Architettura degli Elaboratori – Corso B
Turno di Laboratorio 2

Docente: Claudio Schifanella

Quinta Lezione
IJVM

Esercizio 9 (invoke virtual)

- Si consideri la funzione in Java:

```
int somma(int x, int y) {
int z;
z=x+y;
return z; }
```

- e il seguente frammento di codice:

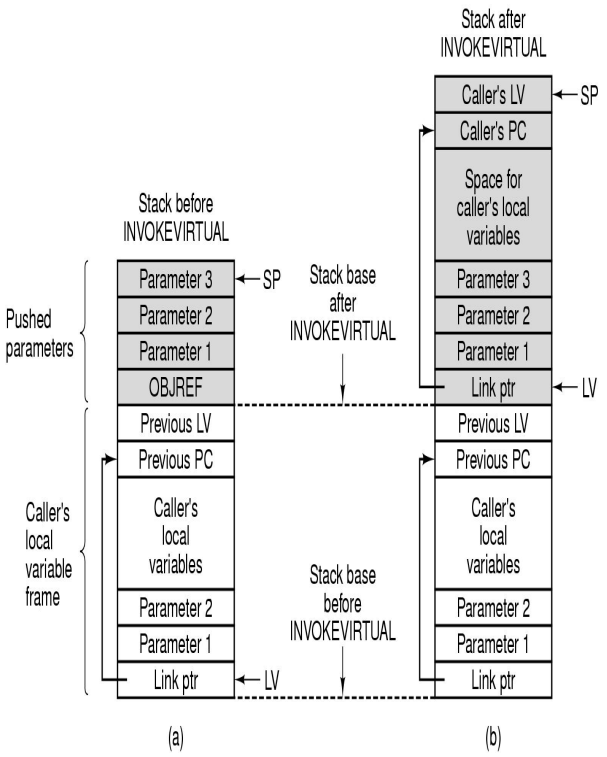
```
int i,j,k;
i=15;
j=i+10;
k=somma(i,j);
```

Esercizio 9 (invokevirtual)

- Codificarle in IJVM, utilizzando come modello il file template.jas.
- Caricare il programma con il simulatore Mic-1, e indicare i valori (in esadecimale e in decimale) dei registri LV, SP
  - subito prima della chiamata alla funzione;
  - all'inizio dell'esecuzione della funzione;
  - subito prima del ritorno dalla funzione;
  - al rientro nel programma principale.

Il microprogramma Mic-1

```
invokevirtual1  PC = PC + 1; fetch      // MBR = index byte 1; inc. PC, get 2nd byte
invokevirtual2  H = MBRU << 8           // Shift and save first byte in H
invokevirtual3  H = MBRU OR H           // H = offset of method pointer from CPP
invokevirtual4  MAR = CPP + H; rd       // Get pointer to method from CPP area
invokevirtual5  OPC = PC + 1            // Save Return PC in OPC temporarily
invokevirtual6  PC = MDR; fetch         // PC points to new method; get param count
invokevirtual7  PC = PC + 1; fetch      // Fetch 2nd byte of parameter count
invokevirtual8  H = MBRU << 8           // Shift and save first byte in H
invokevirtual9  H = MBRU OR H           // H = number of parameters
invokevirtual10 PC = PC + 1; fetch      // Fetch first byte of # locals
invokevirtual11 TOS = SP - H            // TOS = address of OBJREF - 1
invokevirtual12 TOS = MAR = TOS + 1     // TOS = address of OBJREF (new LV)
invokevirtual13 PC = PC + 1; fetch      // Fetch second byte of # locals
invokevirtual14 H = MBRU << 8           // Shift and save first byte in H
invokevirtual15 H = MBRU OR H           // H = # locals
invokevirtual16 MDR = SP + H + 1; wr    // Overwrite OBJREF with link pointer
invokevirtual17 MAR = SP = MDR;         // Set SP, MAR to location to hold old PC
invokevirtual18 MDR = OPC; wr           // Save old PC above the local variables
invokevirtual19 MAR = SP = SP + 1       // SP points to location to hold old LV
invokevirtual20 MDR = LV; wr            // Save old LV above saved PC
invokevirtual21 PC = PC + 1; fetch      // Fetch first opcode of new method.
invokevirtual22 LV = TOS; goto Main1    // Set LV to point to LV Frame
```

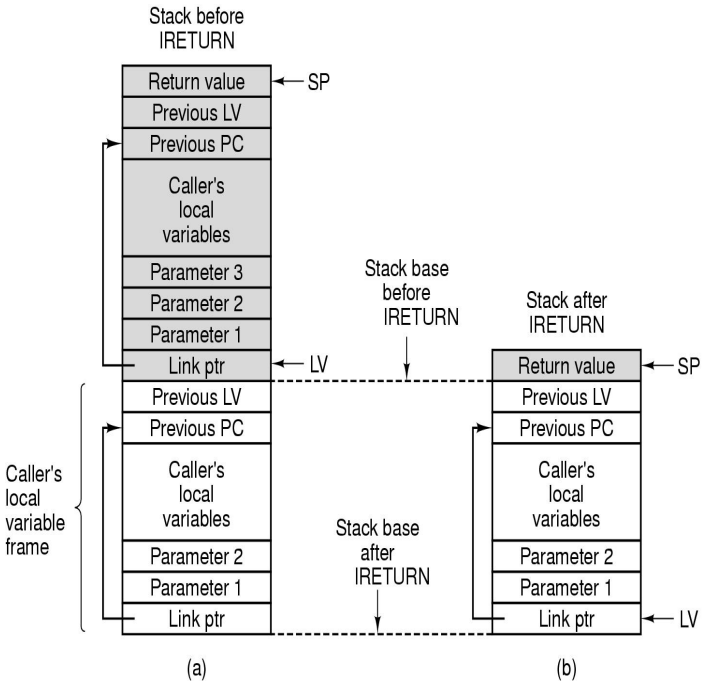


2 byte!

- ***INVOKEVIRTUAL disp***: chiama un metodo

Il microprogramma Mic-1

```
ireturn1        MAR = SP = LV; rd       // Reset SP, MAR to get link pointer
ireturn2                                // Wait for read
ireturn3        LV = MAR = MDR; rd      // Set LV to link ptr; get old PC
ireturn4        MAR = LV + 1            // Set MAR to read old LV
ireturn5        PC = MDR; rd; fetch     // Restore PC; fetch next opcode
ireturn6        MAR = SP                // Set MAR to write TOS
ireturn7        LV = MDR                // Restore LV
ireturn8        MDR = TOS; wr; goto Main1       // Save return value on orig
                                                   inal top of stack
```



- ***IRETURN***: ritorno da un metodo con un valore intero

Esercizio 10:

- Scrivere un programma IJVM **ricorsivo** per il calcolo dell'ennesimo numero della successione di Fibonacci (1,1,2,3,5,8,13,21,34,55….), dove ogni numero della successione è il risultato della somma dei 2 numeri precedenti. A tal proposito tenere conto della definizione ricorsiva della successione:

```
fib (1) = 1
fib (2) = 1
fib (n) = fib (n-1) + fib (n-2)
```

- Assemblare e caricare il programma nel simulatore mic1MMV al fine di seguire l'andamento dell'esecuzione e dell'evoluzione dello stack a livello di singola istruzione IJVM.

Struttura soluzione Esercizio 10:

```
.constant
objref  0xCAFE  // may be any value.  Needed by invokevirtual.
a               9
.end-constant

.main
.var
i
.end-var
                LDC_W objref
                LDC_W a
                INVOKEVIRTUAL fib
                ISTORE i
                HALT
.end-main

.method fib(n)
.var
.end-var
```

Struttura soluzione Esercizio 10:

.method fib(n)

se n = 1 vai a return (IF_ICMPEQ return assumendo che 1 sia sullo stack)

se n = 2 vai a return (IF_ICMPEQ return assumendo che 2 sia sullo stack)

metti objref sullo stack (LDC_W objref)

calcola n-1 e mettilo sullo stack (isub assumendo che n e 1 siano sullo stack → iload n seguito da bipush 1 e da isub

invokevirtual fib // fib(n-1)

metti objref sullo stack

calcola n-2 e mettilo sullo stack

invokevirtual fib // fib(n-2)

somma i due valori in cima allo stack lasciando il risultato in stack

ireturn

return: bipush 1

ireturn

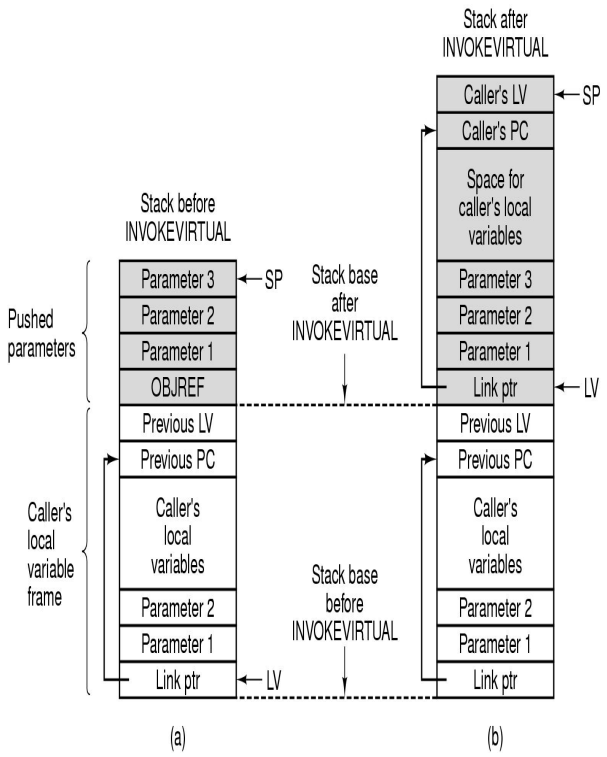
Struttura soluzione esercizio 10:

- Se n = 1 return 1 (cioè salta ad una etichetta in cui si esegue una return restituendo 1 come valore della funzione)
- Se n = 2 return 1
- Altrimenti vai in ricorsione,
  - Metti objref sullo stack, metti n-1 sullo stack e richiama il metodo fib (questa chiamata lascerà il valore di fib(n-1) sullo stack)
  - Metti objref sullo stack, metti n-2 sullo stack e richiama il metodo fib (questa chiamata lascerà il valore di fib(n-2) sullo stack)
  - Somma i valori sullo stack e restituisci il risultato

```
fib (1) = 1
fib (2) = 1
fib (n) = fib (n-1) + fib (n-2)
```

Microinterprete invokevirtual

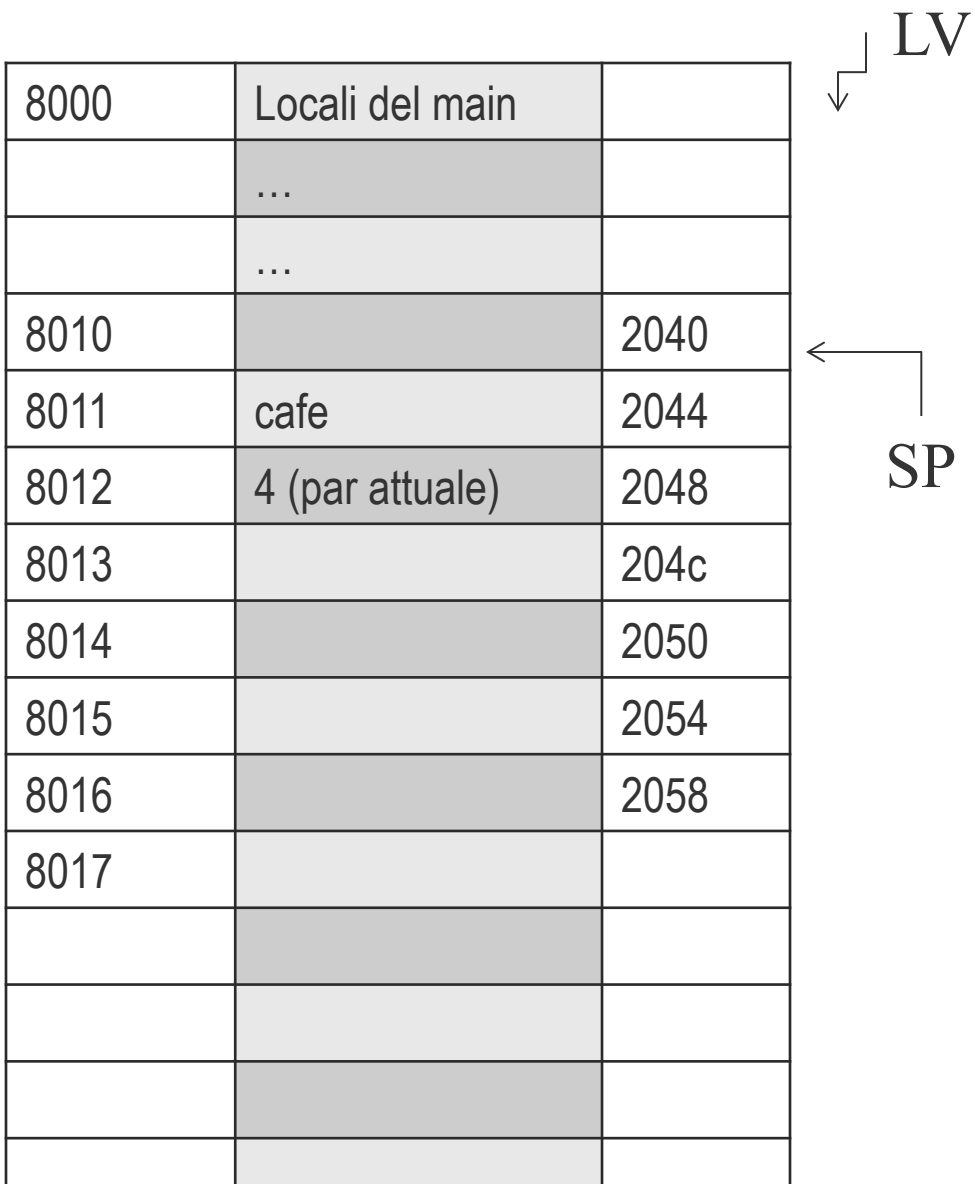
```
invokevirtual1  PC = PC + 1; fetch      // MBR = index byte 1; inc. PC, get 2nd byte
invokevirtual2  H = MBRU << 8           // Shift and save first byte in H
invokevirtual3  H = MBRU OR H           // H = offset of method pointer from CPP
invokevirtual4  MAR = CPP + H; rd       // Get pointer to method from CPP area
invokevirtual5  OPC = PC + 1            // Save Return PC in OPC temporarily
invokevirtual6  PC = MDR; fetch         // PC points to new method; get param count
invokevirtual7  PC = PC + 1; fetch      // Fetch 2nd byte of parameter count
invokevirtual8  H = MBRU << 8           // Shift and save first byte in H
invokevirtual9  H = MBRU OR H           // H = number of parameters
invokevirtual10 PC = PC + 1; fetch      // Fetch first byte of # locals
invokevirtual11 TOS = SP - H            // TOS = address of OBJREF - 1
invokevirtual12 TOS = MAR = TOS + 1     // TOS = address of OBJREF (new LV)
invokevirtual13 PC = PC + 1; fetch      // Fetch second byte of # locals
invokevirtual14 H = MBRU << 8           // Shift and save first byte in H
invokevirtual15 H = MBRU OR H           // H = # locals
invokevirtual16 MDR = SP + H + 1; wr    // Overwrite OBJREF with link pointer
invokevirtual17 MAR = SP = MDR;         // Set SP, MAR to location to hold old PC
invokevirtual18 MDR = OPC; wr           // Save old PC above the local variables
invokevirtual19 MAR = SP = SP + 1       // SP points to location to hold old LV
invokevirtual20 MDR = LV; wr            // Save old LV above saved PC
invokevirtual21 PC = PC + 1; fetch      // Fetch first opcode of new method.
invokevirtual22 LV = TOS; goto Main1    // Set LV to point to LV Frame
```



2 byte!

- ***INVOKEVIRTUAL disp***: chiama un metodo

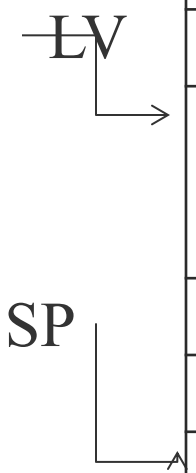
Osservazione dello stack per n=4 (fib(4) = 3):



8000	Locali del main	
	…	
	…	
8010		2040
8011	cafe	2044
8012	4 (par attuale)	2048
8013		204c
8014		2050
8015		2054
8016		2058
8017		

Osservazione dello stack:

8000	Locali del main	
	…	
	…	
8010		2040
8011	cafe	2044
8012	4 (par attuale)	2048
8013		204c
8014		2050
8015		2054
8016		2058
8017		



8000	Locali del main	
	…	
	…	
8010		2040
8011	8013 (punta alla locazione con ind ritorno)	2044
8012	4 (par attuale)	2048
8013	9 (Ind ritorno)	204c
8014	8000 (LV ritorno)	2050
8015		2054
8016		2058
8017		205c

Osservazione dello stack:

8000	Locali del main	
	…	
	…	
8010		20040
8011	8015 (punta alla locazione con ind ritorno)	20044
8012	4 (par attuale)	20048
8013	0 (var locale k)	2004c
8014	0 (var locale j)	20050
8015	9 (Ind ritorno)	20054
8016	8000 (LV ritorno)	20058
8017	cafe	2005c
8018	3	20060
8019		20064
801a		20068
801b		2006c

LV

SP

8011	8015 (punta alla locazione con ind ritorno)	**20044**
8012	4 (par attuale)	20048
8013	0 (var locale k)	2004c
8014	0 (var locale j)	20050
8015	9 (Ind ritorno)	20054
8016	8000 (LV ritorno)	20058
8017	801b (punta alla locazione con ind ritorno della seconda chiamata di fib)	2005c
8018	3	20060
8019	vuoto (var loc k)	20064
801a	vuoto (var locale j)	20068
801b	29 (ind ritorno)	2006c
801c	8011 (LV ritorno)	20070

Osservazione dello stack per n=1 (fib(1) = 1):

8000	Locali del main	
	…	
	…	
8010		20040
8011	cafe	20044
8012	1 (par attuale)	20048
8013		2004c
8014		20050
8015		20054
8016		20058
8017		

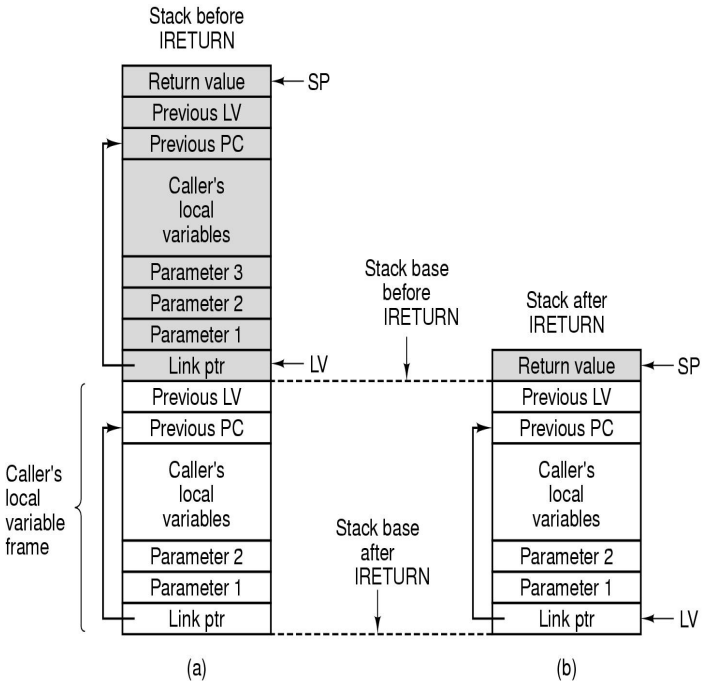
LV

SP

8000	Locali del main	
	…	
	…	
8010		20040
8011	8015 (punta alla locazione con ind ritorno)	20044
8012	1 (par attuale)	20048
8013	0 (var locale k)	2004c
8014	0 (var locale j)	20050
8015	9 (Ind ritorno)	20054
8016	8000 (LV ritorno)	20058
8017		2005c

Codice ireturn

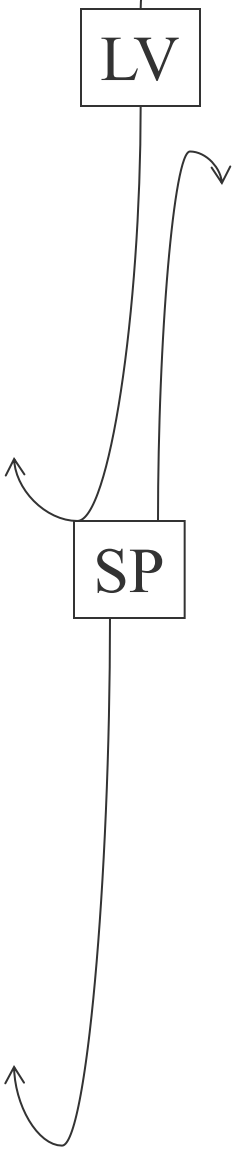
```
ireturn1        MAR = SP = LV; rd       // Reset SP, MAR to get link pointer
ireturn2                                // Wait for read
ireturn3        LV = MAR = MDR; rd      // Set LV to link ptr; get old PC
ireturn4        MAR = LV + 1            // Set MAR to read old LV
ireturn5        PC = MDR; rd; fetch     // Restore PC; fetch next opcode
ireturn6        MAR = SP                // Set MAR to write TOS
ireturn7        LV = MDR                // Restore LV
ireturn8        MDR = TOS; wr; goto Main1       // Save return value on orig
                                                   inal top of stack
```



- ***IRETURN***: ritorno da un metodo con un valore intero

Prima e dopo la ireturn

8000	Locali del main	
	…	
	…	
8010		20040
8011	8015 (punta alla locazione con ind ritorno)	20044
8012	1 (par attuale)	20048
8013	0 (var locale k)	2004c
8014	0 (var locale j)	20050
8015	9 (Ind ritorno)	20054
8016	8000 (LV ritorno)	20058
8017	1 (valore di ritorno su stack	2005c



8000	Locali del main	
	…	
	…	
8010		20040
8011	1 (valore restituito al chiamante)	20044
8012		20048
8013		2004c
8014		20050
8015		20054
8016		20058
8017		2005c

Esercizio 11:

- Scrivere un programma IJVM per il calcolo del **massimo comun divisore** di due numeri interi positivi *a* e *b*. A tale scopo implementare l'algoritmo di Euclide come metodo *MCD(a,b)* da richiamare nel main. L'algoritmo di Euclide in pseudo-codice è il seguente:

```
function MCD(a, b)
    while a <> b
        if a > b
            a := a - b
        else
            b := b - a
    return a
```

Esercizio 12:

- Scrivere un programma IJVM per il calcolo del minimo comune multiplo di due numeri interi positivi *a* e *b*, *mcm(a,b)*, utilizzando la seguente relazione:

$$mcm(\mathrm{a},\mathrm{b}) = \frac{a * b}{MCD(a,b)}$$

dove *MCD(a,b)* è il massimo comun divisore di a e b.

Suggerimento: per calcolare il quoziente tra due numeri si conta quante volte è possibile sottrarre il divisore al dividendo, fino a quando la differenza ottenuta risulta maggiore o uguale al divisore.

Esercizio 13:

- Scrivere un programma IJVM per moltiplicare due numeri interi positivi *a* e *b*. A tale scopo implementare un metodo *mult(a,b)* da richiamare nel main.