Il Livello del linguaggio Assemblatore

con riferimento a IJVM e mic-1

- **Il linguaggio** (IJVM Assembly language specification)
- **Il processo di assemblaggio** (Tanenbaum 7.3)
- **Collegamento e caricamento** (Tanenbaum 7.4)

Il linguaggio Assemblatore

E' il **linguaggio *simbolico*** che consente di programmare un calcolatore utilizzando le istruzioni del linguaggio macchina.

Un programma scritto in ASSEMBLER per poter essere eseguito deve essere **tradotto in linguaggio macchina binario**, in modo tale da tradurre i codici mnemonici delle istruzioni in codici operativi, sostituire tutti i riferimenti simbolici degli indirizzi con la loro forma binaria, e riservare lo spazio di memoria per le variabili.

L'operazione di traduzione viene eseguita dall'ASSEMBLATORE.

Se nel codice sorgente sono presenti riferimenti simbolici definiti in moduli esterni a quello assemblato è necessario anche il LINKER (collegatore).

Assembler IJVM (i)

Istruzioni:

- rappresentazione simbolica di tutte e sole le istruzioni ISA (**`ILOAD, ISTORE, ….`**). Un'istruzione per ogni riga di codice sorgente Assembler

Riferimenti a memoria (nel corpo delle istruzioni):

- **variabili e parametri**: vengono espressi tramite nomi simbolici (***`nome_var, nome_par…`***)
- **destinazioni di salto**:vengono espresse tramite nomi simbolici (**`ciclo, fine, else …`**)

Label (premesse ad una istruzione):

- identificano in modo simbolico le istruzioni destinazioni di un salto. In Assembler IJVM sono terminate dal simbolo : (**`ciclo:, fine:, else:..`**).

Assembler IJVM (ii)

Direttive all'assemblatore:
Non sono istruzioni Assembler ma vengono usate dall'assemblatore per generare la traduzione in linguaggio macchina

1. Direttiva per definire costanti simboliche
Sintassi:

```
.constant
constant1 value1
constant2 value2
.end-constant
```

Le costanti sono dichiarate in una sezione "globale" prima del programma principale e degli eventuali metodi.
Il valore può essere specificato in esadecimale, ottale o decimale. Il nome simbolico può essere utilizzato nelle istruzioni che prevedono come operando una costante (**`LDC_W constant_name`**). I valori delle costanti verrano caricati nella Constant Pool.

Assembler IJVM (iii)

2. Direttiva per dichiarare variabili locali (all'interno del programma principale o di un sottoprogramma). Sintassi:

```
.var
var1
var2
.end-var
```

Le variabili locali vengono allocate sulla pila nell'area di attivazione. In IJVM, ciascuna variabile occupa una parola di memoria**. L'ordine di enunciazione è l'ordine di allocazione**.

Assembler IJVM (vi)

3. Direttiva per definire il programma principale
Sintassi:

```
.main
-- variable declaration --
-- program contents --
.end-main
```

Il programma principale deve essere sempre presente e dichiarato prima di eventuali altri sottoprogrammi.

Assembler IJVM (v)

4. Direttiva per definire un sottoprogramma (metodo)
Sintassi:

```
.method method_name(param1, param2,...)
-- variable declaration --
-- method contents --
.end-method
```

L'indirizzo iniziale (del 1° byte) del sottoprogramma viene aggiunto nell'area della Constant Pool e può quindi essere referenziato tramite il nome del metodo (**`INVOKEVIRTUAL method_name`**).
I parametri del sottoprogramma vengono dichiarati tra parentesi separati da virgole.

Assembler IJVM (vi)

Per consentire la corretta chiamata a sottoprogramma e l'eventuale uso del valore restituito è necessario che

- il chiamante preveda nel suo codice nell'ordine le istruzioni di:
  - caricamento in pila di **`OBJREF`** (ad esempio tramite una **`LDC_W`** o **`BIPUSH`** con valore qualsiasi). In IJVM il valore caricato viene ignorato (e sovrascritto) durante la chiamata del sottoprogramma stesso, ma deve essere riservato lo spazio relativo in memoria.
  - caricamento in pila dei parametri del sottoprogramma nell'ordine indicato nella dichiarazione del sottoprogramma stesso **`(ILOAD param1, ILOAD param2 …)`**
  - chiamata del sottoprogramma tramite **`INVOKEVIRTUAL method_name`**
- se il chiamante fa uso del valore restituito, questo deve trovarsi in cima allo stack del chiamante al termine del sottoprogramma.

L'insieme delle 2 direttive

```
.method method_name(param1, param2,...)
.var
…….
.end-var
```

viene utilizzata dall'assemblatore per "costruire" il ***descrittore del sottoprogramma*** previsto per la corretta esecuzione della **`INVOKEVIRTUAL`**.

Il processo di Assemblaggio (i)

E' un procedimento sequenziale che esamina, riga per riga, il codice sorgente Assembler.

Traduce i codici mnemonici delle istruzioni nei corrispondenti codici binari

Traduce i riferimenti simbolici (variabili, etichette di salto, parametri) nei corrispondenti indirizzi numerici.

Etichette di salto: generano il problema dei *riferimenti in avanti* che viene risolto tramite una **traduzione in 2 passi**. L'operazione di traduzione fa uso di opportune tabelle

Il processo di Assemblaggio (ii)

- **Primo passo dell'Assemblatore**

  Costruzione della **Tabella dei Simboli**: la tabella contiene i riferimenti simbolici presenti nel modulo da tradurre e, al termine del primo passo, conterrà gli indirizzi numerici di tutti i simboli, tranne al più quelli esterni al modulo in esame che sono tipicamente relativi a sottoprogrammi (o globali a tutti i moduli)

  Etichette di salto: il valore dell'indirizzo numerico generato durante la traduzione rappresenta lo spiazzamento (positivo e negativo) tra l'indirizzo dell'istruzione di salto e l'indirizzo dell'istruzione destinazione di salto.

  Per calcolare tale spiazzamento, durante il primo passo, l'assemblatore associa ad ogni istruzione la sua lunghezza in byte e l'indirizzo (d'inizio) dell'istruzione stessa. Gli indirizzi d'inizio di ogni modulo vengono generati a partire dall'indirizzo 0.

Il processo di Assemblaggio (ii)

- **Secondo passo dell'Assemblatore**

  Il programma viene scandito nuovamente e, tramite la Tabella dei Simboli, si sostituiscono i riferimenti simbolici risolti e si marcano le istruzioni che contengono riferimenti simbolici non risolti.

  Viene generato il programma in codice oggetto

Il processo di Collegamento

La fase di assemblaggio può generare più di un modulo oggetto (p. es.uno per ogni sottoprogramma) oppure il programma stesso fa uso di moduli di libreria.

Il **linker** ha il compito di generare dai diversi moduli oggetto un **unico programma binario eseguibile** (in formato rilocabile). Si può anche dire che il linker ha il compito di generare un unico spazio di indirizzamento per tutto il programma.

In base alla lunghezza di ciascun modulo tradotto, è possibile calcolare l'indirizzo d'inizio di ogni modulo nel programma eseguibile. Tali indirizzi vengono sostituiti ai riferimenti non risolti nelle chiamate a sottoprogramma dell'intero programma.

Anche il linker può far uso di opportune tabelle per il processo di collegamento.

Programma in linguaggio C

```
# define   MAX 10
# define   MIN 3

main ( )

{int I, J, K;

  I=1;
  J=3;

  if (I < MIN)
    K = J + PROVA(I);
  else
    K = PROVA(MAX+J);
} /* end main */


int PROVA(int P)

{int C;

  C = P+2;

  return C;
} /* end prova */
```


Programma equivalente assembler IJVM(1)

```
.constant
MAX 10
MIN 3
.end-constant

.main
.var
I
J
K
.end-var
     BIPUSH    1
     ISTORE    I
     BIPUSH    3
     ISTORE    J
     ILOAD     I
     LDC_W     MIN
     ISUB
     IFLT      THEN
     BIPUSH    5   //ramo else
     LDC_W     MAX
     ILOAD     J
     IADD          //val. param. in cima allo stack
     INVOKEVIRTUAL    PROVA
     ISTORE    K
     GOTO      FINE
THEN:
     ILOAD     J   //ramo then
     BIPUSH    5
     ILOAD     I   //val. param.in cima allo stack
     INVOKEVIRTUAL    PROVA
     IADD
     ISTORE    K
FINE:
     NOP
.end-main
```

questa istruzione ha il compito di caricare in OBJREF il valore 5

Programma equivalente assembler IJVM(2)

```
.method PROVA (P)
.var
C
.end-var
     ILOAD     P
     BIPUSH    2
     IADD
     ISTORE    C
     ILOAD     C
     IRETURN
.end-method
```


PROCESSO DI ASSEMBLAGGIO (1)

1. Sezione dei riferimenti simbolici "globali"

Analizza la porzione di codice asssembler sottoriportata e genera una tabella che contiene le costanti globali e i nomi simbolici dei sottoprogrammi

.constant
MAX 10
MIN 3
.end-constant

La tabella fa riferimento alla Constant Pool e quindi il valore dell'indice è espresso rispetto a **CPP**

simbolo	valore nella CP	offset (indice)
MAX	10	0
MIN	3	1
PROVA		2

NOTA:

(a) Il nome simbolico del sottoprogramma, e il relativo valore dell'indice, viene inserito durante l'assemblaggio del sottoprogramma stesso.

(b) Il "valore nella CP" relativo al sottoprogramma è l'indirizzo d'inizio del suo codice e verrà determinato in fase di collegamento (questa tabella viene quindi utilizzata anche dal linker).

PROCESSO DI ASSEMBLAGGIO (2)

2. Modulo main

a) Calcolo della lunghezza delle istruzioni e degli indirizzi d'inizio delle istruzioni

ind	# byte istr		
			.main
			.var
			I
			J
			K
0	4		.end-var
4	2		BIPUSH 1
6	2		ISTORE I
8	2		BIPUSH 3
10	2		ISTORE J
12	2		ILOAD I
14	3		LDC_W MIN
17	1		ISUB
18	3		IFLT THEN
21	2		BIPUSH 5
23	3		LDC_W MAX
26	2		ILOAD J
28	1		IADD
29	3		INVOKEVIRTUAL PROVA
32	2		ISTORE K
34	3		GOTO FINE
		THEN:	
37	2		ILOAD J
39	2		BIPUSH 5
41	2		ILOAD I
43	3		INVOKEVIRTUAL PROVA
46	1		IADD
47	2		ISTORE K
		FINE:	
49	1		NOP
			.end-main

2. Modulo main

b) Tabella dei simboli: tabella generata dalla prima passata dell'assemblatore per main

simbolo	offset (indice)
I	1
J	2
K	3
MIN	non risolto
THEN	37
MAX	non risolto
PROVA	non risolto
FINE	49

NOTE:
- l'indice delle varibili locali è relativo a LV
- l'indice 0 (rispetto a LV) si suppone usato per objref di main
- gli indici di I, J, K sono determinati dall'esame di **.var**
- l'indice per destinazione di salto è relativo a PC e il suo valore rappresenta l'indirizzo dell'istruzione **destinazione di salto**

2. Modulo main

c) Linguaggio macchina simbolico di main
generato dalla seconda passata dell'assemblatore

ind	# byte istr		
0	4		***Descrittore di MAIN***
4	2		BIPUSH 1
6	2		ISTORE 1
8	2		BIPUSH 3
10	2		ISTORE 2
12	2		ILOAD 1
14	3	*	LDC_W MIN
17	1		ISUB
18	3		IFLT +19
21	2		BIPUSH 5
23	3	*	LDC_W MAX
26	2		ILOAD 2
28	1		IADD
29	3	*	INVOKEVIRTUAL PROVA
32	2		ISTORE 3
34	3		GOTO +15
37	2		ILOAD 2
39	2		BIPUSH 5
41	2		ILOAD 1
43	3	*	INVOKEVIRTUAL PROVA
46	1		IADD
47	2		ISTORE 3
49	1		NOP

Per leggibilità si è mantenuto lo mnemonico assembler, si sono sotituiti i simboli, si sono marcate con * le istruzioni che hanno ancora riferimenti non risolti

PROCESSO DI ASSEMBLAGGIO (3)

3. Modulo prova

a) Calcolo della lunghezza delle istruzioni e degli indirizzi d'inizio delle istruzioni

ind	# byte istr		
			.method PROVA (P)
			.var
			C
0	4		.end-var
4	2		ILOAD P
6	2		BIPUSH 2
8	1		IADD
9	2		ISTORE C
11	2		ILOAD C
13	1		IRETURN
			.end-method

b) Tabella dei simboli: tabella generata dalla prima passata dell'assemblatore per prova

simbolo	offset (indice)
P	1
C	2

NOTE:
- l'indice dei parametri e delle varibili locali è relativo a LV
- l'indice 0 (rispetto a LV) è usato per objref di prova
- l'indice dei parametri è determinato dalla scansione ordinata dei parametri tra parentesi
- l'indice di c è determinato dall'esame di **.var**

3. Modulo prova

c) Linguaggio macchina simbolico di prova

generato dalla seconda passata dell'assemblatore

ind	# byte istr		
0	4		***Descrittore di PROVA***
4	2		ILOAD 1
6	2		BIPUSH 2
8	1		IADD
9	2		ISTORE 2
11	2		ILOAD 2
13	1		IRETURN

PROCESSO DI COLLEGAMENTO (1)

Calcolo degli indirizzi per i moduli collegati

ind	# byte istr		
0	4		***Descrittore di MAIN***
4	2		BIPUSH 1
6	2		ISTORE 1
8	2		BIPUSH 3
10	2		ISTORE 2
12	2		ILOAD 1
14	3	*	LDC_W MIN
17	1		ISUB
18	3		IFLT +19
21	2		BIPUSH 5
23	3	*	LDC_W MAX
26	2		ILOAD 2
28	1		IADD
29	3	*	INVOKEVIRTUAL PROVA
32	2		ISTORE 3
34	3		GOTO +15
37	2		ILOAD 2
39	2		BIPUSH 5
41	2		ILOAD 1
43	3	*	INVOKEVIRTUAL PROVA
46	1		IADD
47	2		ISTORE 3
49	1		NOP
50	4		***Descrittore di PROVA***
54	2		ILOAD 1
56	2		BIPUSH 2
58	1		IADD
59	2		ISTORE 2
61	2		ILOAD 2
63	1		IRETURN

PROCESSO DI COLLEGAMENTO (2)

TABELLA DEL COLLEGATORE aggiornata

simbolo	valore nella CP	offset (indice)
MAX	10	0
MIN	3	1
PROVA	50	2

Codice eseguibile prodotto dal collegatore dopo aver utilizzato la tabella

0	***Descrittore di MAIN***
4	BIPUSH 1
6	ISTORE 1
8	BIPUSH 3
10	ISTORE 2
12	ILOAD 1
14	LDC_W 1
17	ISUB
18	IFLT +19
21	BIPUSH 5
23	LDC_W 0
26	ILOAD 2
28	IADD
29	INVOKEVIRTUAL 2
32	ISTORE 3
34	GOTO +15
37	ILOAD 2
39	BIPUSH 5
41	ILOAD 1
43	INVOKEVIRTUAL 2
46	IADD
47	ISTORE 3
49	NOP
50	***Descrittore di PROVA***
54	ILOAD 1
56	BIPUSH 2
58	IADD
59	ISTORE 2
61	ILOAD 2
63	IRETURN