemonico, obbligatorio)
  - nessuno, 1, 2 o anche più argomenti
- L'area del codice eseguibile è una successione di byte (8 bit)
- Pertanto, le istruzioni macchina IJVM si codificano in byte: 1, 2, 3 o anche più byte, a seconda dell'istruzione
- Il primo byte è sempre quello che codifica il codice operativo

Esempi di codifica di istruzioni



Codifica del programma IJVM

# Riga	Etichetta	Opcode	Argomenti	Codifica (hex)	Codifica (bin)		
				1° b 2° b 3° b	1° byte	2° byte	3° byte
1		ILOAD	J	0x15 0x02	00010101	00000010	
2		ILOAD	K	0x15 0x03	00010101	00000011	
3		IADD		0x60	01100000		
4		ISTORE	I	0x36 0x01	00110110	00000001	
5		ILOAD	I	0x15 0x01	00010101	00000001	
6		BIPUSH	3	0x10 0x03	00010000	00000011	
7		IFCMPEQ	RAMO_THEN	0x9F 0x00 0x0D	10011111	00000000	00001101
8	RAMO_ELSE:	ILOAD	J	0x15 0x02	00010101	00000010	
9		BIPUSH	1	0x10 0x01	00010000	00000001	
10		ISUB		0x64	01100100		
11		ISTORE	J	0x36 0x02	00110110	00000010	
12		GOTO	FINE	0xA7 0x00 0x07	10100111	00000000	00000111
13	RAMO_THEN:	BIPUSH	0	0x10 0x00	00010000	00000000	
14		ISTORE	K	0x36 0x03	00110110	00000011	
15	FINE:	…	…				

Codifica delle istruzioni macchina IJVM del programma; vi sono istruzioni da 1, 2 e 3 byte

Estensione l'istruzione WIDE

- Con un solo byte di indice (varnum), nelle istruzioni ILOAD e ISTORE si indicano solo fino a 256 variabili locali
- Anteponendo loro l'istruzione speciale WIDE, l'indice viene esteso a 16 bit