L'Assembler x86

Le procedure

M. Rebaudengo - M. Sonza Reorda

Politecnico di Torino
Dip. di Automatica e Informatica

Le procedure

L'Assembler 8086 permette l'uso delle *procedure*, in modo analogo a quanto avviene nei linguaggi di alto livello.

Attraverso le *procedure* è possibile scrivere una volta sola quelle parti di codice che vengono ripetutamente eseguite in un programma, ottenendo vari vantaggi:

- **maggior leggibilità del codice**
- **risparmio di tempo per il programmatore**
- **risparmio di memoria occupata dal codice.**

Definizione di procedura

Per definire una procedura si utilizzano le direttive `PROC` e `ENDP`.

```
label     PROC tipo
          ...                ; corpo della
          ...                ; procedura
label     ENDP
```

Il campo *`label`* corrisponde al nome della procedura.

Il tipo può essere `NEAR` o `FAR`.

Procedure NEAR e FAR

Una procedura di tipo `NEAR` è richiamabile solo all'interno dello stesso segmento di codice

Una procedura di tipo `FAR` può essere chiamata da procedure appartenenti a segmenti di codici diversi.

Se il tipo della procedura non è specificato, l'assemblatore assume che sia `NEAR`.

Chiamata di una procedura

L'istruzione `CALL` trasferisce il controllo del flusso del programma a una procedura.

Formato

```
CALL target
```

Funzionamento

L'istruzione `CALL` esegue le seguenti operazioni:

- **salva nello stack *l'indirizzo di ritorno***
- **trasferisce il controllo all'operando `target`.**

L'indirizzo di ritorno è l'indirizzo dell'istruzione successiva a quella di `CALL`.

Ad essa la procedura *ritorna* al termine della sua esecuzione.

Tipi di procedure

L'istruzione `CALL` si comporta diversamente in base al tipo di procedura chiamata:

- **se la procedura chiamata è di tipo `NEAR`, carica nello stack solo il contenuto dell'Instruction Pointer (`IP`), cioè l'offset dell'istruzione successiva;**
- **se la procedura chiamata è di tipo `FAR`, carica nello stack prima il contenuto del registro di segmento `CS` e poi il contenuto del registro `IP`.**

Operando *target*

L'operando target può essere:

- **un *indirizzo diretto***
- **un *indirizzo indiretto*.**

Nel caso di indirizzo diretto, l'operando target è costituito dal nome stesso della procedura.

Nel caso di indirizzo indiretto, *un registro* o *una variabile di memoria*, contenente l'indirizzo della procedura da eseguire.

Esempi:

- **`CALL AX`**
- **`CALL TABLE[SI]`**

L'istruzione `RET`

L'istruzione `RET` permette di restituire il controllo alla procedura chiamante, una volta che la procedura chiamata ha terminato l'esecuzione.

Formato

`RET pop_value`

L'operando *`pop_value`* è opzionale e corrisponde ad un valore immediato; esso permette di eseguire l'operazione di liberazione dello stack al momento del ritorno alla procedura chiamante di un numero di byte pari a *`pop_value`*.

L'istruzione `RET`

L'istruzione `RET` assume che l'indirizzo di ritorno si trovi in cima allo stack.

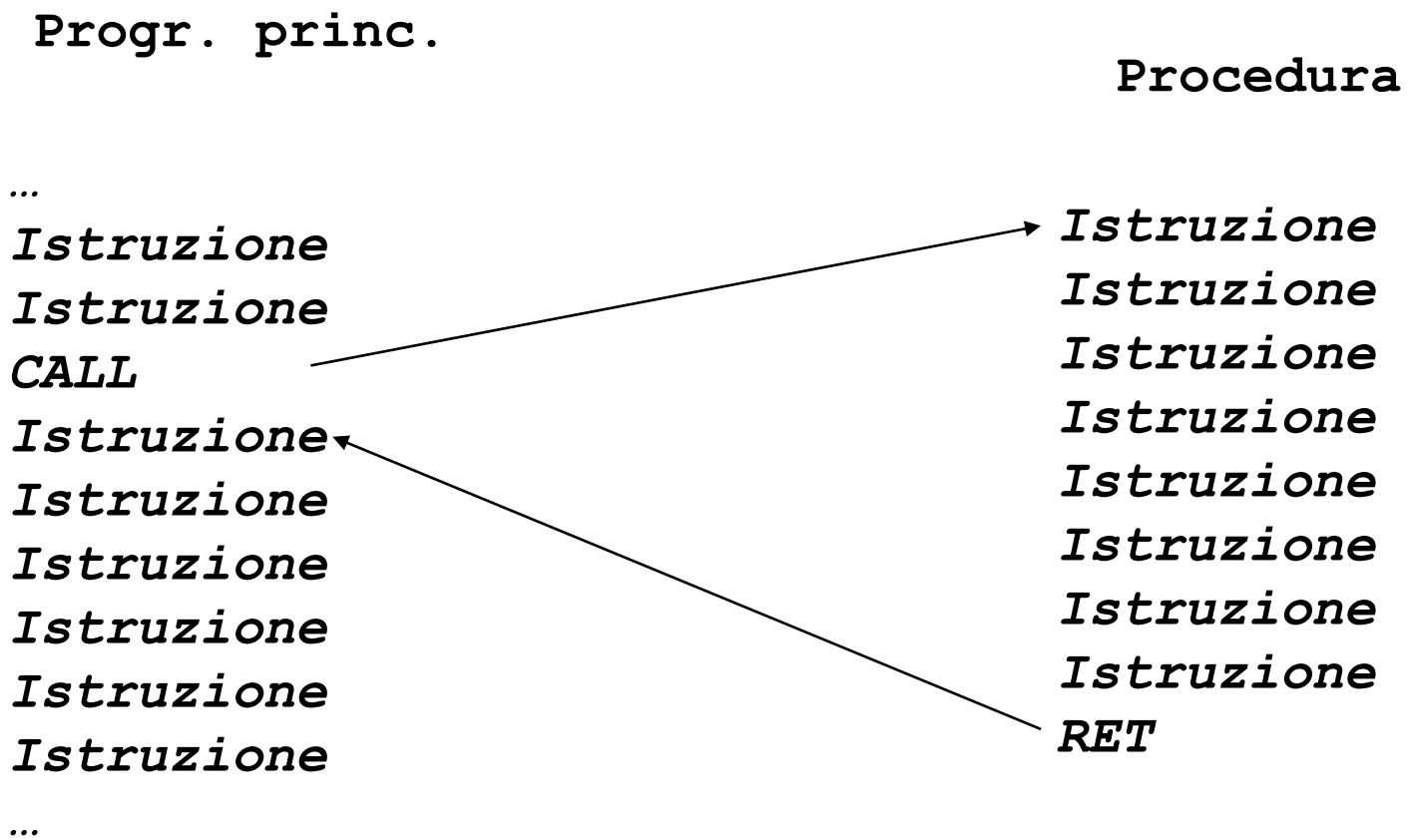
L'istruzione `RET` esegue le seguenti operazioni:

- ***pop* dallo stack dell'*indirizzo di ritorno***
- **eliminazione dallo stack di *pop_value* byte**
- ***salto* all'indirizzo di ritorno.**

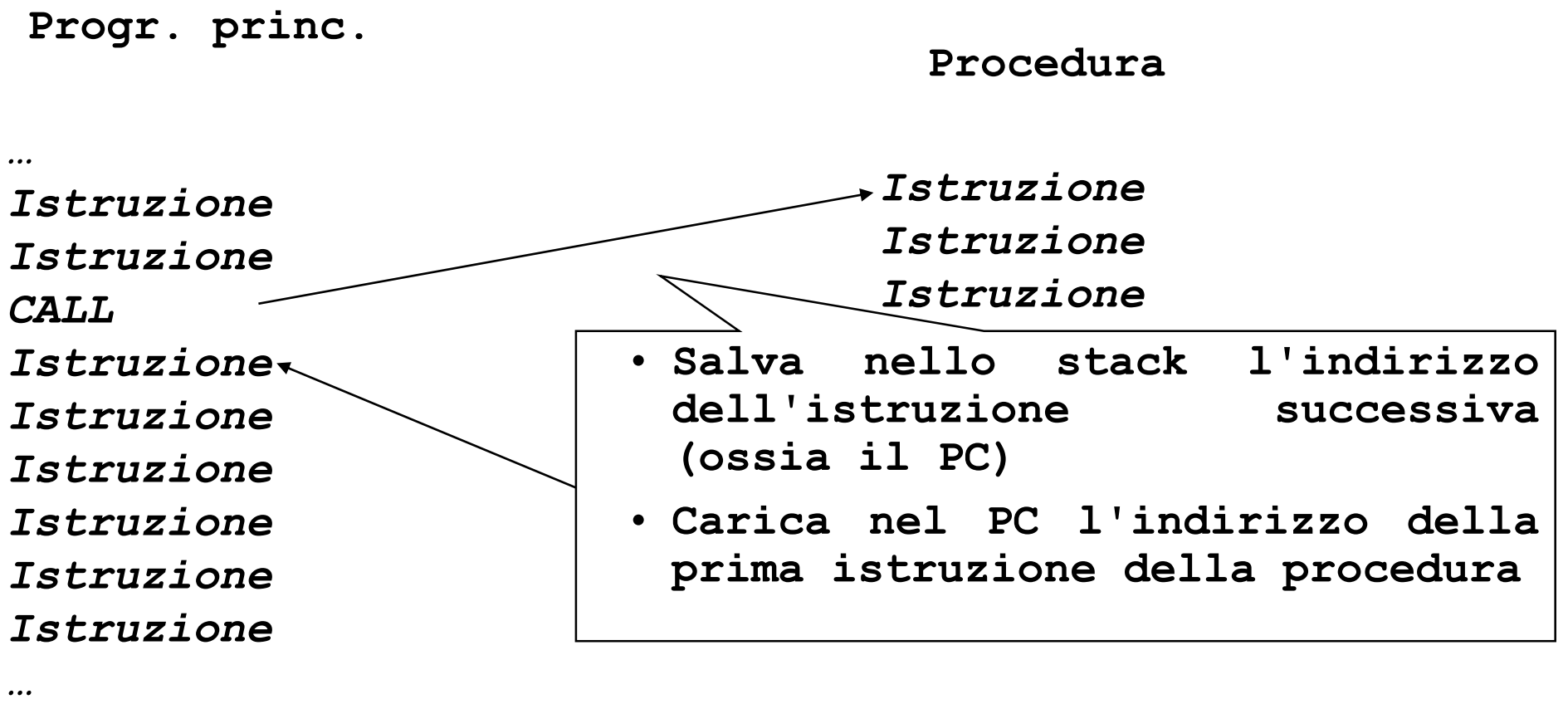
Se la procedura è di tipo `NEAR` il processore preleva dallo stack una word contenente l'offset dell'indirizzo di ritorno, e la scrive in IP.

Se la procedura è di tipo `FAR` il processore preleva dallo stack due word equivalenti all'intero indirizzo di ritorno e ne copia il valore in CS e IP.

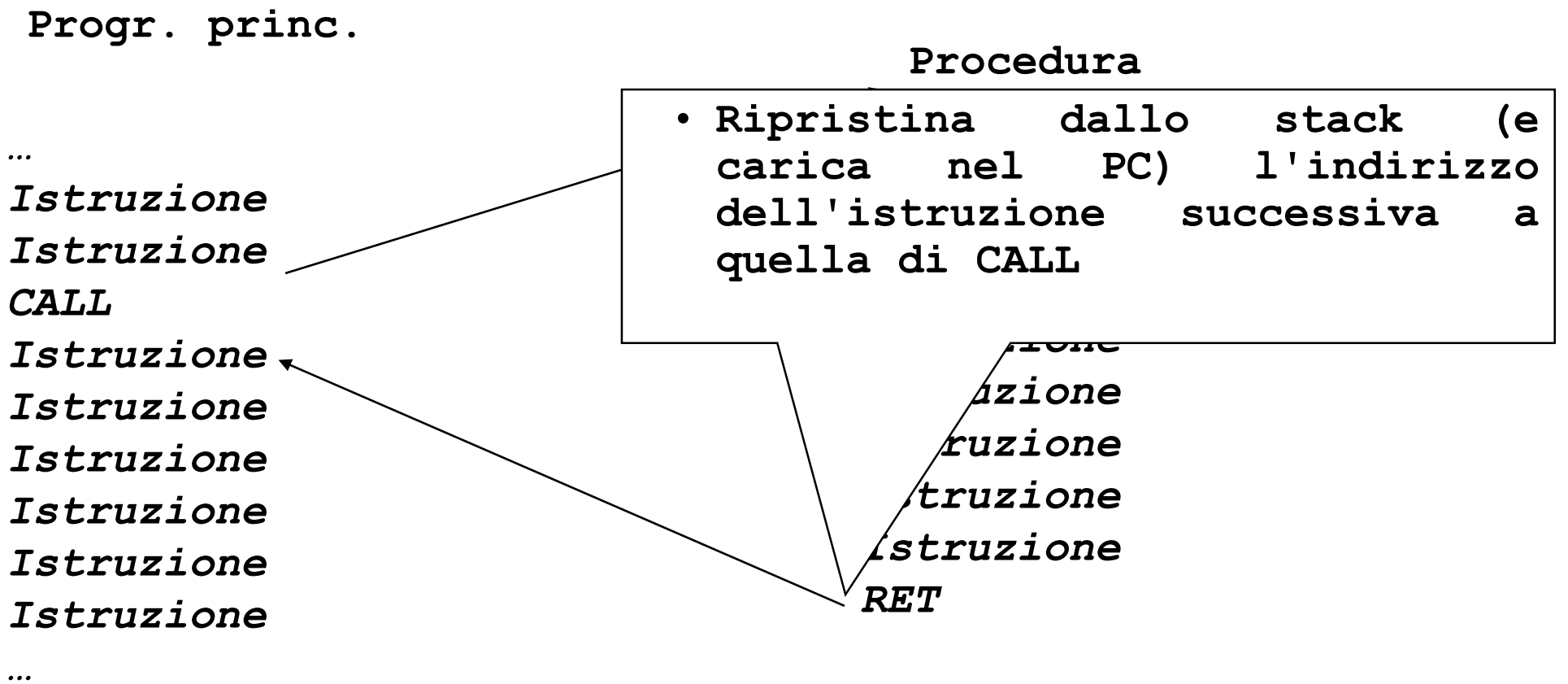
Chiamate a procedura



Chiamate a procedura



Chiamate a procedura



Vantaggi

L’uso dello stack nella gestione delle procedure presenta il vantaggio di permettere l’*annidamento* delle procedure.

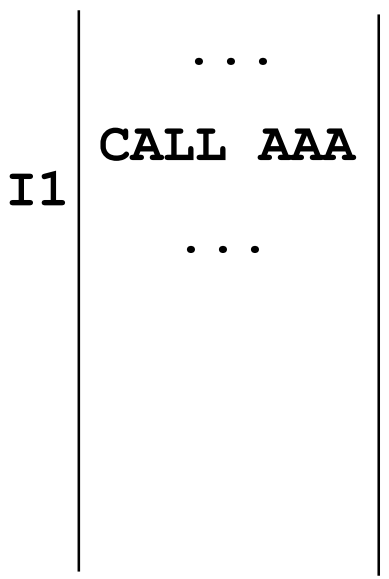
In tal caso ciascuna istruzione RET accede all’indirizzo di ritorno salvato nello stack dall’ultima istruzione CALL eseguita.

Procedure annidate

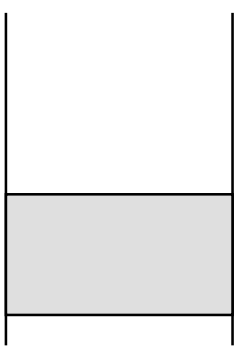
L’uso dello stack permette l’esecuzione di procedure annidate.

Il massimo livello di annidamento permesso è limitato dalle dimensioni dello stack.

Esempio

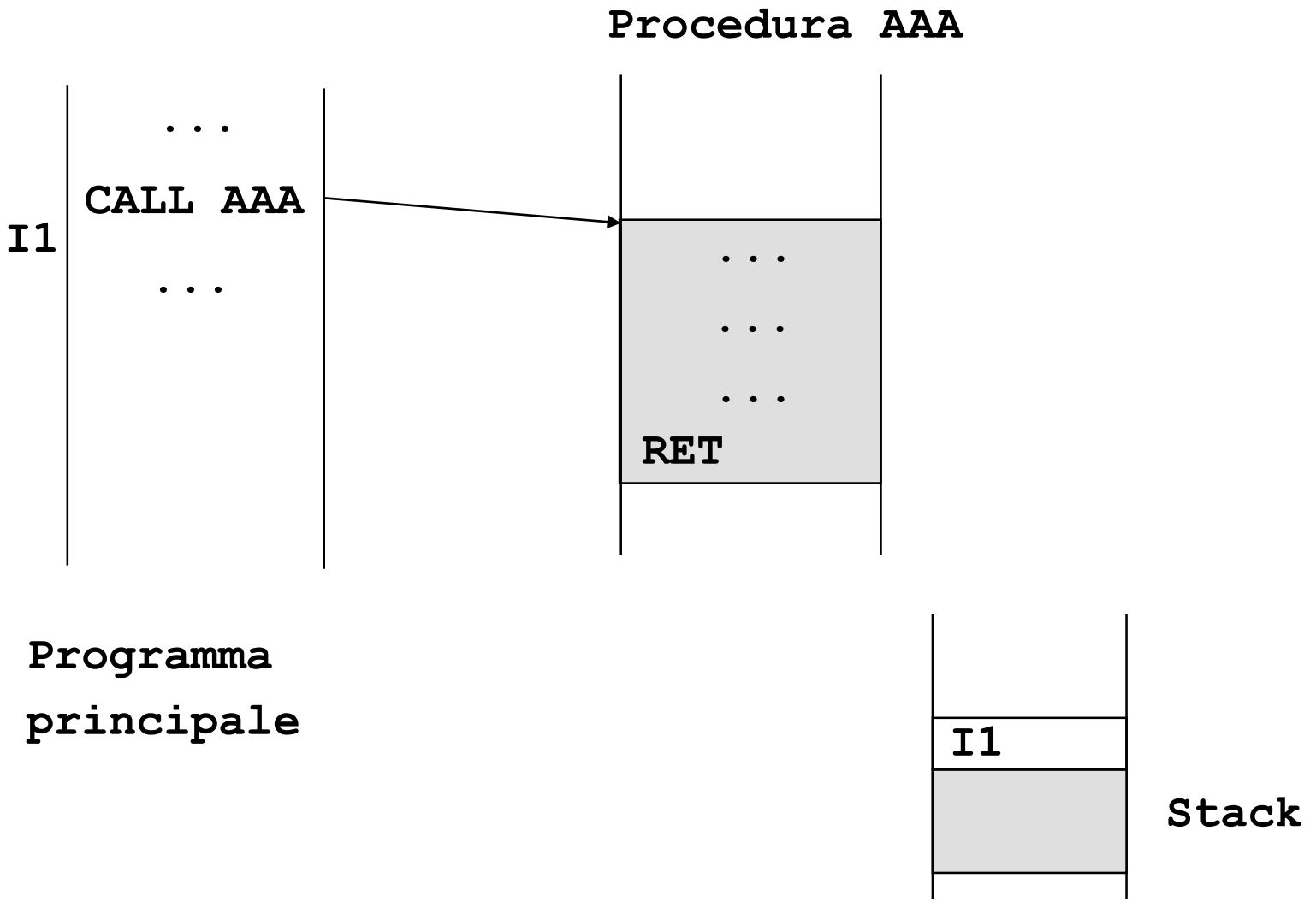


Programma
principale

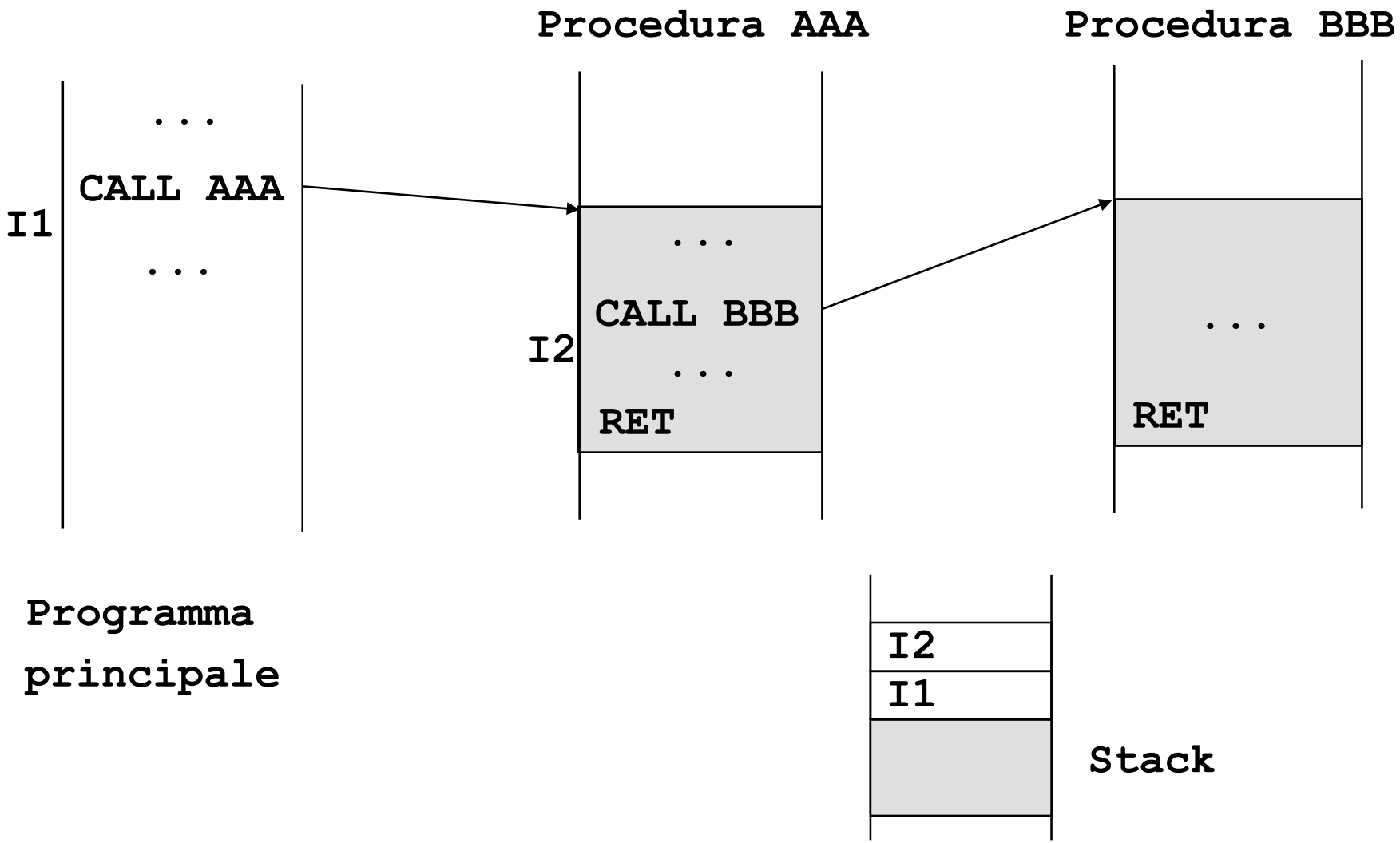


Stack

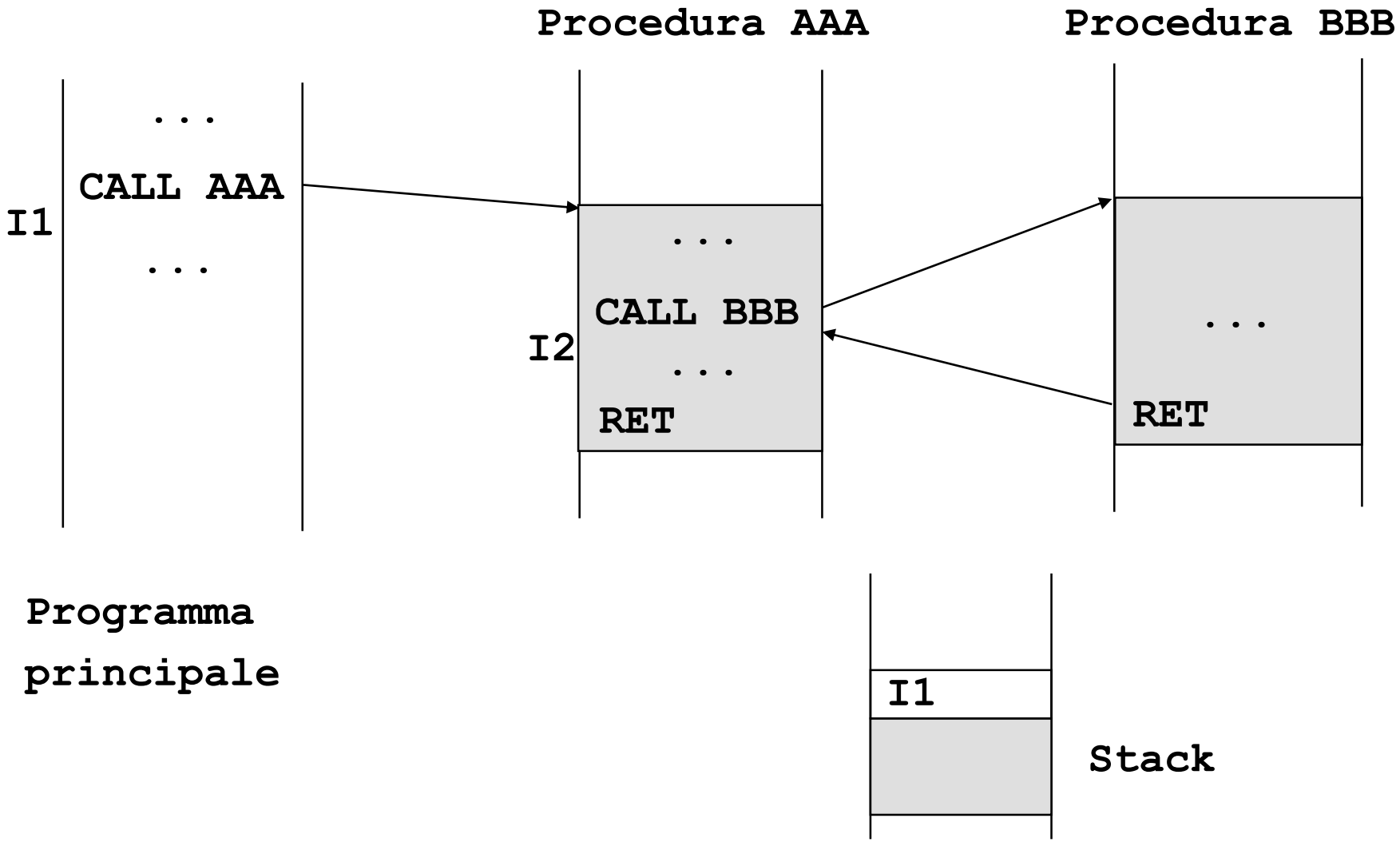
Esempio


Procedura AAA
...
CALL AAA
I1
...
...
...
...
RET
Programma
principale
I1
Stack

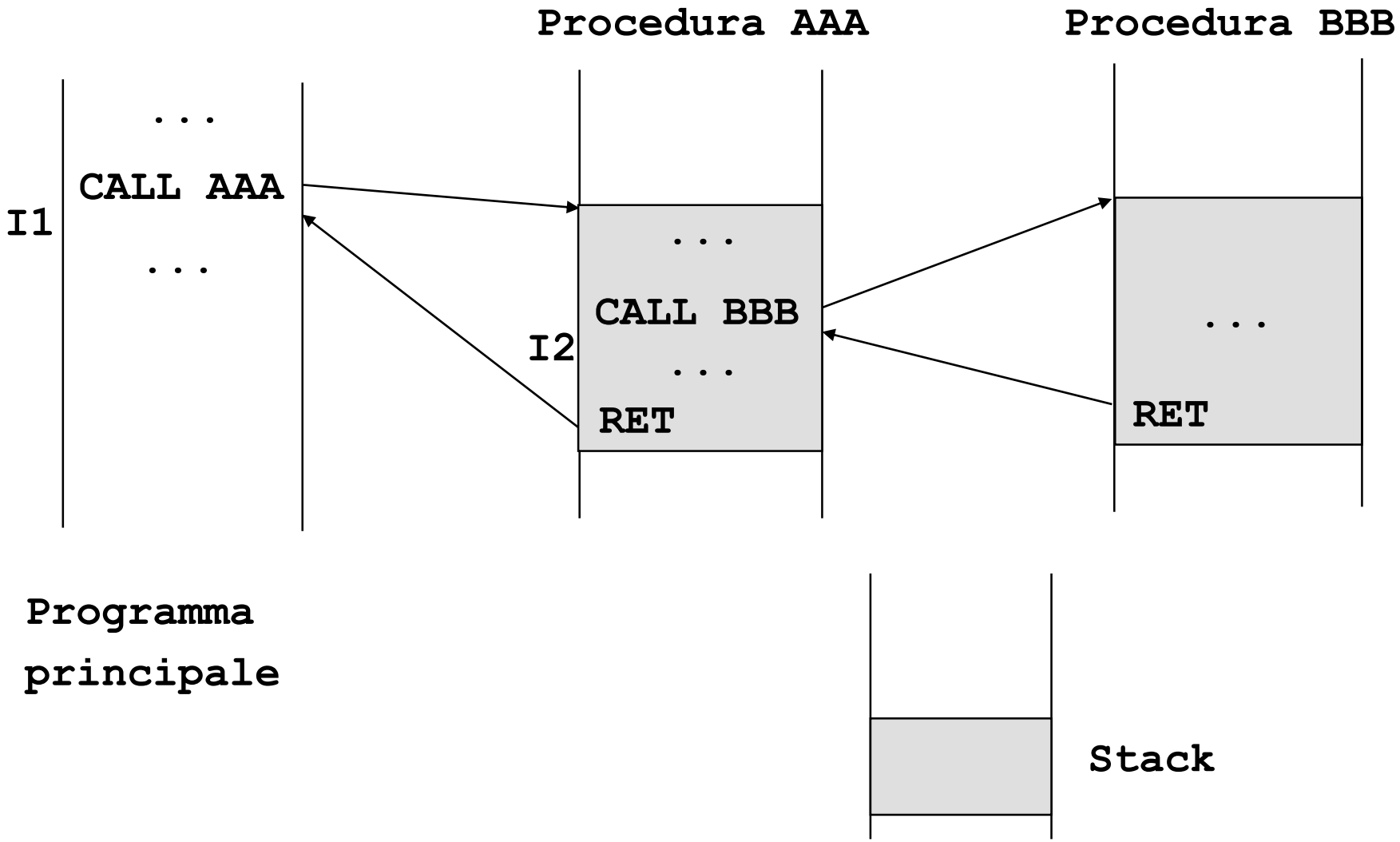
Esempio


Procedura AAA
Procedura BBB
...
CALL AAA
I1
...
...
CALL BBB
I2
...
RET
...
RET
Programma
principale
I2
I1
Stack

Esempio


Procedura AAA
Procedura BBB
...
CALL AAA
I1
...
...
CALL BBB
I2
...
RET
...
RET
Programma
principale
I1
Stack

Esempio


Procedura AAA
Procedura BBB
...
CALL AAA
I1
...
...
CALL BBB
I2
...
RET
...
RET
Programma
principale
Stack

Salvataggio dei registri

È buona regola fare in modo che ogni procedura

- **esegua come prima operazione il salvataggio nello stack di tutti i registri che vengono da essa modificati**
- **al termine ripristini i valori originari dei registri.**

Lo stack è una coda di tipo *LIFO*: l'ordine delle istruzioni `POP` deve essere l'inverso di quello delle istruzioni `PUSH`.

<u>Esempio</u>

```
XXX   PROC
      PUSH AX
      PUSH DX
      ...
      POP  DX
      POP  AX
      RET
XXX   ENDP
```

Passaggio di parametri

Le procedure comunicano con il programma chiamante attraverso i parametri.

Il passaggio dei parametri può avvenire tramite:

- **le variabili globali**
- **i registri**
- **lo stack.**

Caso di studio

```
int som_vett(int *vett, int count)
{
 int i, somma = 0;
     for (i=0 ; i < count ; i++)
          somma += vett[i];
     return(somma);
}
```

Variabili globali

Il modo più semplice per passare i parametri alle procedure consiste nell'utilizzare variabili globali, accessibili sia alla procedura chiamante sia a quella chiamata.

Questo metodo

- **è estremamente semplice da implementare**
- **è in generale sconsigliabile in quanto le procedure che utilizzano variabili globali come parametri sono poco riutilizzabili poiché la stessa procedura non può lavorare su dati diversi.**

Esempio

```
LUNG EQU  100
     .MODEL    small
     .STACK
     .DATA
VETT DW   LUNG DUP (?)
SOM  DW   ?
     ...
     .CODE
     ...
     CALL   SOM_VETT
     ...
```

```
SOM_VETT  PROC
          PUSH SI
          PUSH AX
          PUSH CX
          MOV  SI, 0
          MOV  AX, 0
          MOV  CX, LUNG
ciclo:    ADD  AX, VETT[SI]
          ADD  SI, 2
          LOOP ciclo
          MOV  SOM, AX
          POP  CX
          POP  AX
          POP  SI
          RET
SOM_VETT  ENDP
```

Uso dei registri

I parametri di ingresso ed uscita possono essere letti e scritti utilizzando i registri general purpose.

È un metodo

- **semplice ed estremamente rapido da implementare**
- **utilizzabile solo quando i parametri sono in numero limitato.**

Esempio

```
LUNG   EQU 100
       .MODEL   small
       .STACK
       .DATA
VETT   DW  LUNG DUP (?)
SOMMA  DW  ?
       .CODE
           …
       MOV  AX, LENGTH VETT
       LEA  BX, VETT
       CALL SOM_VETT
       MOV  SOMMA, AX
           …
```

```
SOM _VETT PROC
          PUSH BX
          PUSH CX
          MOV  CX, AX
          MOV  AX, 0
ciclo:    ADD  AX, [BX]
          ADD  BX, 2
          LOOP ciclo
          POP  CX
          POP  BX
          RET
SOM_VETT  ENDP
```

Uso dello stack

Il metodo più flessibile per il passaggio dei parametri si basa sullo stack.

Tale metodo

- **è più complesso da implementare**
- **non ha limiti sul numero di parametri (il limite sta nella dimensione dello stack)**
- **la memoria richiesta per i parametri è allocata solo per il tempo corrispondente all'esecuzione della procedura.**

Caricamento dei dati nello stack

Prima dell'esecuzione dell'istruzione `CALL`, la procedura chiamante deve eseguire tante istruzioni di `PUSH` quanti sono i parametri da passare.

Esempio

```
...
PUSH LUNG
PUSH OFFSET VETT
CALL SOM_VETT
...
```

Lettura dei parametri di ingresso

L’istruzione CALL è eseguita dopo che i parametri sono stati caricati nello stack.

L’indirizzo di ritorno è caricato nello stack dopo tutti i parametri e si trova quindi in cima allo stack nel momento in cui la procedura inizia l’esecuzione.

Ciò significa che la procedura chiamata non può eseguire l’operazione di pop dallo stack per prelevare i parametri senza perdere l’indirizzo di ritorno.

Lettura dei parametri di ingresso

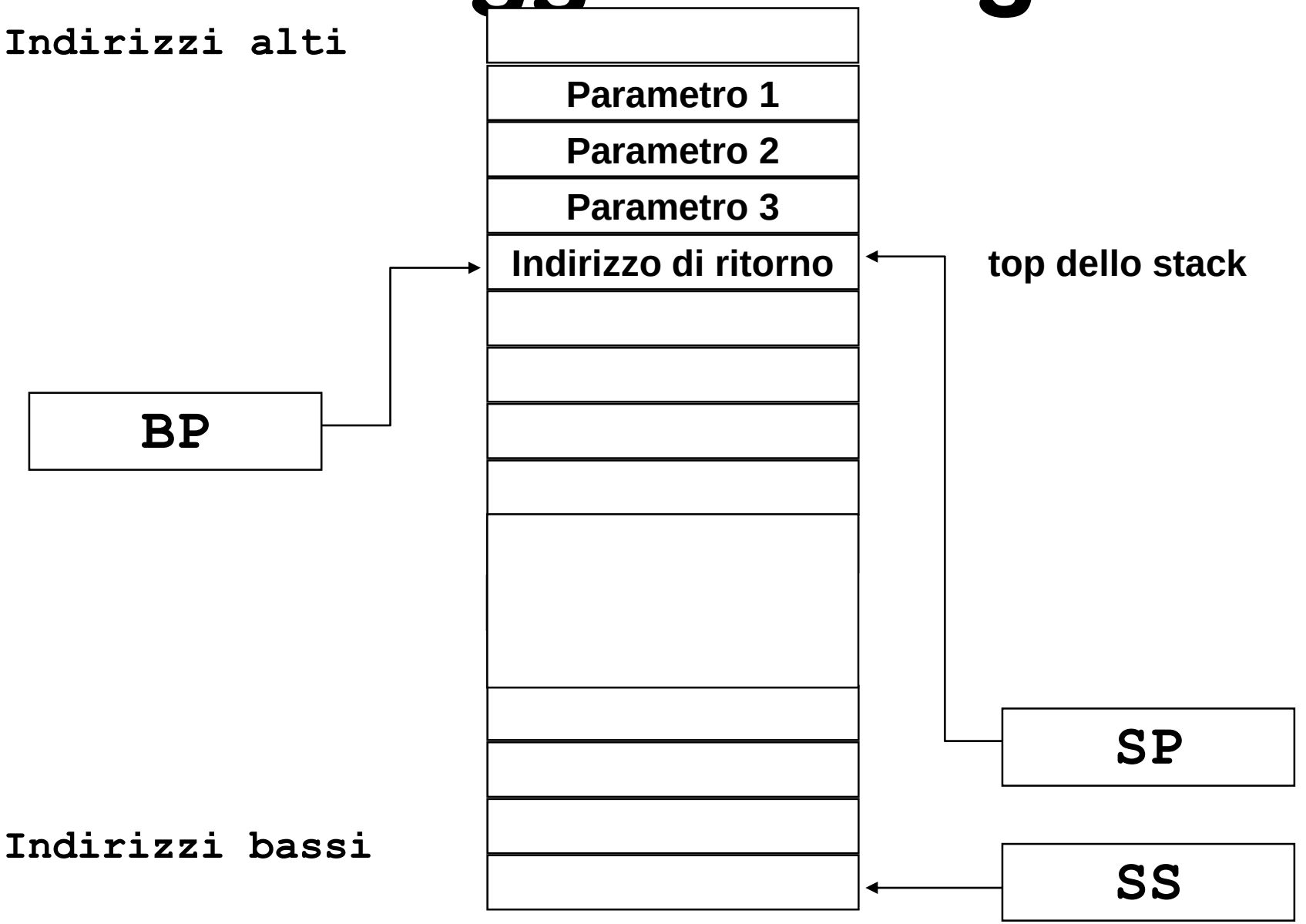
(segue)

Per la lettura dei parametri si utilizza il registro *Base Pointer* (`BP`) per fare accesso allo stack.

Il registro `BP` permette di accedere ai dati presenti nello stack senza eseguire operazioni di *push* e di *pop*, ossia senza cambiare il contenuto del registro `SP`.

La prima operazione da effettuare all'interno di una procedura è copiare il valore del registro `SP` nel registro `BP`.

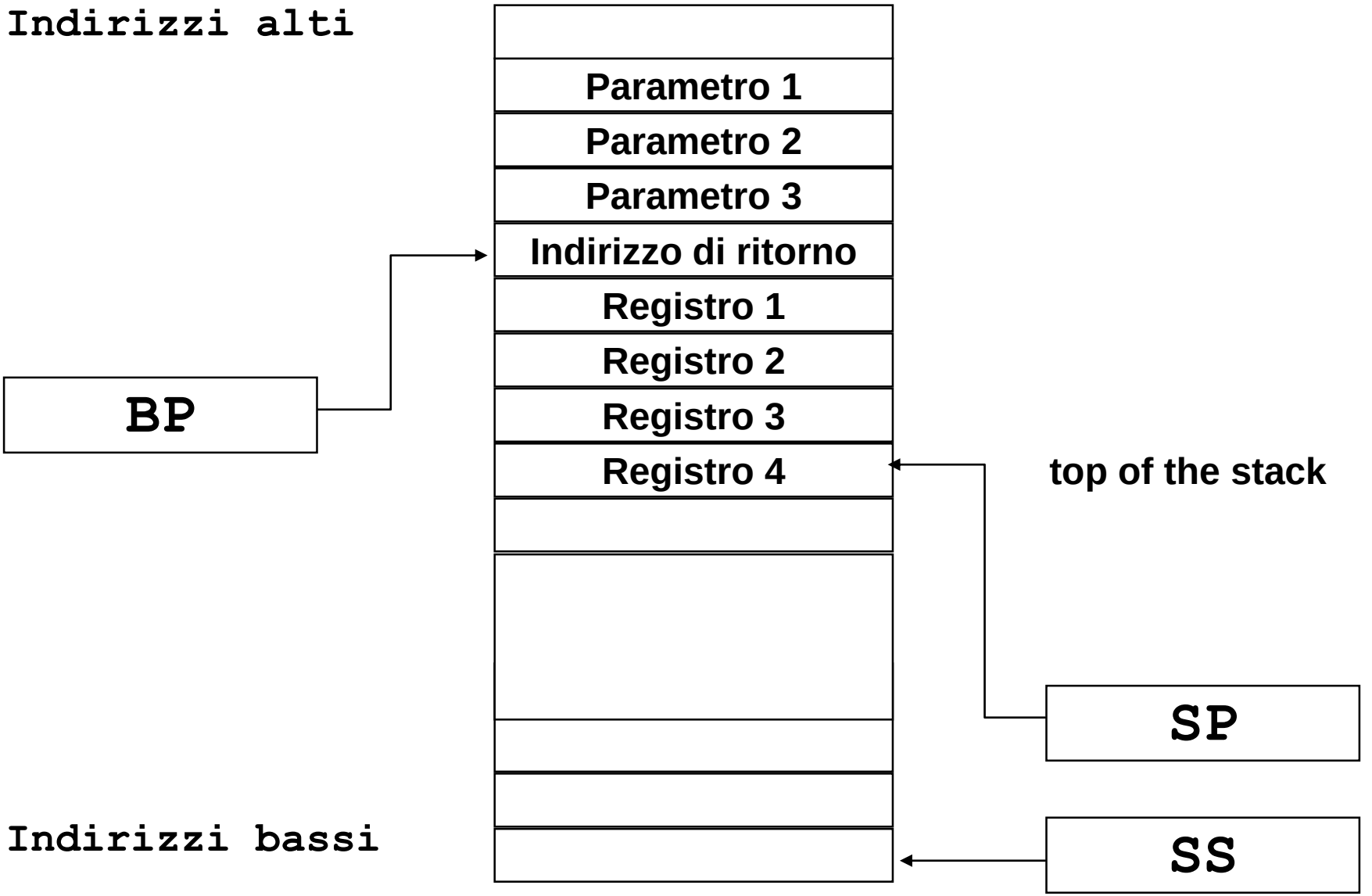
Stato dello stack prima del salvataggio dei registri



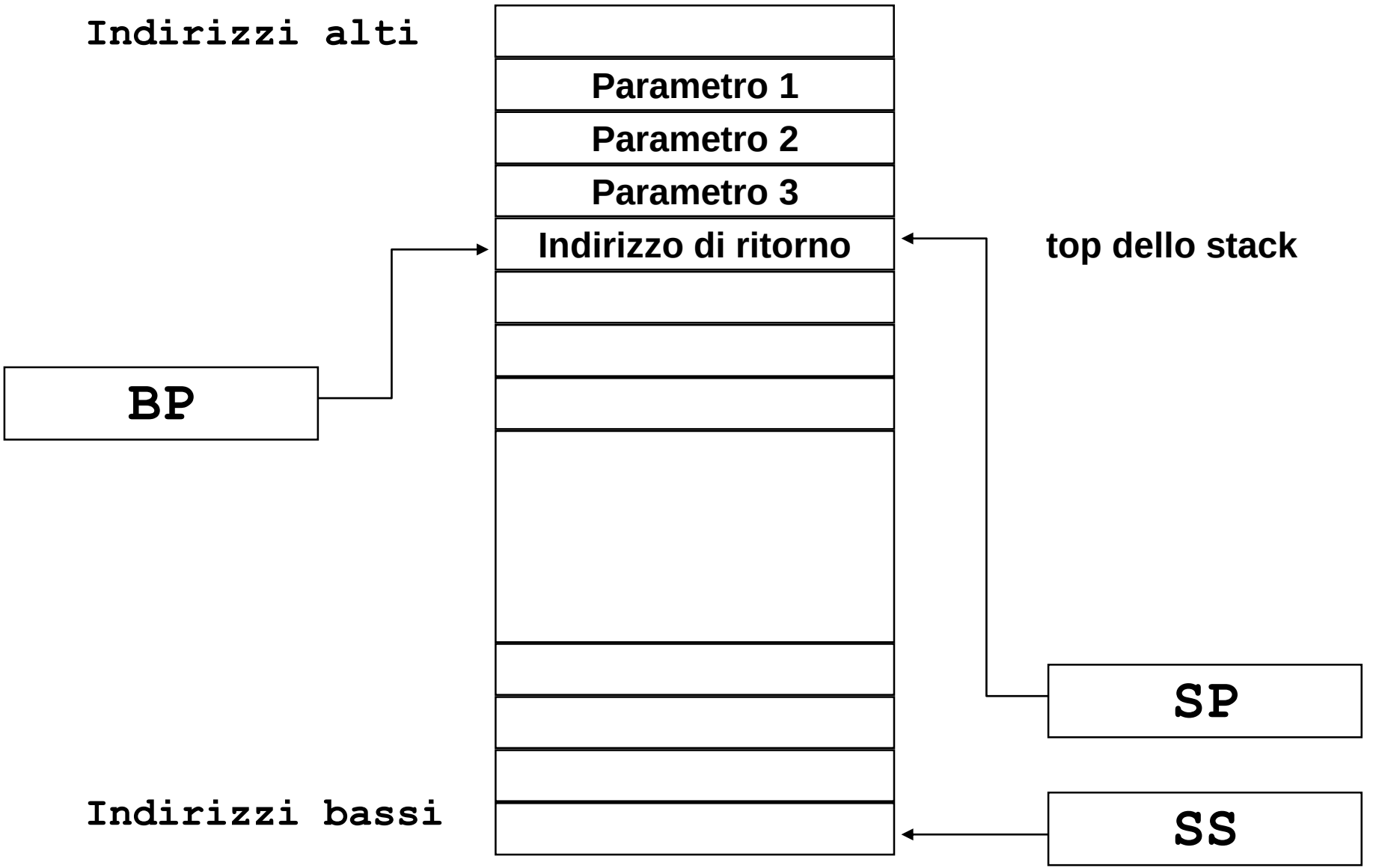
Salvataggio dei registri

Una volta che il registro `BP` è caricato, la procedura chiamata può salvare i registri nello stack.

Stato dello stack dopo il salvataggio dei registri



Stack prima della chiamata della RET



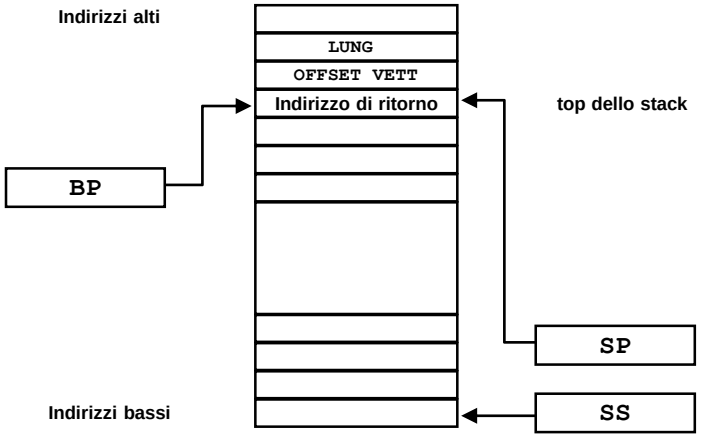
Parametri in uscita

È possibile utilizzare lo stack anche per passare alla procedura chiamante i parametri di uscita.

Essi non possono essere caricati nello stack con un'operazione di push perché altrimenti sarebbero posizionati in cima allo stack e non permetterebbero un corretto ritorno alla procedura chiamante.

Anche per la scrittura dei parametri nello stack conviene utilizzare il registro `BP`.

È compito della procedura chiamante eseguire le opportune operazioni di pop per la lettura dei valori di ritorno.



Esempio

```
SOM _VETT PROC
     MOV  BP, SP
     PUSH BX
     PUSH CX
     MOV  CX, [BP+4]
     MOV  BX, [BP+2]
     MOV  AX, 0
ciclo:    ADD  AX, [BX]
          ADD  BX, 2
          LOOP ciclo
          POP  CX
          POP  BX
          RET
SOM_VETT  ENDP
```

Liberazione dello stack

Di norma è compito della procedura chiamante liberare lo stack, cancellando le parole che sono state utilizzate per il passaggio dei parametri.

La liberazione dello stack può essere fatta:

- **con successive operazioni di *pop***
- **incrementando opportunamente il valore di `SP`.**

Esempio

```
PUSH PARAM1
PUSH PARAM2
PUSH PARAM3
CALL MY_PROC
POP  DX
POP  DX
POP  DX
```

```
PUSH PARAM1
PUSH PARAM2
PUSH PARAM3
CALL MY_PROC
ADD  SP, 6
```

Liberazione dello stack

(segue)

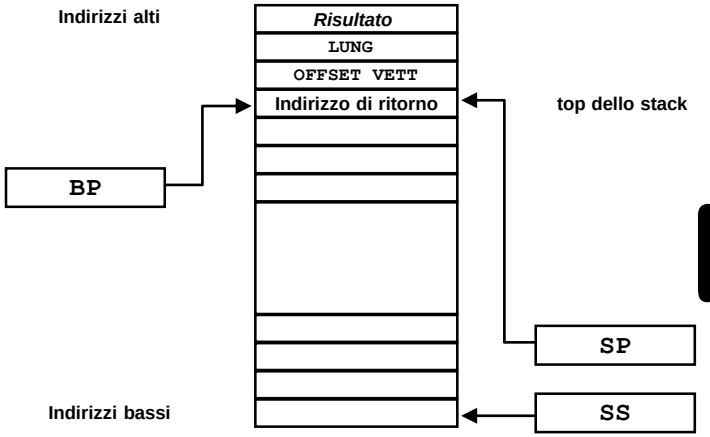
La pulizia dello stack può anche essere fatta all'interno della procedura chiamante mediante l'esecuzione dell'istruzione `RET`.

Questa può avere un operando immediato, che rappresenta il numero di byte da togliere dallo stack.

```
SOM_VETT PROC
     MOV  BP, SP
     PUSH BX
     PUSH CX
     MOV  CX, [BP+4]
     MOV  BX, [BP+2]
     MOV  AX, 0
ciclo:    ADD  AX, [BX]
          ADD  BX, 2
          LOOP ciclo
          POP  CX
          POP  BX
          RET    4
SOM_VETT  ENDP
```

Esempio: programma chiamante

```
...
SUB   SP, 2
PUSH LUNG
PUSH OFFSET VETT
CALL SOM_VETT
ADD   SP, 4
POP   SOMMA
```



Esempio: procedura

```
SOM _VETT PROC
     MOV  BP, SP
     PUSH BX
     PUSH CX
     MOV  CX, [BP+4]
     MOV  BX, [BP+2]
     MOV  AX, 0
ciclo:    ADD  AX, [BX]
          ADD  BX, 2
          LOOP ciclo
          MOV [BP+6], AX
          POP  CX
          POP  BX
          RET
SOM_VETT  ENDP
```