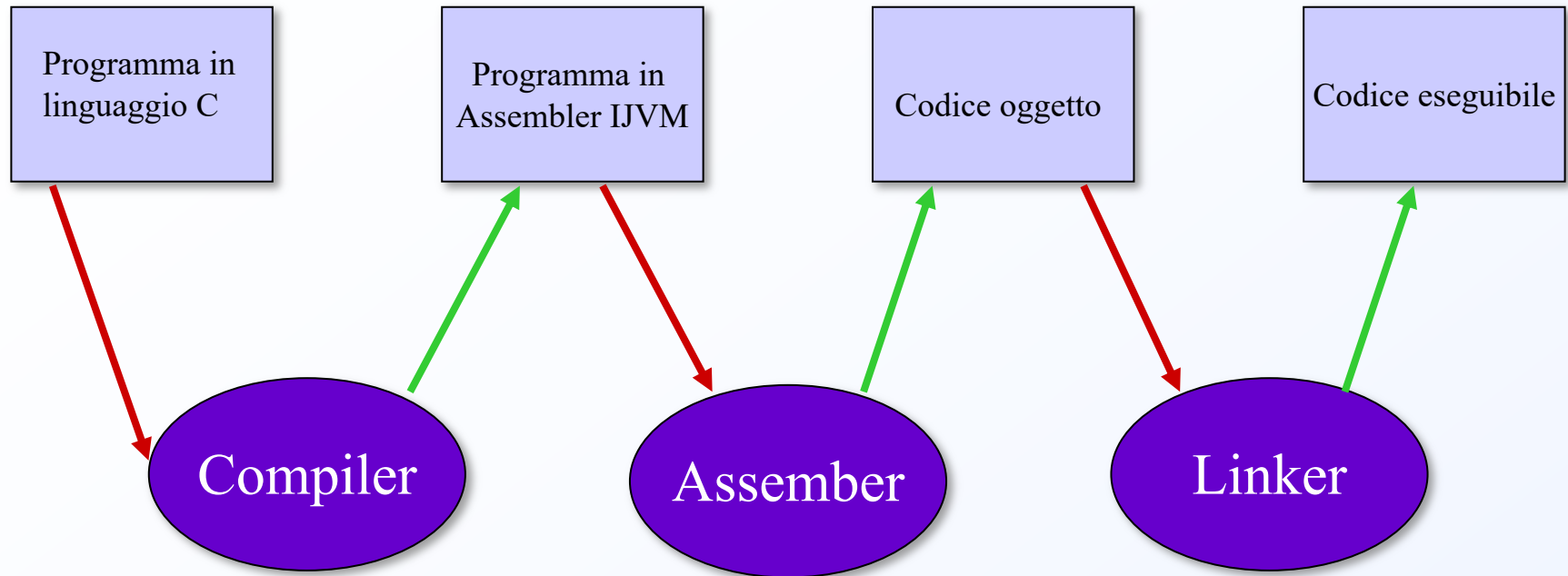


Architettura degli Elaboratori a.a. 2018/19

Da un programma scritto in linguaggio C al suo codice eseguibile passando attraverso l'assembler IJVM

Dal codice sorgente al codice eseguibile



Programma in linguaggio C

```
#define MAX 10    /* Definizione di costanti globali */
#define MIN 3

main()            /* Funzione principale */
{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)  /* Sottoprogramma invocato dal main */
{int C;
  C=P+2;
  return C;
} /*end prova*/
```

COMPILAZIONE

- Poiché nel programma in linguaggio C, preso in esame, sono presenti due moduli distinti: il **main** e il sottoprogramma **PROVA**, è necessario distinguere la compilazione (traduzione in linguaggio Assemblativo IJVM) in due momenti diversi, ciascuno per ogni modulo presente:
 - Compilazione del **main**
 - Compilazione di **PROVA**

COMPILAZIONE MAIN

```
#define MAX 10  
#define MIN 3
```


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

```
main()          */  
{int I, J, K;  
  I=1;  
  J=3;  
  if (I < MIN)  
    K=J+PROVA(I);  
  else  
    K=PROVA(MAX+J);  
} /*end main*/
```

```
.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  
    LDC_W MAX  
    ILOAD J  
    IADD  
    INVOKEVIRTUAL PROVA  
    ISTORE K  
    GOTO FINE  
THEN: ILOAD J  
    BIPUSH 5  
    ILOAD I  
    INVOKEVIRTUAL PROVA  
    IADD  
    ISTORE K  
FINE: NOP  
.end-main
```

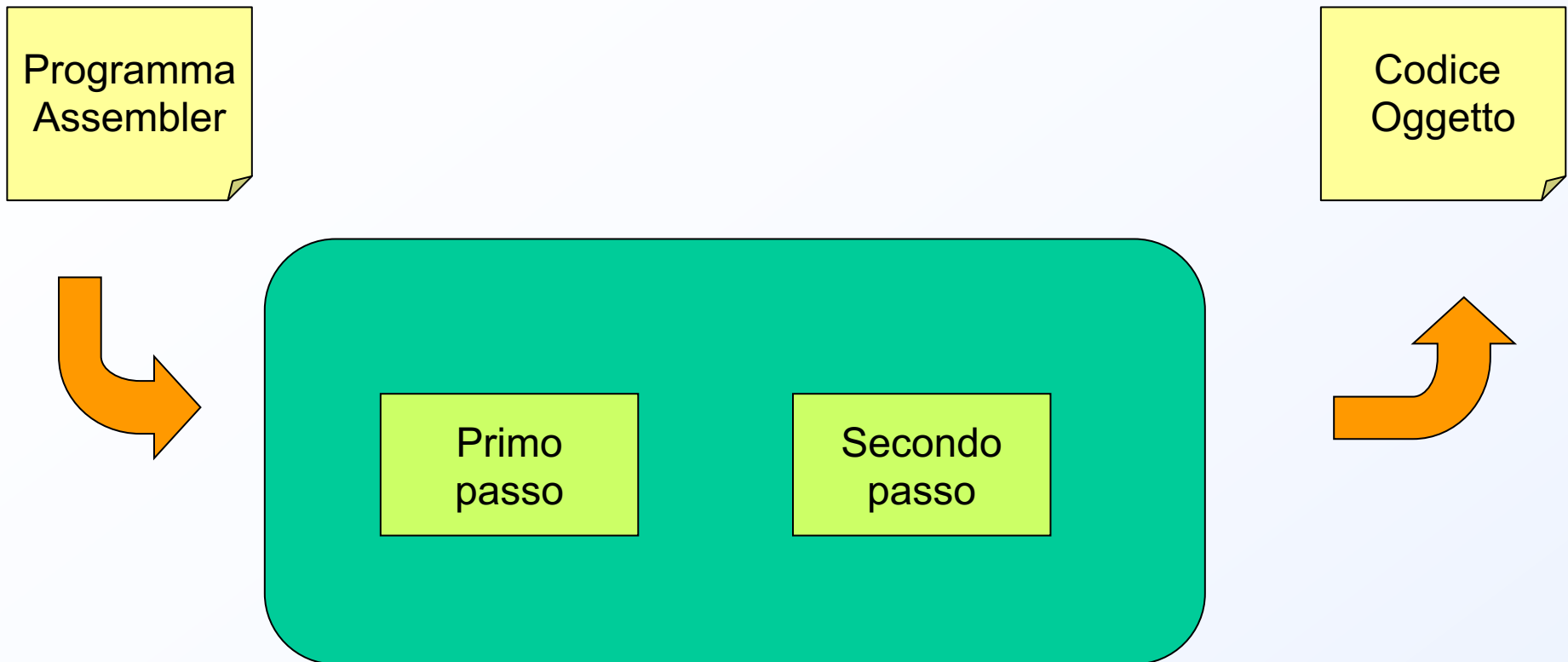
COMPILAZIONE PROVA

```
int PROVA(int P)
{int C;
  C=P+2;
  return C;
} /*end prova*/
```



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

Processo di assemblaggio



PRIMO PASSO

- Durante il primo l'assemblatore costruisce due tabelle: la **Tabella delle Costanti**, contenente le costanti globali e i nomi simbolici dei sottoprogrammi e la **Tabella dei Simboli**, contenente i valori dei simboli.
- Nell'assegnare un valore ad un simbolo nel campo etichetta di una istruzione, l'assemblatore deve sapere quale indirizzo avrà quella istruzione in fase di esecuzione. Per tener traccia di tale indirizzo, l'assemblatore mantiene una variabile durante l'assemblaggio, nota come *contatore della locazione dell'istruzione* o **ILC**.
- Per determinare la lunghezza di una istruzione l'assemblatore usa la **Tabella degli Opcode**, che contiene almeno un elemento per ogni codice operativo simbolico (mnemonico) del linguaggio Assemblativo IJVM.

PRIMO PASSO MAIN

Tabella delle costanti

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

L'offset indica la
distanza dalla base
della Constant Pool

Simbolo	Valore nella CP	Offset
MAX	10	0
MIN	3	1

PRIMO PASSO MAIN

Tabella dei simboli

```
.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
    LDC_W MAX
    ILOAD J
    IADD
    INVOKEVIRTUAL PROVA
    ISTORE K
    GOTO FINE
THEN:
    ILOAD J
    BIPUSH 5
    ILOAD I
    INVOKEVIRTUAL PROVA
    IADD
    ISTORE K
FINE:
    NOP
.end-main
```

Simbolo	Offset
I	1
j	2
k	3
MIN	non risolto
THEN	37
MAX	non risolto
PROVA	non risolto
FINE	49

PRIMO PASSO PROVA

Tabella delle costanti

```
.method PROVA(P)  
.var  
C  
.end-var
```

Il valore nella CP
verrà determinato in
fase di linking

Simbolo	Valore nella CP	Offset
MAX	10	0
MIN	3	1
PROVA		2

PRIMO PASSO PROVA

Tabella dei simboli

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

Simbolo	Offset
P	1
C	2

SECONDO PASSO

- × La funzione del secondo passo è di generare il *codice oggetto* del programma sorgente.
- × Durante questo passo, 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*; inoltre, le istruzioni vengono *tradotte in codice oggetto*, viene rilevata la presenza di *eventuali errori* (nel sorgente) e vengono *emesse le informazioni necessarie al Linker* per collegare i moduli assemblati in momenti diversi.

×

SECONDO PASSO

MAIN

```

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
  LDC_W MAX
  ILOAD J
  IADD
  INVOKEVIRTUAL PROVA
  ISTORE K
  GOTO FINE
THEN:ILOAD J
  BIPUSH 5
  ILOAD I
  INVOKEVIRTUAL PROVA
  IADD
  ISTORE K
FINE:NOP
.end-main

```

Ind. Primo byte	# Byte		Mnemonico Assembler	Codifica Esad.
0	4		descrittore di main	0x00 0x01 0x00 0x03
4	2		BIPUSH 1	0x10 0x01
6	2		ISTORE 1	0x36 0x01
8	2		BIPUSH 3	0x10 0x03
10	2		ISTORE 2	0x36 0x02
12	2		ILOAD 1	0x15 0x01
14	3	*	LDC_W MIN	0x13 0x00 0x00
17	1		ISUB	0x64
18	3		IFLT +19	0x9B 0x00 0x13
21	2		BIPUSH 5	0x10 0x05
23	3	*	LDC_W MAX	0x13 0x00 0x00
26	2		ILOAD 2	0x15 0x02
28	1		IADD	0x60
29	3	*	INVOKEVIRTUAL PROVA	0xB6 0x00 0x00
32	2		ISTORE 3	0x36 0x03
34	3		GOTO +15	0xA7 0x00 0x0F
37	2		ILOAD 2	0x15 0x02
39	2		BIPUSH 5	0x10 0x05
41	2		ILOAD 1	0x15 0x01
43	3	*	INVOKEVIRTUAL PROVA	0xB6 0x00 0x00
46	1		IADD	0x60
47	2		ISTORE 3	0x36 0x03
49	1		NOP	0x00

SECONDO PASSO PROVA

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

Ind. Primo byte	# Byte		Mnemonico Assembler	Codifica Esad.
0	4		descrittore di PROVA	0x00 0x02 0x00 0x01
4	2		ILOAD 1	0x15 0x01
6	2		BIPUSH 2	0x10 0x021
8	1		IADD	0X60
9	2		ISTORE 2	0x36 0x02
11	2		ILOAD 2	0x15 0x02
13	1		IRETURN	0XAC

Il Processo di Collegamento:

- Il Linker deve fondere i due moduli oggetto separati in un unico spazio di indirizzamento
- A tale scopo il linker costruisce una tabella contenente per ciascun modulo:
 - ♦ nome
 - ♦ lunghezza
 - ♦ indirizzo di inizio

Il Processo di Collegamento:

- × Il Linker deve fondere i due moduli oggetto separati in un unico spazio di indirizzamento
- × A tale scopo il linker costruisce una tabella contenente per ciascun modulo:
 - ♦ nome
 - ♦ lunghezza
 - ♦ indirizzo di inizio

Il Processo di Collegamento

Ind. Primo byte	# Byte		Mnemonico Assembler	Codifica Esad.
0	4		descrittore di main	0x00 0x01 0x00 0x03
49	1		NOP	0x00

Modulo	Lunghezza	Ind. inizio
main	50	0
PROVA	14	50

Ind. Primo byte	# Byte		Mnemonico Assembler	Codifica Esad.
0	4		descrittore di PROVA	0x00 0x02 0x00 0x01
13	1		IRETURN	0x0AC

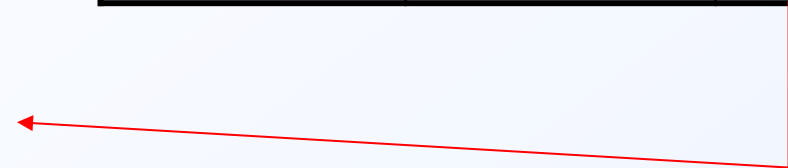
Il Processo di Collegamento

Ind. Primo byte	# Byte		Mnemonico Assembler	Codifica Esad.
0	4		descrittore di main	0x00 0x01 0x00 0x03
14	3	*	LDC_W MIN	0x13 0x00 0x00
23	3	*	LDC_W MAX	0x13 0x00 0x00
26	2		ILOAD 2	0x15 0x02
28	1		IADD	0x60
29	3	*	INVOKEVIRTUAL PROVA	0xB6 0x00 0x00
43	3	*	INVOKEVIRTUAL PROVA	0xB6 0x00 0x00
49	1		NOP	0x00

50	4		descrittore di PROVA	0x00 0x02 0x00 0x01
54	2		ILOAD 1	0x15 0x01
56	2		BIPUSH 2	0x10 0x021
58	1		IADD	0X60
59	2		ISTORE 2	0x36 0x02
61	2		ILOAD 2	0x15 0x02
63	1		IRETURN	0XAC

Basandosi sulla tabella costruita il linker si crea un unico spazio di indirizzamento

Modulo	Lunghezza	Ind. inizio
main	50	0
PROVA	14	50



Il Processo di Collegamento

- Determinato l'indirizzo di inizio del sottoprogramma PROVA è possibile aggiornare la Tabella delle Costanti inserendo il valore relativo a PROVA
- L'ultima operazione consiste nello scandire il modulo oggetto per sostituire ai riferimenti non risolti i relativi valori di offset

Il Processo di Collegamento

- Determinato l'indirizzo di inizio del sottoprogramma PROVA è possibile aggiornare la Tabella delle Costanti inserendo il valore relativo a PROVA

Simbolo	Valore nella CP	Offset
MAX	10	0
MIN	3	1
PROVA	50	2

Il Processo di Collegamento

L' ultima operazione consiste nello scandire il modulo oggetto per sostituire ai riferimenti non risolti i relativi valori di offset

Simbolo	Valore nella CP	Offset
MAX	10	0
MIN	3	1
PROVA	50	2

Ind. Primo byte	# Byte		Mnemonico Assembler	Codifica Esad.
0	4		descrittore di main	0x00 0x01 0x00 0x03
14	3		LDC_W 1	0x13 0x00 0x01
23	3		LDC_W 0	0x13 0x00 0x00
26	2		ILOAD 2	0x15 0x02
28	1		IADD	0x60
29	3		INVOKEVIRTUAL 2	0xB6 0x00 0x02
43	3		INVOKEVIRTUAL 2	0xB6 0x00 0x02
49	1		NOP	0x00

II LOADER

A questo punto il LOADER porta tutti i moduli oggetto in memoria centrale.

Dal momento che l'assembler del IJVM è un linguaggio ***rilocabile*** in caso di un'eventuale rilocazione del programma è necessario solamente aggiornare, nella Tabella delle Costanti, l'indirizzo di inizio dei sottoprogrammi.