

Architettura degli Elaboratori – Corso B

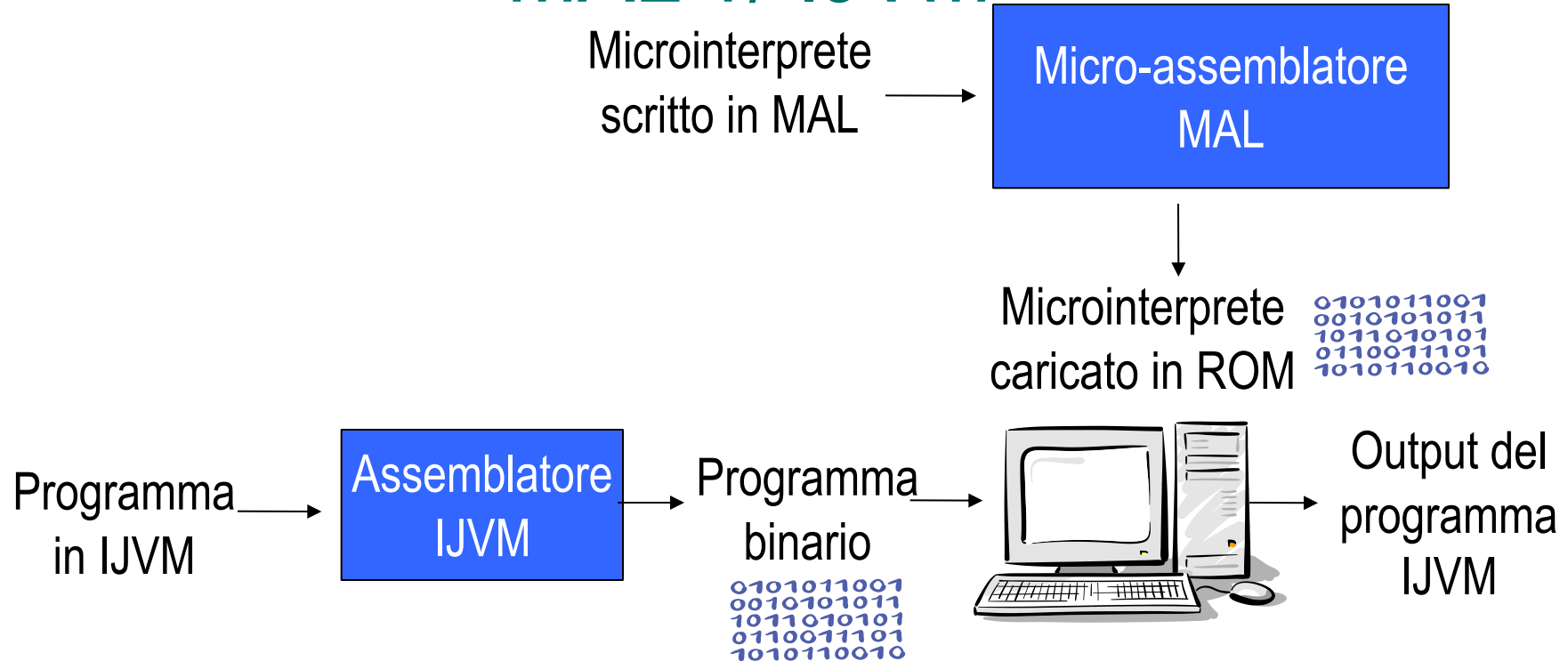
Turno di Laboratorio 2

Docente: Claudio Schifanella

Terza Lezione

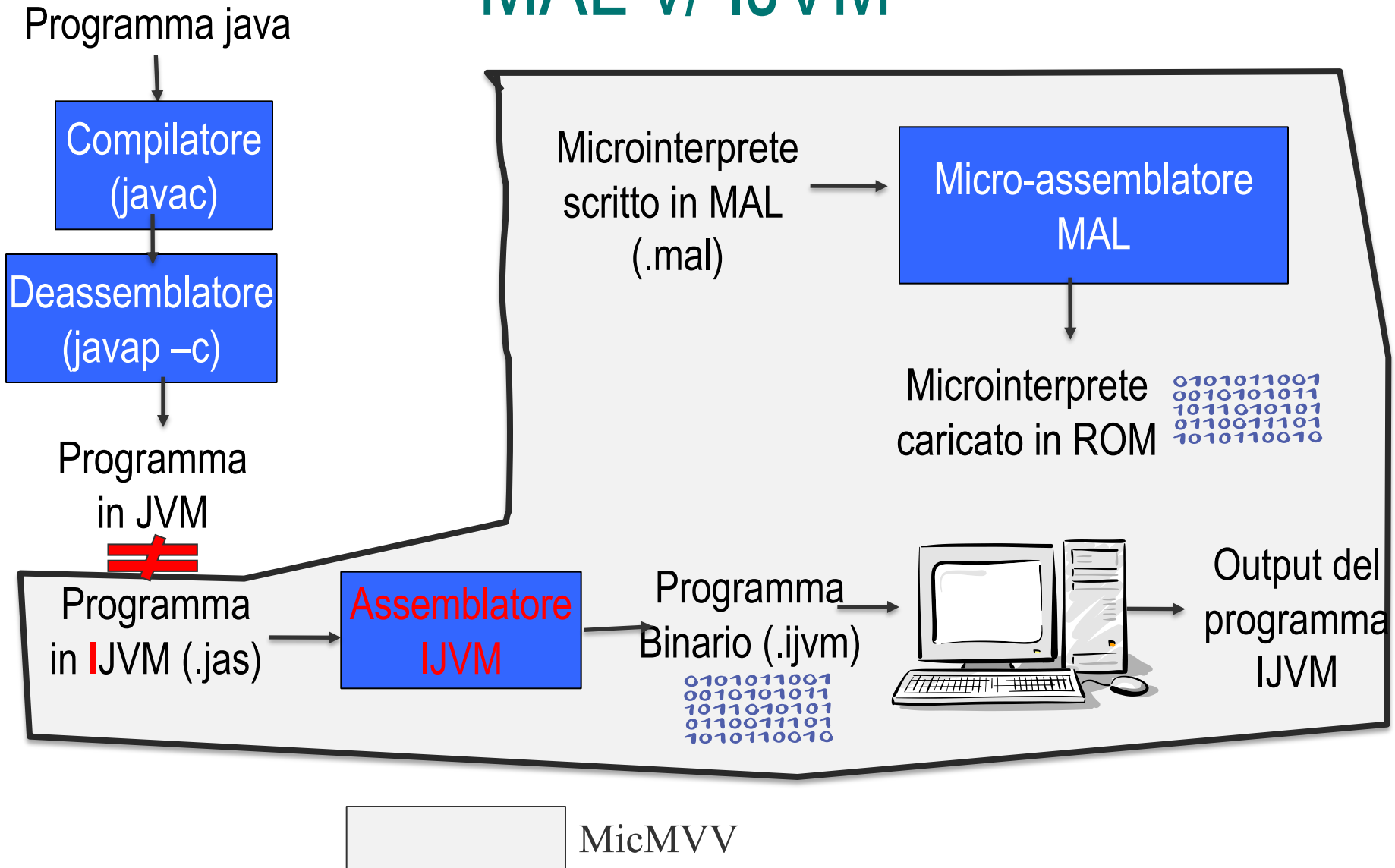
IJVM

MAL v/ IJVM



- Il microinterprete interpreta, pilotando il data-path, il programma scritto in IJVM
- Sia MAL che IJVM sono assemblati da opportuni programmi (l'assemblatore e il micro-assemblatore)

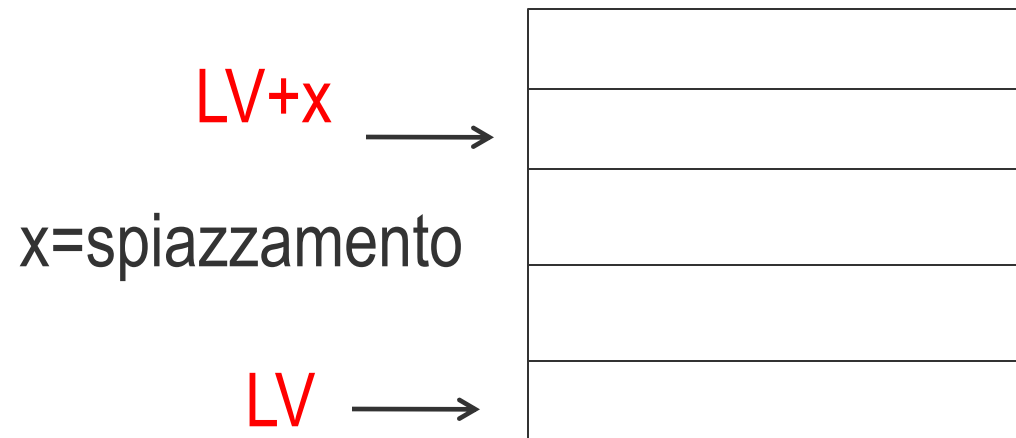
MAL v/ IJVM



- **Modello di memoria IJVM (visto a lezione)**
- Traduzione programmi Java in JVM (con i programmi della suite Java)
- Traduzione programmi Java in IJVM (per l'emulatore MicMVV
 - Semplici programmi in cui ci sono operazioni matematiche su valori
 - Uso della stack

Modello di memoria di IJVM (I)

- Parole da 4 byte
- Le istruzioni IJVM che accedono alla RAM usano indirizzi impliciti che costituiscono la base per l'accesso con indici...



In Mic1MMV, LV=0x8000 punta alla locazione di memoria 0x20000

Modello di memoria di JVM (II)



Modello di memoria di JVM (III)

CPP



Porzione costante (variabili globali)

In Mic1MMV, CPP=0x4000 punta alla locazione di memoria 0x10000

PC



Area dei metodi

In Mic1MMV, PC=0x0000 punta alla locazione di memoria 0x0000

Modello di memoria di JVM (IV)

- Puntatori alla parola (4 byte)
 - LV (blocco corrente var. locali)
 - SP (stack)
 - CPP (porzione costante)
- I loro “scostamenti” sono espressi come numero di parole
- Puntatori al byte (scostamenti espressi come numero di byte)
 - PC (area metodi)

Il set di istruzioni di JVM (IJVM)

Esa	Nome mnemonico	Significato
0x10	BIPUSH <i>byte</i>	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 <i>offset</i>	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 <i>offset</i>	Estrae una parola dalla cima dello stack ed effettua una diramazione se ha valore zero
0x9B	IFLT <i>offset</i>	Estrae una parola dalla cima dello stack ed effettua una diramazione se ha valore negativo
0x9F	IF_ICMPEQ <i>offset</i>	Estrae le due parole in cima allo stack ed effettua una diramazione se sono uguali
0x84	IINC <i>varnum const</i>	Aggiunge una costante a una variabile locale
0x15	ILOAD <i>varnum</i>	Scrive una variabile locale in cima allo stack
0xB6	INVOKEVIRTUAL <i>disp</i>	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

Insieme d'istruzioni di IJVM. Gli operandi *byte*, *const* e *varnum* sono lunghi 1 byte. Gli operandi *disp*, *index* e *offset* sono lunghi 2 byte.

Il set di istruzioni di JVM (IJVM)

Es	Nome mnemonico	Significato
0x36	ISTORE <i>varnum</i>	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 <i>index</i>	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

Insieme d'istruzioni di IJVM. Gli operandi byte, const e varnum sono lunghi 1 byte. Gli operandi disp, index e offset sono lunghi 2 byte.

Alcune istruzioni su interi di JVM (IJVM)

CodOp(Esa)	Istruzioni	Significato
0x60	IADD	Legge le due parole in cima allo stack (2 pop) e scrive la loro somma in cima allo stack (push)
0x64	ISUB	Legge le due parole in cima allo stack (2 pop) e scrive la loro differenza in cima allo stack (push)
0x15	ILOAD x	Scrive in cima allo stack (push) una parola proveniente dal blocco corrente delle variabili locali (x è lo spiazzamento), operando lungo 1 byte
0x36	ISTORE x	Legge una parola dalla cima dello stack (pop) e la memorizza nel blocco delle variabili locali (x è lo spiazzamento), operando lungo 1 byte
0x10	BIPUSH a	Scrive l'operando a in cima allo stack (push), operando lungo 1 byte
0xA7	GOTO offset	Diramazione incondizionata. Il valore di PC (area metodi) viene modificato in base allo spiazzamento (offset di 2 byte)
0x9F	IF_CMPEQ offset	Legge le due parole in cima allo stack (2 push) ed effettua una diramazione se sono uguali (offset di 2 byte)

- Modello di memoria IJVM (visto a lezione)
- Traduzione programmi Java in JVM (con i programmi della suite Java)
- Traduzione programmi Java in IJVM (per l'emulatore MicMVV
 - Semplici programmi in cui ci sono operazioni matematiche su valori
 - Uso della stack

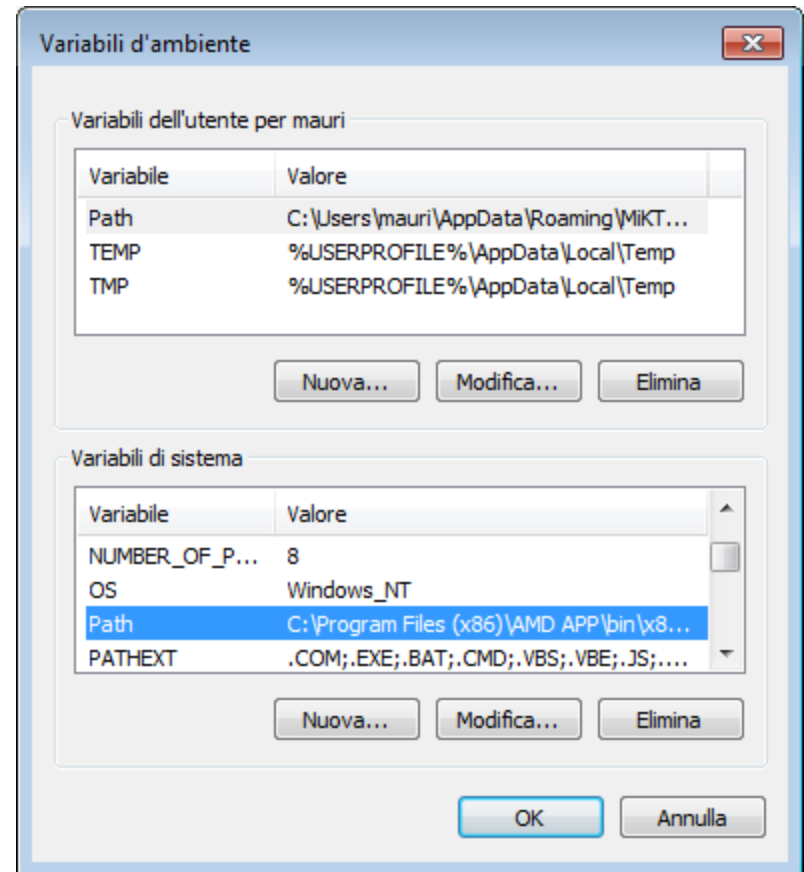
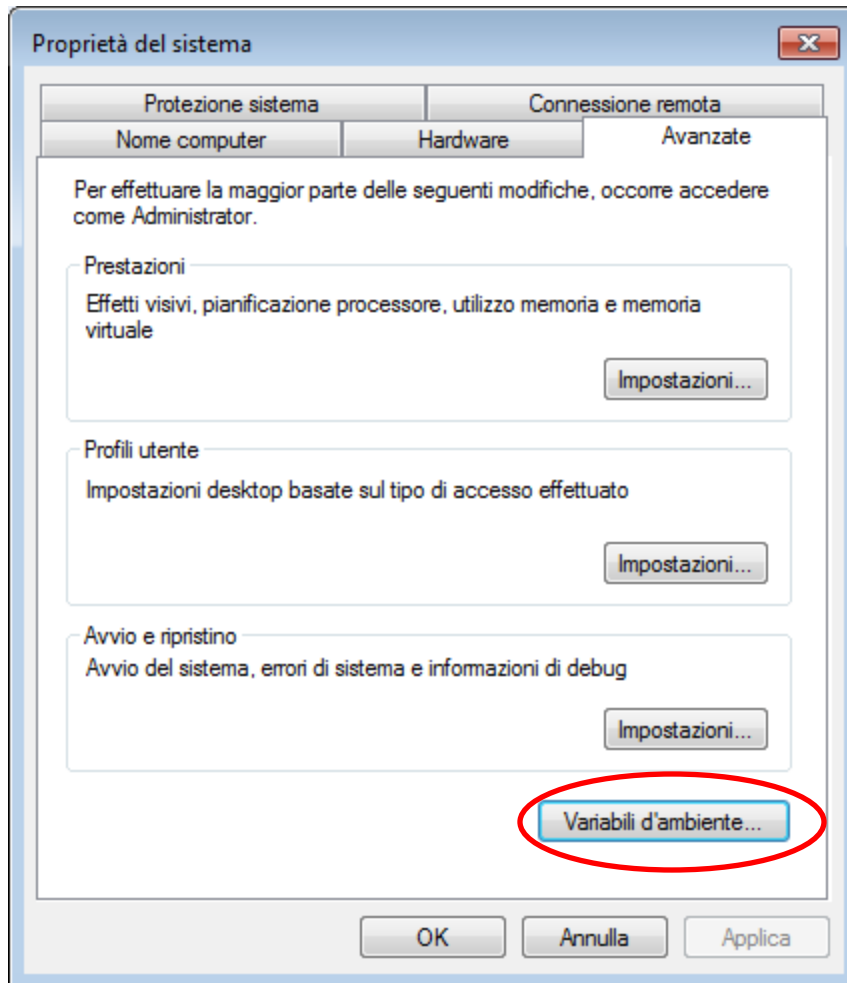
Uso decompilatore linea di comando

- Compilare il sorgente java:
 - **javac Nome.java**
- Traduzione da Java a (I)JVM:
 - **javap -c Nome > Nome.decomp**
- Aprire il file **Nome.decomp** con un editor di testo
- NB: se l'exec javap non e' riconosciuto come comando controllare se e' impostata la variabile d'ambiente.....

In Windows

- `PATH= ... ;<JAVA_HOME>/bin`
 - da aggiungere in fondo (NON sostituire!):
 - NB: senza spazi prima del punto e virgola!
 - Di solito è `java/jdk***/bin`
- Indispensabile per invocare i comandi java (compilatore **javac**, decompilatore **javap**, ...)
- Pannello di controllo- >
 - Sistema -> Avanzate -> Variabili d'ambiente

In Windows



Esercizio 1

Compilare codice Java in codice JVM

- 1) Scaricare dalla pagina del corso il file `Calcola2.java` (dentro la cartella di IJVM <https://informatica.i-learn.unito.it/mod/folder/view.php?id=105280>)
- 2) compilare con `javac Calcola2.java` (produce il `.class`)
- 3) decompilare per ottenere l'assemblativo corrispondente (JVM) con `javap -c Calcola2 > Calcola2.jasjava`
- 4) Aprire il file `Calcola2.jasjava` e osservare le differenze rispetto al linguaggio IJVM visto a lezione
- 5) disegnare lo stato dello stack dopo l'esecuzione dei blocchi di istruzione:
0-2, 3-5, 6-9, 10-13, 16-18, 25-26


```

public class Calcola2 {
public static void main (String[] args)
{int i, j=6, k=7;
    i= j+k;
    if (i != 7)
        k = 0;
    else
        j=j-1;
    i=0;
}
}

```

```

javac Calcola2.java
javap -c Calcola2
public static void main(String[]);
Code:

```

```

0: bipush      6
2: istore_2
3: bipush      7
5: istore_3
6: iload_2
7: iload_3
8: iadd
9: istore_1
10: iload_1
11: bipush      7
13: if_icmpeq    21
16: iconst_0
17: istore_3
18: goto         25
21: iload_2
22: iconst_1
23: isub
24: istore_2
25: iconst_0
26: istore_1

```

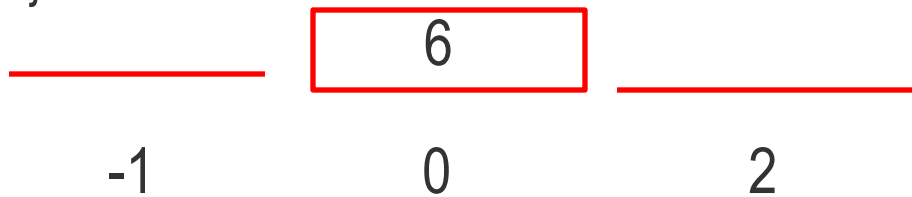
0: bipush	6
2: istore_2	
3: bipush	7
5: istore_3	
6: iload_2	
7: iload_3	
8: iadd	
9: istore_1	
10: iload_1	
11: bipush	7
13: if_icmpeq	21
16: iconst_0	
17: istore_3	
18: goto	25
21: iload_2	
22: iconst_1	
23: isub	
24: istore_2	
25: iconst_0	
26: istore_1	

Esa	Nome mnemonico	Significato
0x10	BIPUSH <i>byte</i>	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 <i>offset</i>	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 <i>offset</i>	Estrae una parola dalla cima dello stack ed effettua una diramazione se ha valore zero
0x9B	IFLT <i>offset</i>	Estrae una parola dalla cima dello stack ed effettua una diramazione se ha valore negativo
0x9F	IF_ICMPEQ <i>offset</i>	Estrae le due parole in cima allo stack ed effettua una diramazione se sono uguali
0x84	IINC <i>varnum const</i>	Aggiunge una costante a una variabile locale
0x15	ILOAD <i>varnum</i>	Scrive una variabile locale in cima allo stack
0xB6	INVOKEVIRTUAL <i>disp</i>	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 <i>varnum</i>	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 <i>index</i>	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

```

public class Calcola2 {
public static void main (String[] args)
{int i, j=6, k=7;
    i= j+k;
    if (i != 7)
        k = 0;
    else
        j=j-1;
    i=0;
}
}

```



Stack dopo ogni operazione

```

javac Calcola2.java
javap -c Calcola2
public static void main(String[]);
Code:

```

```

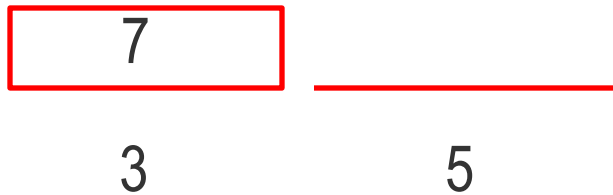
0: bipush      6
2: istore_2
3: bipush      7
5: istore_3
6: iload_2
7: iload_3
8: iadd
9: istore_1
10: iload_1
11: bipush      7
13: if_icmpeq    21
16: iconst_0
17: istore_3
18: goto         25
21: iload_2
22: iconst_1
23: isub
24: istore_2
25: iconst_0
26: istore_1

```

```

public class Calcola2 {
public static void main (String[] args)
{int i, j=6, k=7;
    i= j+k;
    if (i != 7)
        k = 0;
    else
        j=j-1;
    i=0;
}
}

```



Stack dopo ogni operazione

```

javac Calcola2.java
javap -c Calcola2
public static void main(String[]);
Code:
    0: bipush      6
    2: istore_2
    3: bipush      7
    5: istore_3
    6: iload_2
    7: iload_3
    8: iadd
    9: istore_1
   10: iload_1
   11: bipush      7
   13: if_icmpeq    21
   16: iconst_0
   17: istore_3
   18: goto        25
   21: iload_2
   22: iconst_1
   23: isub
   24: istore_2
   25: iconst_0
   26: istore_1

```

```
public class Calcola2 {
public static void main (String[] args)
```

```
{int i, j=6, k=7;
```

```
    i= j+k;
```

```
    if (i != 7)
```

```
        k = 0;
```

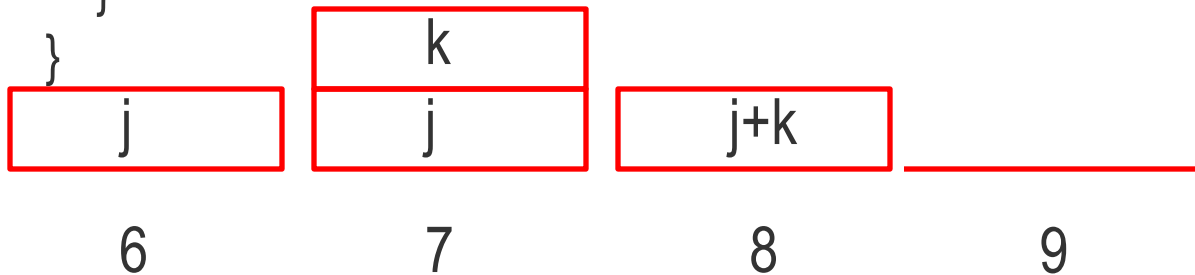
```
    else
```

```
        j=j-1;
```

```
        i=0;
```

```
    }
```

```
}
```



Stack dopo ogni operazione

```
javac Calcola2.java
```

```
javap -c Calcola2
```

```
public static void main(String[]);
```

```
Code:
```

```
0: bipush      6
```

```
2: istore_2
```

```
3: bipush      7
```

```
5: istore_3
```

```
6: iload_2
```

```
7: iload_3
```

```
8: iadd
```

```
9: istore_1
```

```
10: iload_1
```

```
11: bipush      7
```

```
13: if_icmpeq    21
```

```
16: iconst_0
```

```
17: istore_3
```

```
18: goto         25
```

```
21: iload_2
```

```
22: iconst_1
```

```
23: isub
```

```
24: istore_2
```

```
25: iconst_0
```

```
26: istore_1
```

```
public class Calcola2 {
public static void main (String[] args)
```

```
{int i, j=6, k=7;
```

```
    i= j+k;
```

```
    if (i != 7)
```

```
        k = 0;
```

```
    else
```

```
        j=j-1;
```

```
    i=0;
```

```
}
```

```
}
```

i

10

7
i

11

13

Stack dopo ogni operazione

javac Calcola2.java

javap -c Calcola2

public static void main(String[]);

Code:

0: bipush 6

2: istore_2

3: bipush 7

5: istore_3

6: iload_2

7: iload_3

8: iadd

9: istore_1

10: iload_1

11: bipush 7

13: if_icmpeq 21

16: iconst_0

17: istore_3

18: goto 25

21: iload_2

22: iconst_1

23: isub

24: istore_2

25: iconst_0

26: istore_1

```

public class Calcola2 {
public static void main (String[] args)
{int i, j=6, k=7;
  i= j+k;
  if (i != 7)
      k = 0;
  else
      j=j-1;
  i=0;
}
}

```

0

16

17

18

Stack dopo ogni operazione

```

javac Calcola2.java
javap -c Calcola2
public static void main(String[]);
Code:

```

```

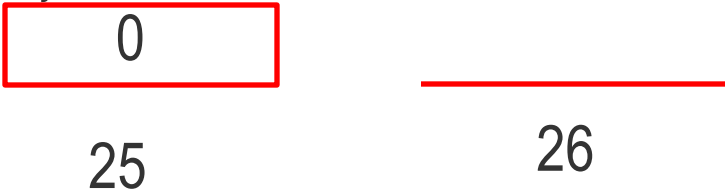
0: bipush      6
2: istore_2
3: bipush      7
5: istore_3
6: iload_2
7: iload_3
8: iadd
9: istore_1
10: iload_1
11: bipush      7
13: if_icmpeq    21
16: iconst_0
17: istore_3
18: goto         25
21: iload_2
22: iconst_1
23: isub
24: istore_2
25: iconst_0
26: istore_1

```

```

public class Calcola2 {
public static void main (String[] args)
    {int i, j=6, k=7;
        i= j+k;
        if (i != 7)
            k = 0;
        else
            j=j-1;
        i=0;
    }
}

```



```

javac Calcola2.java
javap -c Calcola2
public static void main(String[]);
Code:

```

```

0: bipush      6
2: istore_2
3: bipush      7
5: istore_3
6: iload_2
7: iload_3
8: iadd
9: istore_1
10: iload_1
11: bipush      7
13: if_icmpeq    21
16: iconst_0
17: istore_3
18: goto         25
21: iload_2
22: iconst_1
23: isub
24: istore_2
25: iconst_0
26: istore_1

```

Stack dopo ogni operazione

Differenze tra IJVM e JVM (I)

- **bipush 0** <-> **iconst_0**:
- **iconst_0**: Inserisce la costante intera 0 in cima allo stack.
- Esistono diverse alternative per questa istruzione:

Istruzione	Costante associata
iconst_m1	-1
iconst_0	0
iconst_1	1
iconst_2	2
iconst_3	3
iconst_4	4
iconst_5	5

- Per costanti fuori [-1,5] viene usata la bipush

Differenze tra IJVM e JVM (II)

- Varianti delle istruzioni di salto condizionato:

Istruzione	Costante associata
IFEQ L	Salto se il valore sullo stack e' 0
IFNE L	Salto se il valore sullo stack e' diverso da 0
IFLT L	Salto se il valore sullo stack e' minore di 0
IFLE L	Salto se il valore sullo stack e' minore o uguale a 0
IFGT L	Salto se il valore sullo stack e' maggiore di zero
IFGE L	Salto se il valore sullo stack e' maggiore o uguale a zero
IF_CMPEQ L	Legge le due parole in cima allo stack (2 push) ed effettua una diramazione se sono uguali (offset L)
IF_CMPNE L	Legge le due parole in cima allo stack (2 push) ed effettua una diramazione se sono diverse (offset L)
....

- Modello di memoria IJVM
- Traduzione programmi Java in JVM (con i programmi della suite Java)
- Traduzione programmi Java in IJVM (per l'emulatore Mic1MVV)
 - Semplici programmi in cui ci sono operazioni matematiche su valori
 - Uso della stack

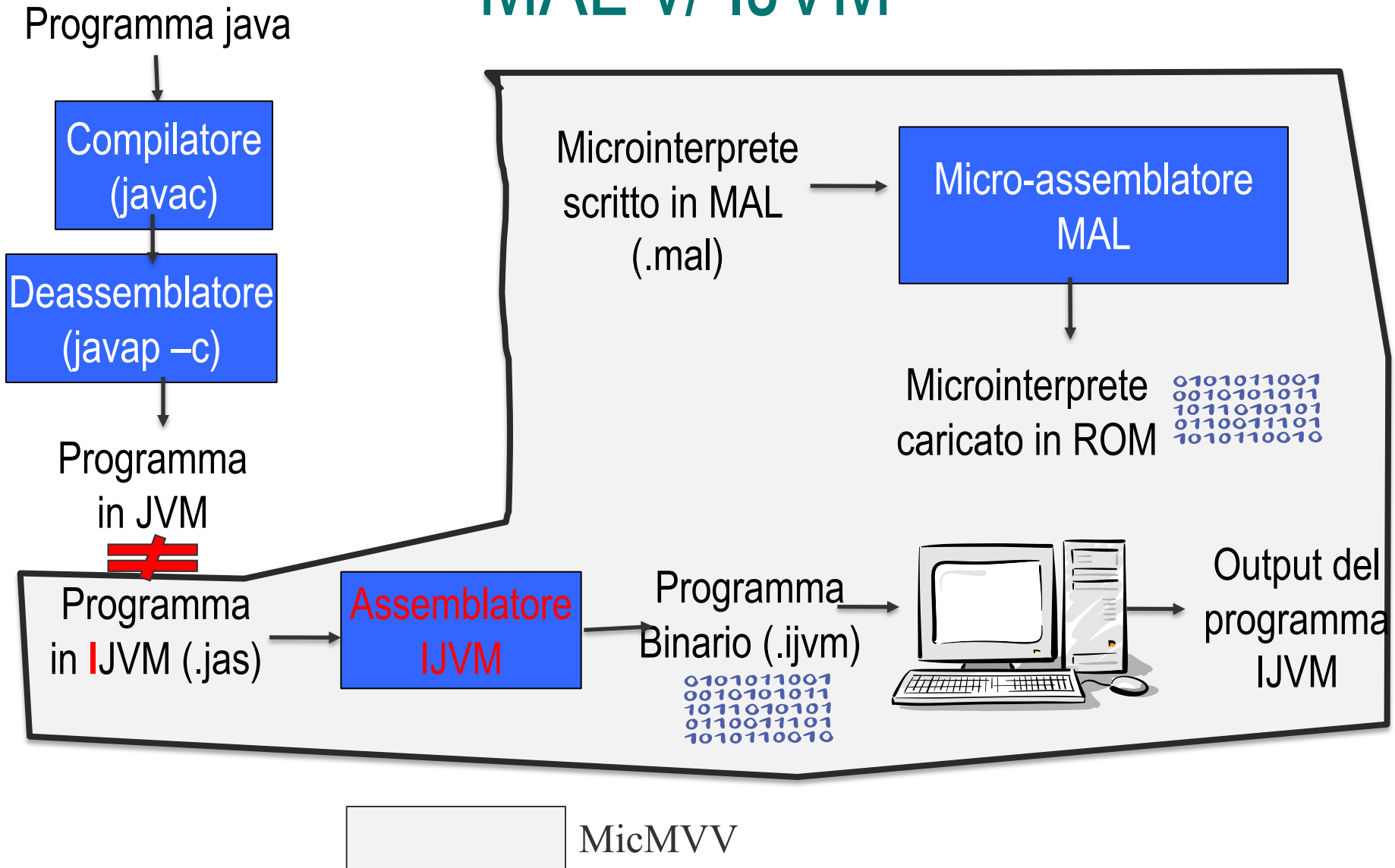
Simulatore Mic1MMV

- Che cos'è ?
- Un simulatore interattivo dell'architettura MIC-1 vista a lezione (Tanenbaum - cap.4)
- Si può utilizzare a diverse velocità (cioè livelli di granularità)
- È implementato in Java

Simulatore Mic1MMV – Funzionalità

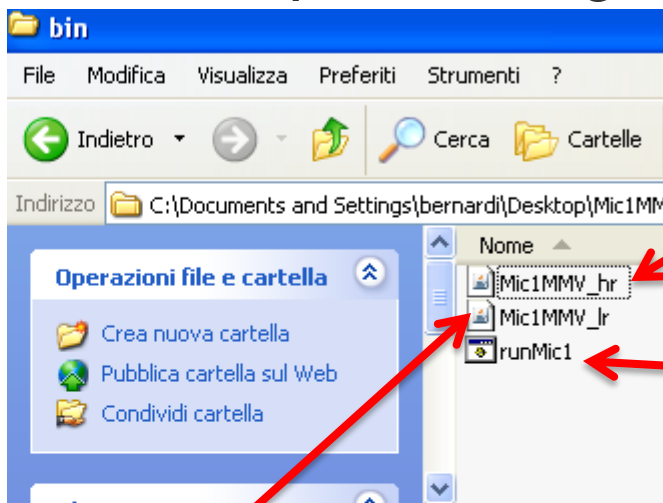
- Interpreta le istruzioni IJVM
- Assembla programmi scritti in IJVM (file .jas compilati in file .ijvm)
- Assembla programmi scritti in MAL (file .mal compilati in file .mic)

MAL v/ IJVM



Simulatore Mic1MMV – Installazione

- Scaricare il file Mic1MMV.zip da Moodle
- Decomprimerlo, gli eseguibili in: ./Mic1MMV/bin

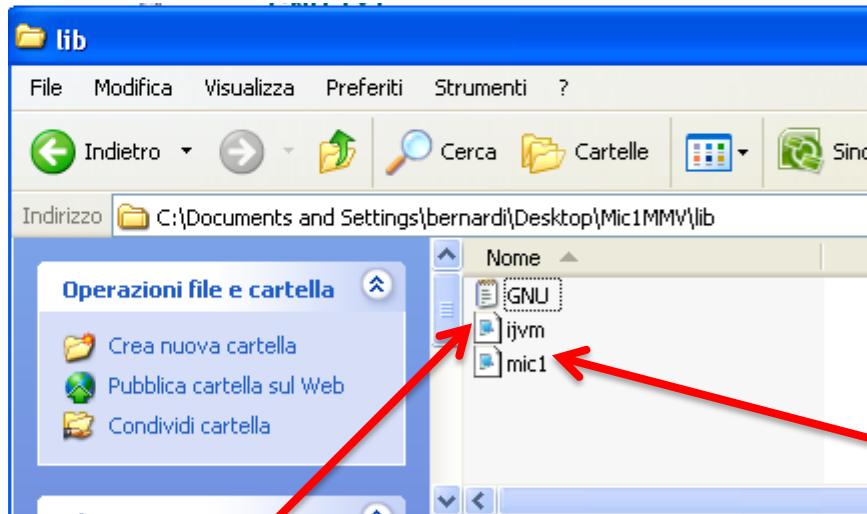


Mic1MMV_hr.jar schermi ad alta risoluzione (da 1280 x 960)

In alternativa: **runMic1.bat**
batch file da editare e usare per lanciare il simulatore.

Mic1MMV_lr.jar schermi a bassa risoluzione

Simulatore Mic1MMV – Lib

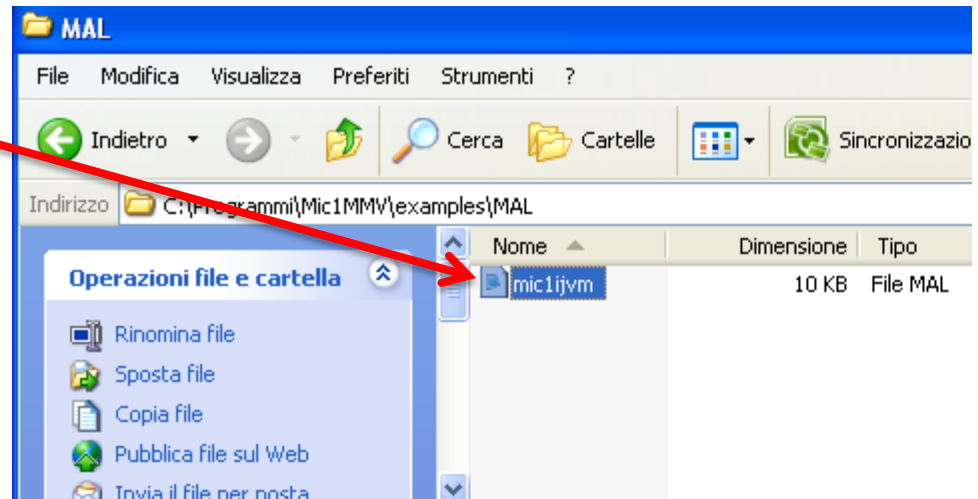
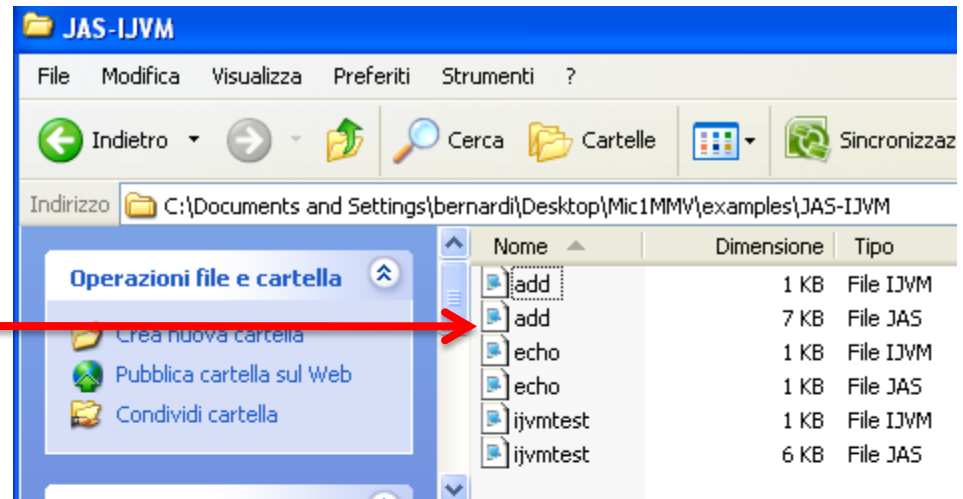


mic1.properties file per impostare le proprietà di default

ijvm.conf File di configurazione per l'assemblatore ijvmasm. Descrive il linguaggio IJVM

Simulatore Mic1MMV – Esempi

- **JAS-IJVM**: esempi programmi IJVM in codice sorgente (.jas) e assemblato (.ijvm)
- **MAL**: contiene il microinterprete di IJVM scritto in MAL (**mic1ijvm.mal**).



Simulatore Mic1MMV – Altro

- /doc: documentazione
- /src: sorgente simulatore

Datapath

Area
Metodi

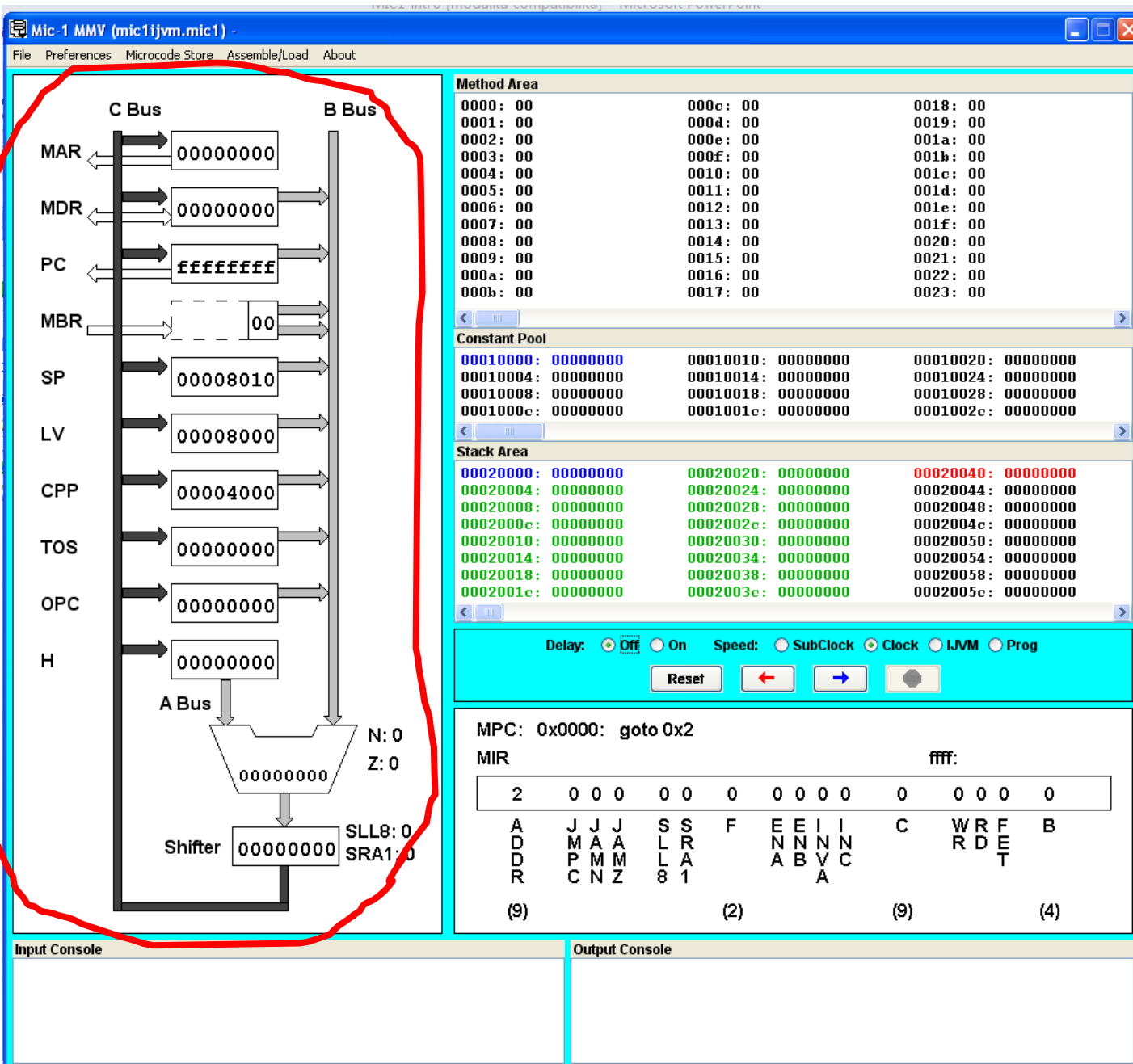
Pool
costante

Stack

Console
comandi

MIR

Console
output



Console
input