

# Corso di Architettura degli Elaboratori e Laboratorio (M-Z)

## Insieme di istruzioni macchina

*Nino Cauli*

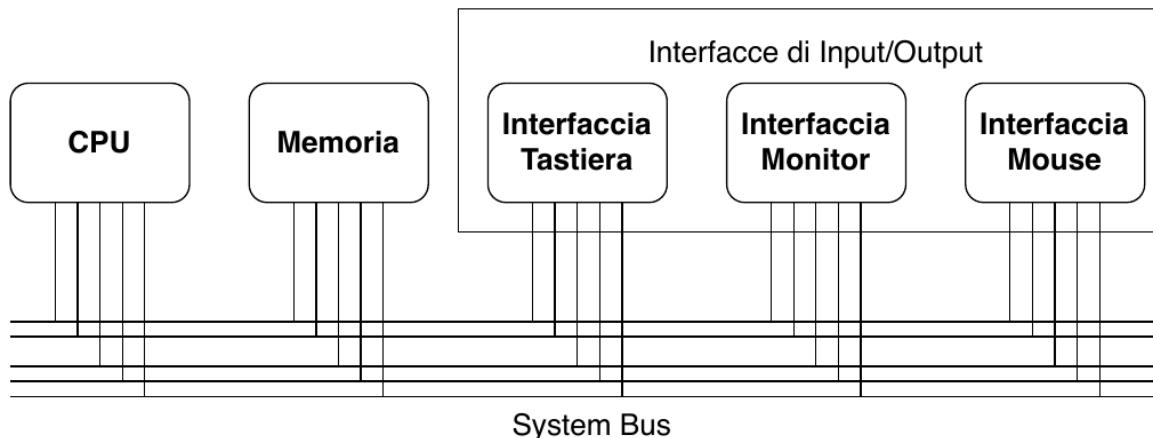


UNIVERSITÀ  
degli STUDI  
di CATANIA

Dipartimento di Matematica e Informatica

# Architettura base di un calcolatore elettronico

- **CPU**: esegue istruzioni elementari
- **MEMORIA**: contiene il programma (sequenza di istruzioni elementari) che la CPU deve eseguire e i dati necessari
- **INTERFACCE DI INPUT/OUTPUT**: circuiti elettronici che permettono di connettere la CPU al mondo esterno
- **BUS DI SISTEMA**: insieme di collegamenti elettrici che interconnettono i vari componenti di un calcolatore

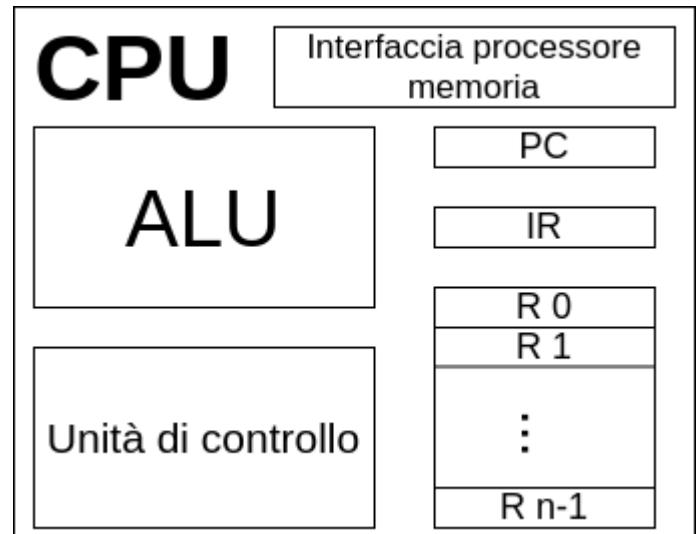


# Come funziona?

- Il calcolatore elettronico esegue **SEQUENZIALMENTE** una serie di **ISTRUZIONI**
- Le istruzioni definiscono delle operazioni da eseguire e sono raggruppate in **PROGRAMMI**
- Spesso le operazioni devono essere eseguite su dei **DATI**
- L'utente può interagire con il calcolatore tramite le **INTERFACCE DI I/O (PERIFERICHE)**

# Processore (CPU)

- È un **CIRCUITO ELETTRONICO INTEGRATO** (chip) con il ruolo di **CERVELLO** del calcolatore
- Capace di caricare ed eseguire le **ISTRUZIONI ELEMENTARI** necessarie per eseguire i **PROGRAMMI**
- Esempi di istruzioni elementari: operazioni aritmetiche, operazioni logiche, confronti, salti incondizionati e condizionati.



- Le unità memoria sono usate per immagazzinare informazione necessaria per eseguire i programmi
- Sono circuiti elettronici in grado di preservare l'informazione che può essere costituita da:
  - **ISTRUZIONI**, eseguite dalla CPU
  - **DATI**, utilizzati dalle istruzioni eseguite
- La memoria si può dividere in **MEMORIA CENTRALE** e **MEMORIA DI MASSA**

## Passi operativi elementari per ciascuna istruzione:

- **PRELIEVO**: prelievo della prossima istruzione dalla memoria (scrivere la prossima istruzione nel registro di istruzione IR)
- **DECODIFICA**: decodifica dell'istruzione (quale operazione bisogna eseguire? Dove si trovano i dati da usare?)
- **ESECUZIONE**: esecuzione dell'istruzione (leggere o scrivere un dato in memoria, eseguire operazioni matematiche e logiche sui registri)

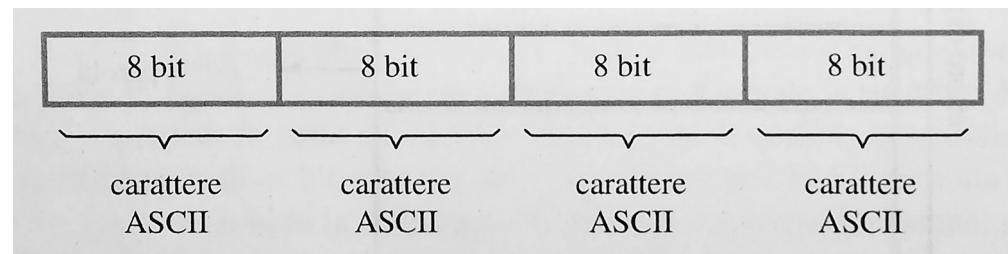
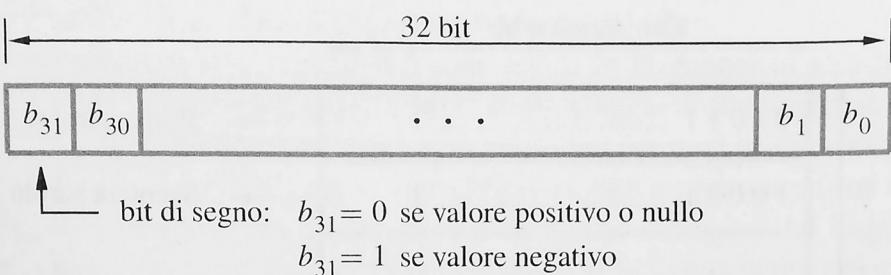
# Rappresentazione di istruzioni e dati

- Le istruzioni e i dati sono rappresentati da **SEQUENZE** di **CIFRE BINARIE (bit)**
- Per convenzione una sequenza di 8 bit è detta **Byte**
- I byte vengono raggruppati in blocchi con un numero di elementi espresso con potenze di 2:
  - Kilobyte = KB =  $2^{10} = 1024 \approx 10^3$
  - Megabyte = MB =  $2^{20} = 1024 * 1024 \approx 10^6$
  - Gigabyte = GB =  $2^{30} = 1024 * 1024 * 1024 \approx 10^9$
  - Terabyte = TB =  $2^{40} = 1024 * 1024 * 1024 * 1024 \approx 10^{12}$



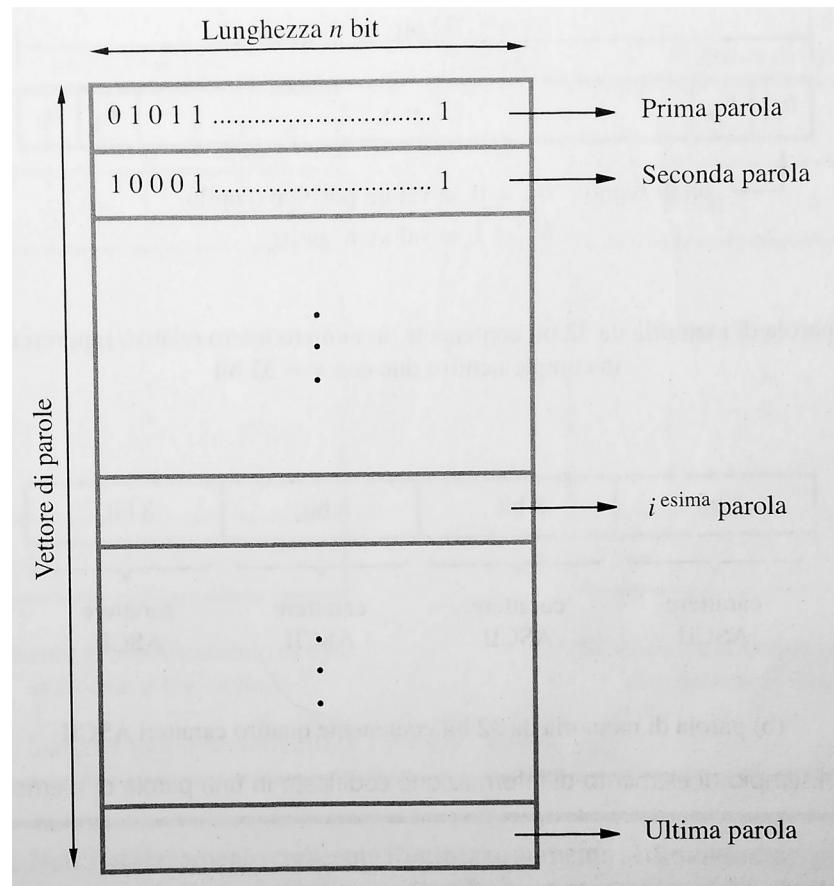
# Parola di memoria (memory word)

- Il calcolatore non lavora su singoli bit ma su gruppi di bit detti **PAROLE** di lunghezza da 8 a 64 bit (sempre potenze di 2)
- La dimensione delle parole dipende dall'architettura del calcolatore
- I dati possono occupare da un singolo byte a diverse parole
- Le istruzioni possono occupare una o più parole



# Organizzazione della memoria

- L'informazione è immagazzinata in memoria sotto forma di un vettore di parole (parole in successione)
- Ad ogni parola nel vettore è associato un indirizzo binario univoco
- Un numero binario di  $m$  bit può rappresentare  $2^m$  indirizzi
- Parole consecutive sono associate ad indirizzi consecutivi
- L'insieme degli indirizzi associati a ciascuna parola di memoria è chiamato spazio di indirizzamento



# Indirizzamento e ordinamento di byte



- Di norma l'unità minima di informazione indirizzabile in memoria è il byte
  - Si assegnano indirizzi consecutivi ai byte contenuti in ciascuna parola
  - Gli indirizzi delle parole saranno quindi multipli della loro lunghezza in byte
  - Vi sono 2 schemi di indirizzamento di byte:
    - **Crescente (big-endian)**: indirizzo aumenta al diminuire del peso aritmetico del byte
    - **Decrescente (little-endian)**: indirizzo aumenta all'aumentare del peso aritmetico del byte

Indirizzo di parola	Indirizzo di byte			
0	0	1	2	3
4	4	5	6	7
	.	.	.	.
$2^m - 4$	$2^m - 4$	$2^m - 3$	$2^m - 2$	$2^m - 1$

(a) schema di indirizzamento di byte crescente o big-endian

(b) schema di indirizzamento di byte  
*decrecente* o *little-endian*

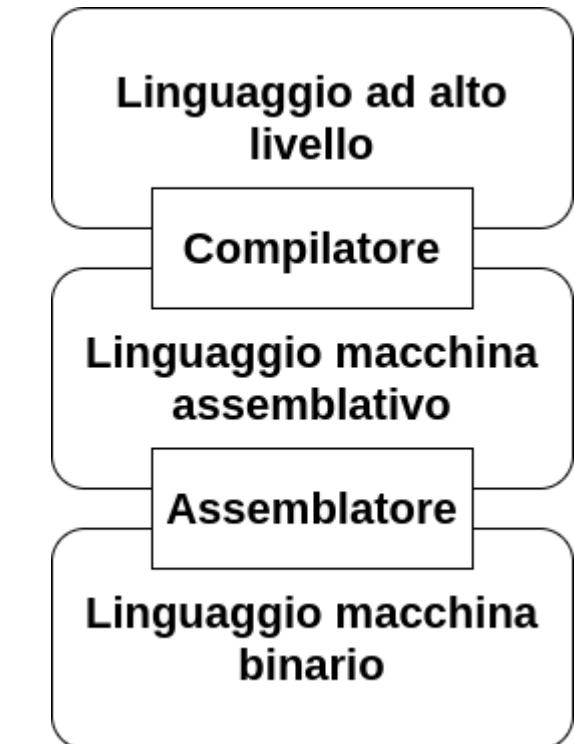
# Instruction set architecture ISA



- Il processore è in grado di eseguire un insieme di operazioni base chiamate istruzioni macchina
- L'insieme delle istruzioni eseguibili da un processore e le loro modalità d'uso è chiamato ISA (Instruction Set Architecture)
- Ogni processore commerciale ha il suo specifico ISA
- Il linguaggio macchina permette di definire le istruzioni attraverso un alfabeto binario {0, 1}
- Il linguaggio assemblativo è una rappresentazione simbolica leggibile del linguaggio macchina

# Come si programma?

- Il programmatore scrive i programmi in **LINGUAGGIO ASSEMBLATIVO (ASSEMBLY)**
- Il programma assemblativo viene tradotto in sequenze binarie dall'**ASSEMBLATORE**
- Linguaggi ad alto livello (C, C++, etc.) ancora più espressivi
- Il **COMPILATORE** traduce il codice ad alto livello in codice assemblativo



Esistono due approcci nella progettazione dell'insieme di istruzioni dei calcolatori:

## Reduced Instruction Set Computer (RISC):

- Insieme di istruzioni base ridotto
- Ogni istruzione occupa una sola parola di memoria
- Gli operandi delle istruzioni aritmetiche e logiche devono trovarsi nei registri del processore
- Prestazioni elevate grazie ad un'elaborazione a stadi (pipeline)

## Complex Instruction Set Computer (CISC):

- Insieme di istruzioni base complesse
- Ogni istruzione può occupare più di una parola di memoria
- Gli operandi delle istruzioni aritmetiche e logiche possono trovarsi in memoria

- I programmi eseguiti da un calcolatore sono composti da una sequenza di istruzioni base (addizione, confronto, caricamento di dati, ecc.)
- L'insieme di istruzioni riconosciute deve comprendere almeno queste quattro tipologie:
  - Trasferimento dati tra memoria e registri del processore
  - Operazioni aritmetiche e logiche sui dati
  - Operazioni di controllo dell'ordine di esecuzione delle istruzioni
  - Trasferimento dati tra unità di I/O e registri del processore

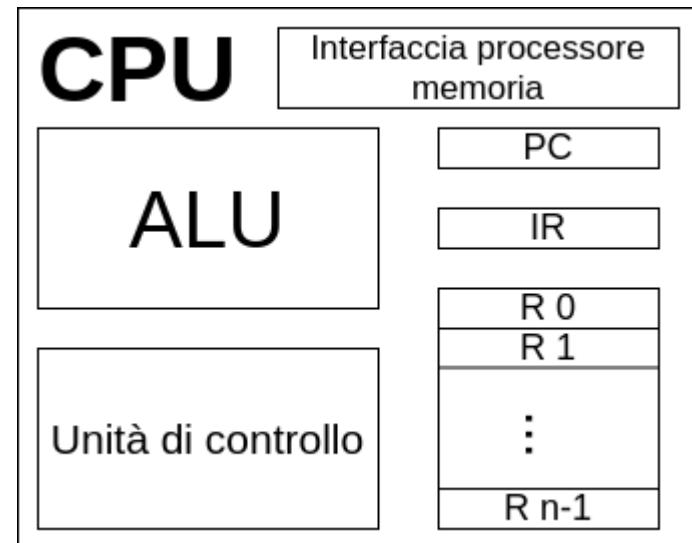
# Linguaggio assemblativo RISC generico



- Le diverse ISA dei processori commerciali posseggono linguaggi assemblativi con formalismi differenti (sebbene simili)
- Nella teoria di questo corso useremo un linguaggio assemblativo generico non appartenente a nessun processore commerciale
- Utile per capire i concetti base che possono essere applicati a qualsiasi architettura
- Verrà presentato un set di istruzioni base per programmare un processore nella pratica (non il set di istruzioni completo)

# Registri e locazioni di memoria

- È necessario definire una notazione formale per riferirsi ai registri e alle locazioni di memoria nel linguaggio assemblativo generico
- I registri sono identificati attraverso il loro nome:
  - Registri generici del processore: R0, R1, ..., Rn
  - Registri speciali del processore: PC, IR, ecc.
  - Registri di I/O: INGRESSO\_DATO, USCITA\_DATO, ecc.
- Le locazioni di memoria sono identificate attraverso il loro indirizzo in forma:
  - Di costante numerica
  - Di costante simbolica dichiarata in precedenza: VAR1, IND, CICLO, ecc.



## *Load destinazione sorgente*

- Istruzione usata per caricare un dato dalla memoria ad un registro del processore
- Il campo destinazione è il nome di un registro del processore
- Il campo sorgente è una locazione di memoria
- La locazione di memoria può essere indicata in vari modi a seconda del modi di indirizzamento usato

---

## *Store sorgente destinazione*

- Istruzione usata per salvare in memoria un dato presente in un registro del processore
- Il campo sorgente è il nome di un registro del processore
- Il campo destinazione è una locazione di memoria
- La locazione di memoria può essere indicata in vari modi a seconda del modi di indirizzamento usato

## *Add destinazione sorgente1 sorgente2*

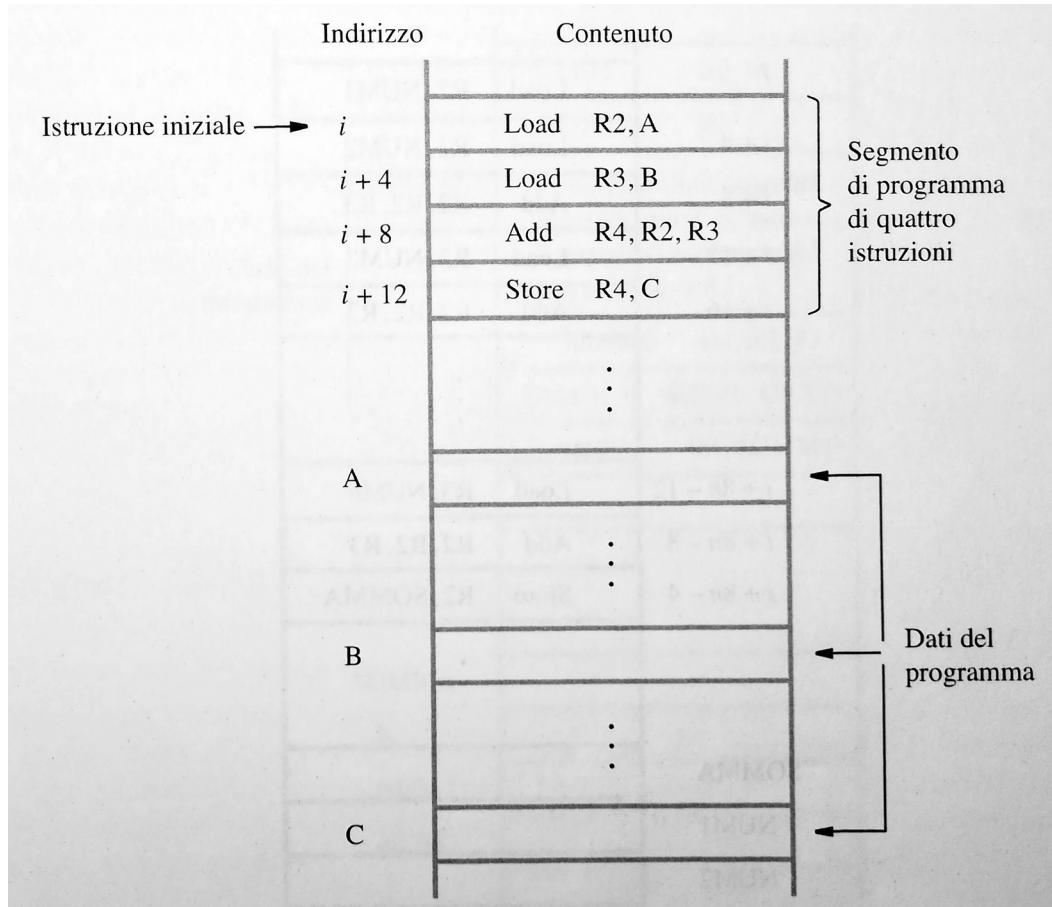
- Istruzione usata per sommare il contenuto di due registri
- Il campo destinazione è il nome di un registro del processore su cui scrivere la somma
- I campi sorgente1 e sorgente2 rappresentano i numeri da sommare
- Gli addendi possono essere espressi come nomi di registri o direttamente come valore

## *Subtract destinazione sorgente1 sorgente2*

- Istruzione usata per sottrarre il contenuto di due registri
- Il campo destinazione è il nome di un registro del processore su cui scrivere la differenza
- I campi sorgente1 e sorgente2 rappresentano i numeri da sottrarre
- Gli operandi possono essere espressi come nomi di registri o direttamente come valore

# Esempio di programma di somma

- Esempio di programma che somma due valori presenti in memoria e ne salva il risultato
- Programma composto da 4 istruzioni (2 Load, 1 Add e 1 Store)
- Le quattro istruzioni sono memorizzate in parole di memoria consecutive
- Istruzioni lette sequenzialmente
- PC contiene l'indirizzo della prossima istruzione da eseguire e IR contiene l'istruzione in esecuzione



# Modi di indirizzamento

- Nel linguaggio assemblativo, gli operandi e il risultato delle istruzioni possono essere espressi in modi diversi
- I metodi con cui specificare operandi e risultato vengono chiamati modi di indirizzamento
- I modi di indirizzamento base di un'architettura RISC sono:
  - **Modo immediato**
  - **Modo di registro**
  - **Modo assoluto (diretto)**
  - **Indiretto da registro**
  - **Con indice e spiazzamento**
  - **Con base e indice**

# Modi di registro e assoluto

- I modi di indirizzamento visti fino ad ora sono:
  - **Modo di registro:** Il nome (= indirizzo) di un registro di processore contenente l'operando o il risultato è dato nell'istruzione
  - **Modo assoluto (diretto):** L'indirizzo di una parola di memoria contenente l'operando o il risultato è dato nell'istruzione
- Nei processori RISC c'è un limite al numero di bit per un indirizzo assoluto (un'istruzione = una parola)
- Per processori a 32 bit = indirizzo assoluto 16 bit

**Load R2, NUM1**

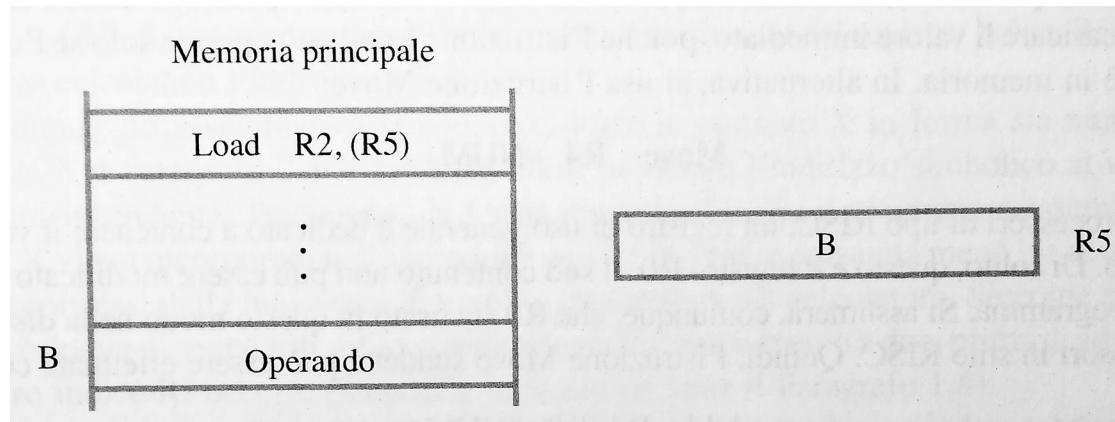
# modo immediato

- Per usare una costante numerica come operando si ricorre al modo di indirizzamento immediato
- **Modo immediato:** L'operando è dato esplicitamente nell'istruzione
- Si precede la costante dal simbolo cancelletto: **#valore**
- Esempio in cui si aggiunge il valore 200 al contenuto di R6 e si pone il risultato in R4:

**Add R4, R6, #200**

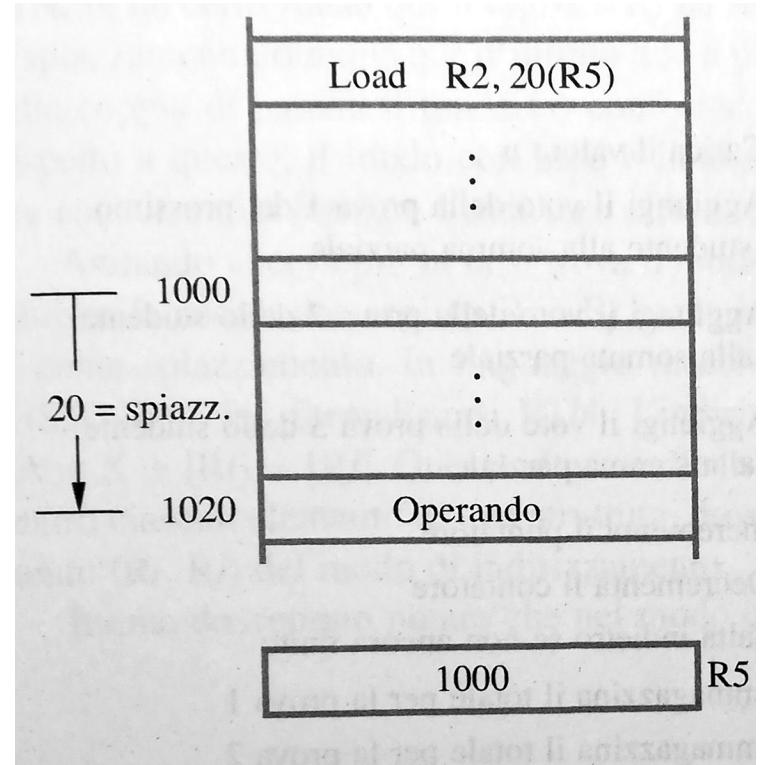
# Modo indiretto da registro

- **Modo indiretto:** Il nome di un registro di processore contenente l'INDIRIZZO di memoria dell'operando o del risultato è dato nell'istruzione
- Viene rappresentato con il nome del registro tra parentesi tonde (·)
- Usato in casi in cui si voglia riutilizzare una stessa istruzione in memoria più volte cambiando gli operandi



# Modo con indice e spiazzamento

- **Modo con indice e spiazzamento:** L'indirizzo effettivo di operando o risultato è ottenuto addizionando un valore costante (spiazzamento) al contenuto di un registro (indirizzo)
- Per indicare indice e spiazzamento si usa la scrittura  $X(Ri)$ , dove  $X$  è lo spiazzamento e  $Ri$  è il nome del registro contenente l'indirizzo
- Utile nel gestire vettori o liste
- Esistono versioni più complesse come il **modo con base e indice** dove l'indirizzo effettivo è ottenuto sommando il contenuto di due registri, denotato così:  $(Ri, Rj)$



## Branch\_if\_condizione destinazione\_salto

- Istruzione usata per saltare all'esecuzione di un'istruzione specifica nel caso la condizione di salto sia vera
- La condizione di salto può essere tra valori contenuti nei registri (espressi tra quadre: [R<sub>i</sub>]) o valori espressi esplicitamente
- La destinazione del salto è espressa come locazione di memoria contenente l'istruzione da eseguire nel caso la condizione sia vera
- Esempio di salto all'istruzione CICLO nel caso il contenuto di R2 sia maggiore di 0:

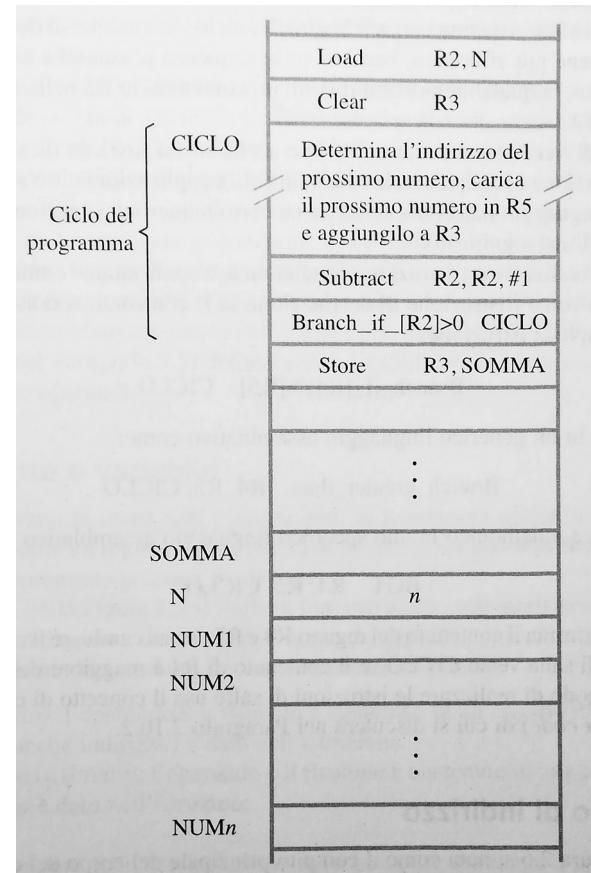
**Branch\_if\_[R2]>0 CICLO**

# Esempio somma di n numeri

$i$	Load R2, NUM1
$i + 4$	Load R3, NUM2
$i + 8$	Add R2, R2, R3
$i + 12$	Load R3, NUM3
$i + 16$	Add R2, R2, R3
	:
	:
$i + 8n - 12$	Load R3, NUM $n$
$i + 8n - 8$	Add R2, R2, R3
$i + 8n - 4$	Store R2, SOMMA
	:
SOMMA	
NUM1	
NUM2	
	:
NUM $n$	

## Esempio sequenziale (poco efficiente)

## Esempio con salto (più efficiente)



# Esempio somma di n numeri con salto

CICLO:	Load	R2,N	Carica la dimensione della lista
	Clear	R3	Inizializza la somma a 0
	Move	R4,#NUM1	Carica l'indirizzo del primo numero
	Load	R5, (R4)	Preleva il prossimo numero
	Add	R3, R3, R5	Aggiungi questo numero alla somma
	Add	R4, R4, #4	Incrementa il puntatore alla lista
	Subtract	R2, R2, #1	Decrementa il contatore
	Branch_if_[R2]>0	CICLO	Salta indietro se non ancora finito
	Store	R3, SOMMA	Immagazzina la somma finale

# Esempio voti studenti

N	n (num.d'elementi in lista)	Elemento allievo 1
LISTA	Num. di matricola allievo	
LISTA + 4	Voto della prova 1	Elemento allievo 2
LISTA + 8	Voto della prova 2	
LISTA + 12	Voto della prova 3	Elemento allievo 1
LISTA + 16	Num. di matricola allievo	
	Voto della prova 1	Elemento allievo 2
	Voto della prova 2	
	Voto della prova 3	Elemento allievo 1
	:	

Move	R2, #LISTA	Carica l'indirizzo LISTA
Clear	R3	
Clear	R4	
Clear	R5	
Load	R6, N	Carica il valore n
CICLO:	Load R7, 4(R2)	Aggiungi il voto della prova 1 del prossimo studente alla somma parziale
	Add R3, R3, R7	
	Load R7, 8(R2)	Aggiungi il voto della prova 2 dello studente alla somma parziale
	Add R4, R4, R7	
	Load R7, 12(R2)	Aggiungi il voto della prova 3 dello studente alla somma parziale
	Add R5, R5, R7	
	Add R2, R2, #16	Incrementa il puntatore
	Subtract R6, R6, #1	Decrementa il contatore
	Branch_if_[R6]>0 CICLO	Salta indietro se non ancora finito
	Store R3, SOMMA1	Immagazzina il totale per la prova 1
	Store R4, SOMMA2	Immagazzina il totale per la prova 2
	Store R5, SOMMA3	Immagazzina il totale per la prova 3

# Direttive di assemblatore



- L'assemblatore è in grado di produrre il codice macchina binario (programma oggetto) di un programma scritto in codice assemblativo (programma sorgente)
- Per produrre il programma oggetto l'assemblatore deve risolvere inoltre i seguenti problemi:
  - Assegnare valori numerici a nomi e simboli
  - Dove collocare in memoria le istruzioni macchina
  - Dove collocare in memoria gli operandi e i risultati del programma
- Per questo il linguaggio assemblativo non contiene solo le istruzioni del programma, ma anche comandi specifici per l'assemblatore (direttive di assemblatore)
- Le direttive di assemblatore non vengono tradotte nel programma oggetto, ma servono per dare informazioni utili all'assemblatore

# Dichiarazione di eguaglianza



- Serve per associare un valore numerico ad un nome usato nel programma sorgente
- La sua sintassi nel nostro linguaggio generico è la seguente:

*NOME*      **EQU**    *Valore\_numerico*

- Per produrre il programma oggetto, l'assemblatore sostituirà ogni occorrenza della stringa *NOME* nel programma sorgente con il valore *Valore\_numerico*

- Indica all'assemblatore l'indirizzo di partenza dove inserire le istruzioni e i dati definiti nelle righe seguenti
- La sua sintassi nel nostro linguaggio generico è la seguente:

**ORIGIN**

*Indirizzo\_di\_memoria*

- Alle istruzioni e ai dati seguenti la direttiva ORIGIN verranno assengati gli indirizzi a partire dall'indirizzo *Indirizzo\_di\_memoria*

- Indica all'assemblatore di riservare uno spazio di memoria espresso in byte
- La sua sintassi nel nostro linguaggio generico è la seguente:

**RESERVE**

*Spazio\_in\_byte*

- La locazione di memoria riservata non viene inizializzata

- Indica all'assemblatore di riservare una parola di memoria e le assegna un contenuto
- La sua sintassi nel nostro linguaggio generico è la seguente:

**DATAWORD**

*Contenuto\_da\_assegnare*

- La parola di memoria viene inizializzata con il valore *Contenuto\_da\_assegnare*

# Linea di codice assemblativo

- Nel maggiore dei casi, una linea di codice assemblativo presenta i seguenti campi:

***Etichetta      Operazione      Operandi      Commento***

- **Etichetta:** Nome che viene associato all'indirizzo della parola di memoria assegnata all'istruzione o all'indirizzo del blocco di memoria riservato. È facoltativa
- **Operazione:** Il nome (codice operativo) dell'istruzione oppure una direttiva di assemblatore.
- **Operandi:** Informazione di indirizzamento per accedere agli operandi
- **Commento:** Testo di commento, ignorato dall'assemblatore

# Esempio

## Memoria

	100	Load	R2, N
	104	Clear	R3
	108	Move	R4, #NUM1
CICLO	112	Load	R5, (R4)
	116	Add	R3, R3, R5
	120	Add	R4, R4, #4
	124	Subtract	R2, R2, #1
	128	Branch_if_[R2]>0	CICLO
	132	Store	R3, SOMMA
	...	...	
	200		
SOMMA	204	150	
N	208		
NUM1	212		
NUM2	...		
NUMn	804		

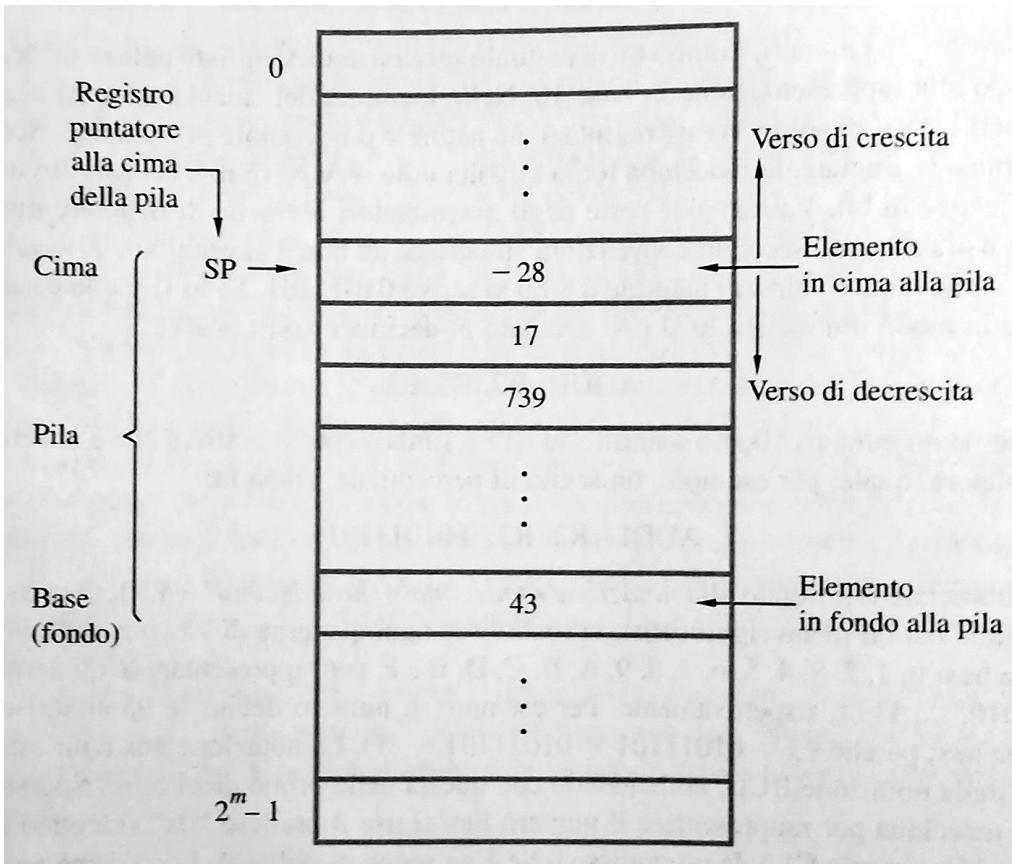
Etichetta	Operazione	Operandi
CICLO:	ORIGIN	100
	Load	R2, N
	Clear	R3
	Move	R4, #NUM1
	Load	R5, (R4)
	Add	R3, R3, R5
	Add	R4, R4, #4
	Subtract	R2, R2, #1
SOMMA:	Branch_if_[R2]>0	CICLO
	Store	R3, SOMMA
	...	...
	...	...
N:	ORIGIN	200
	RESERVE	4
	DATAWORD	150
	RESERVE	600
NUM1:		END

# Notazione numeri

- L'assemblatore ci permette di denotare i numeri in diversi formati: binario, decimale, esadecimale
- Per indicare quale rappresentazione si vuole usare si usano dei prefissi:
  - Binaria: %
  - Decimale: nessun prefisso
  - Esadecimale: 0x
- Esempio su come usare un operando della somma in modo immediato:
  - Binario: Add R2, R3, **93**
  - Decimale: Add R2, R3, **%01011101**
  - Esadecimale: Add R2, R3, **0x5D**

# Pila (Stack)

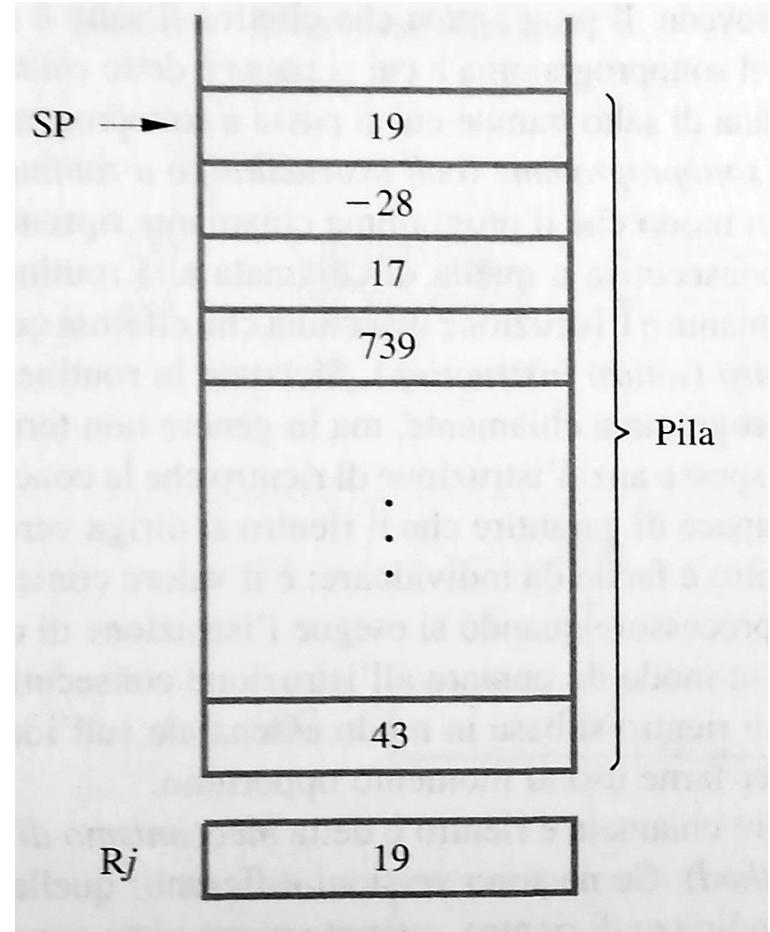
- Lista di elementi (parole) dove si immaginano i dati posizionati uno sull'altro
- Gli elementi possono essere solo aggiunti e prelevati dalla cima della pila: l'ultimo elemento inserito è il primo ad essere prelevato (Last In First Out – LIFO)
- **Stack Pointer – SP:** registro che punta alla cima della pila
- Gli elementi della pila hanno indirizzi in ordine **decrescente** dalla base alla cima



# Push (Impila)

- L'operazione di **Push** aggiunge un elemento in cima alla pila
- In un architettura RISC si realizza con due istruzioni:
  - Diminuire l'indirizzo contenuto in SP di una parola per puntare alla nuova cima
  - Scrivere il valore richiesto nella parola puntata da SP

**Subtract SP, SP, #4**  
**Store Rj, (SP)**

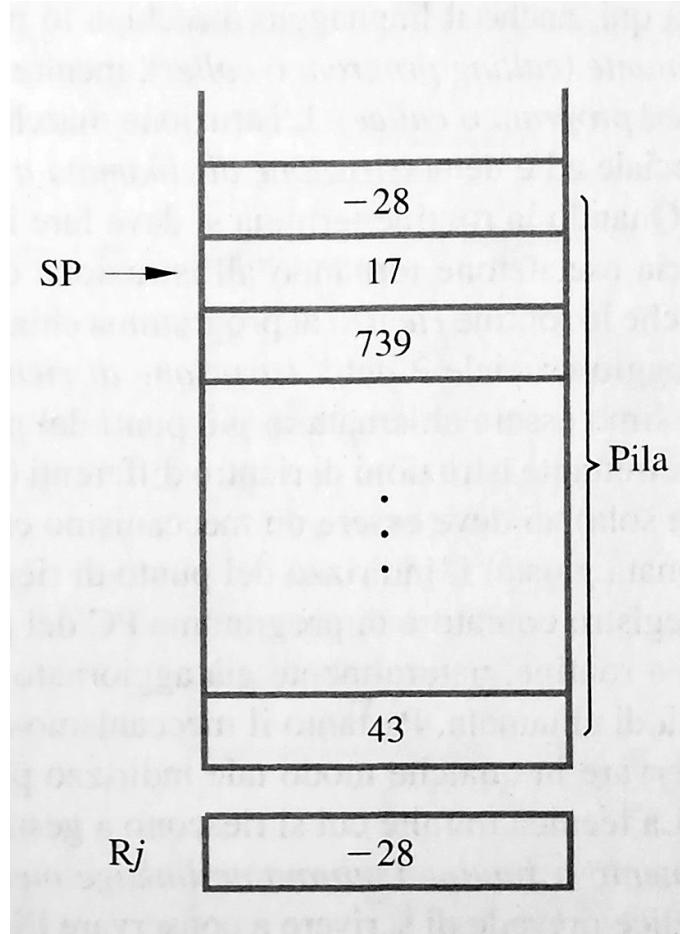


# Pop (Spila)

- L'operazione di **Pop** preleva un elemento dalla cima della pila
- In un architettura RISC si realizza con due istruzioni:
  - Copiare il valore contenuto nella locazione di memoria puntata da SP in un registro del processore
  - Aumentare l'indirizzo contenuto in SP di una parola per puntare alla nuova cima

**Load Rj, (SP)**

**Add SP, SP, #4**



# Sottoprogramma

- Un **sottoprogramma o routine** è una lista di istruzioni che eseguono un compito specifico e che possono essere richiamate in un qualsiasi momento durante l'esecuzione di un programma
- Per richiamare una routine, un programma usa una funzione di salto particolare detta **Call instruction**
- Per ritornare dalla routine all'istruzione successiva alla Call nel programma principale si usa una funzione di salto particolare detta **Return instruction**
- Il **Link Register** è un registro speciale in cui si memorizza l'indirizzo dell'istruzione di rientro durante l'esecuzione di un sottoprogramma

# Chiamata a sottoprogramma

- L'operazione di chiamata a sottoprogramma presenta questa sintassi:

**Call    INDIRIZZO**

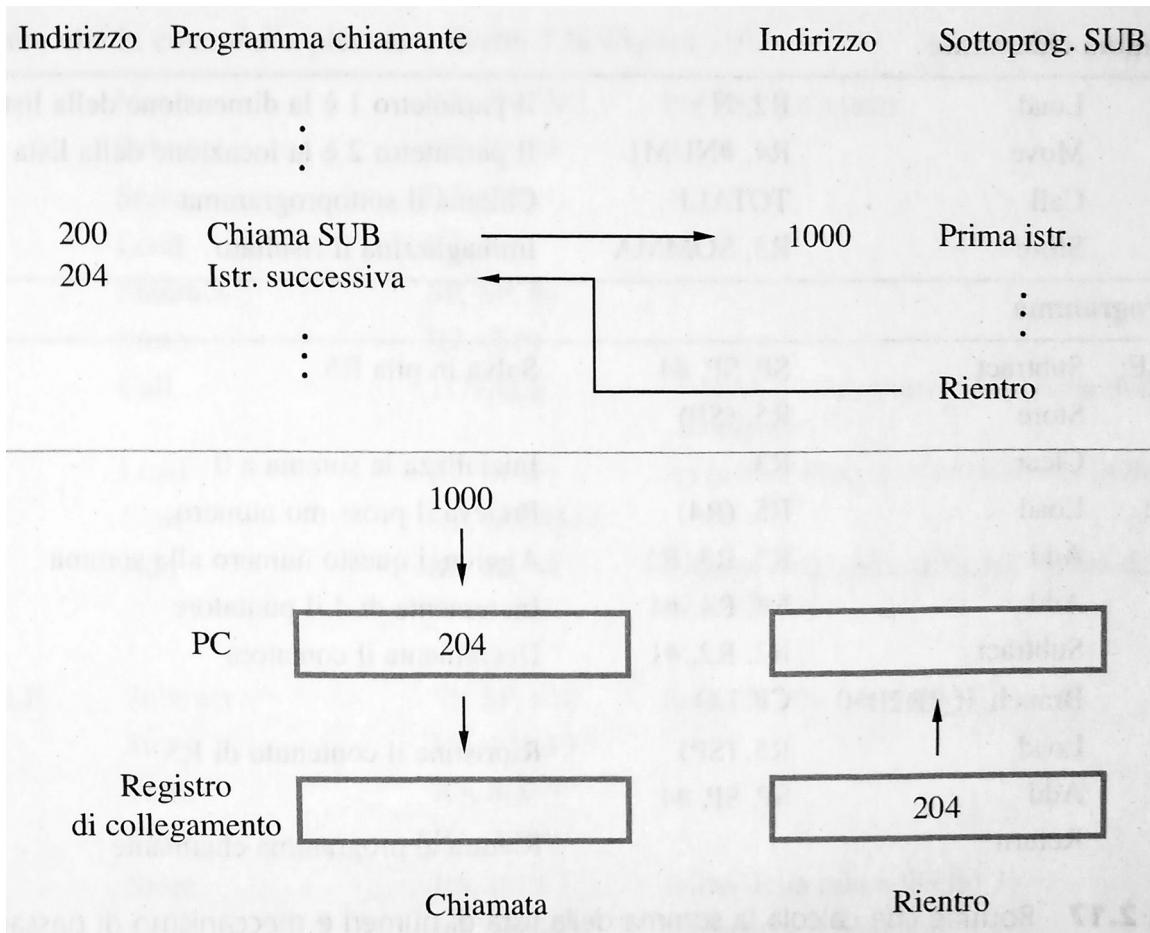
- L'operazione di chiamata esegue due passi:
  - 1) Salva il contenuto del registro **PC** nel **Link Register**
  - 2) Salta all'indirizzo di destinazione indicato nell'istruzione di chiamata

- L'istruzione di rientro da sottoprogramma presenta questa sintassi:

## Return

- L'istruzione di rientro salta all'indirizzo di rientro contenuto nel Link Register nel seguente modo:
  - 1) Salva il contenuto del **Link Register** nel registro **PC**

# Meccanismo di collegamento a routine



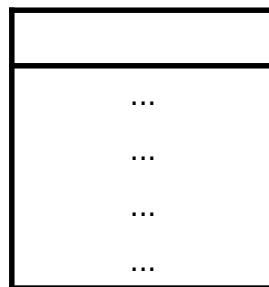
# Passaggio di parametri

- Una routine ha spesso bisogno di:
  - Parametri di ingresso su cui operare
  - Restituire un risultato al programma chiamante
- Esistono 2 tecniche di **passaggio di parametri**:
  - 1) Passaggio tramite registri del processore (si usano alcuni registri generici per salvare i parametri). Numero di parametri limitato dal numero di registri
  - 2) Passaggio attraverso la pila (si impilano i parametri nella pila). Numero di parametri virtualmente illimitato

# Esempio passaggio tramite registri

PC 

LINK\_reg 

SP -> 

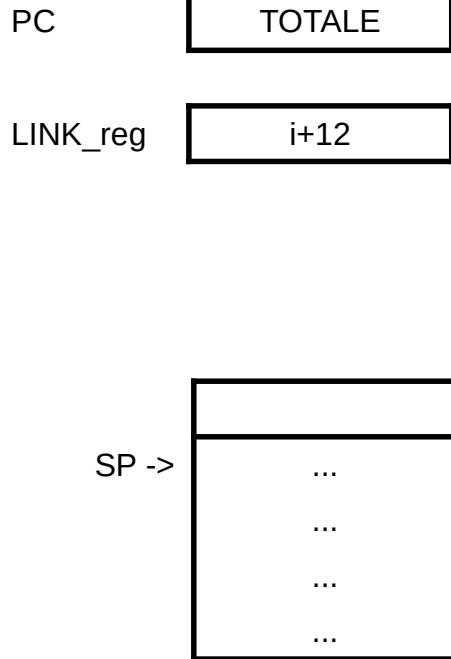
## Programma chiamante

Load	R2, N	Il parametro 1 è la dimensione della lista
Move	R4, #NUM1	Il parametro 2 è la locazione della lista
Call	TOTALE	Chiama il sottoprogramma
Store	R3, SOMMA	Immagazzina il risultato

## Sottoprogramma

TOTALE:	Subtract	SP, SP, #4	Salva in pila R5
	Store	R5, (SP)	
	Clear	R3	Inizializza la somma a 0
CICLO:	Load	R5, (R4)	Preleva il prossimo numero
	Add	R3, R3, R5	Aggiungi questo numero alla somma
	Add	R4, R4, #4	Incrementa di 4 il puntatore
	Subtract	R2, R2, #1	Decrementa il contatore
	Branch_if_[R2]>0	CICLO	
	Load	R5, (SP)	Ripristina il contenuto di R5
	Add	SP, SP, #4	
	Return		Rientra al programma chiamante

# Esempio passaggio tramite registri

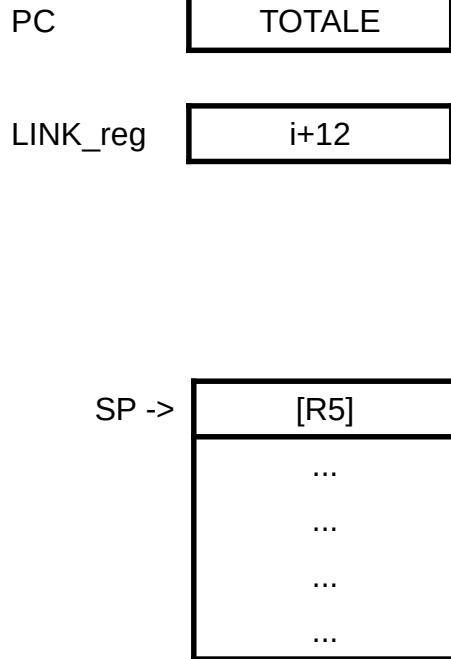


Programma chiamante			
	Load	R2, N	Il parametro 1 è la dimensione della lista
	Move	R4, #NUM1	Il parametro 2 è la locazione della lista
→	Call	TOTALE	Chiama il sottoprogramma
	Store	R3, SOMMA	Immagazzina il risultato

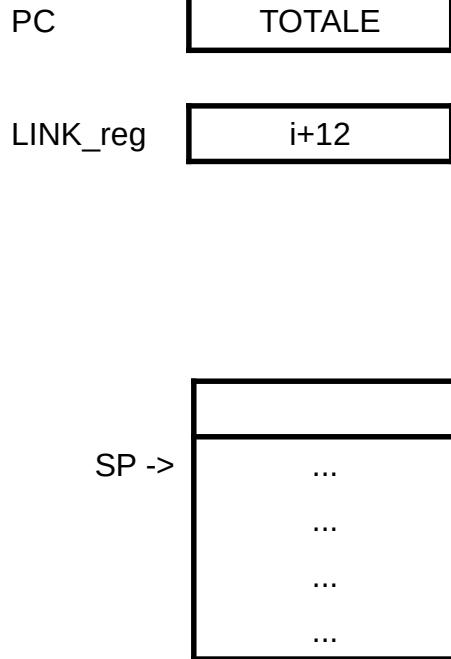
Sottoprogramma			
TOTALE:	Subtract	SP, SP, #4	Salva in pila R5
	Store	R5, (SP)	
	Clear	R3	Inizializza la somma a 0
CICLO:	Load	R5, (R4)	Preleva il prossimo numero
	Add	R3, R3, R5	Aggiungi questo numero alla somma
	Add	R4, R4, #4	Incrementa di 4 il puntatore
	Subtract	R2, R2, #1	Decrementa il contatore
	Branch_if_[R2]>0	CICLO	
	Load	R5, (SP)	Ripristina il contenuto di R5
	Add	SP, SP, #4	
	Return		Rientra al programma chiamante

# Esempio passaggio tramite registri



Programma chiamante			
Load	R2, N	Il parametro 1 è la dimensione della lista	
Move	R4, #NUM1	Il parametro 2 è la locazione della lista	
Call	TOTALE	Chiama il sottoprogramma	
Store	R3, SOMMA	Immagazzina il risultato	
Sottoprogramma			
TOTALE:	Subtract SP, SP, #4	Salva in pila R5	
	Store R5, (SP)		
	Clear R3	Inizializza la somma a 0	
CICLO:	Load R5, (R4)	Preleva il prossimo numero	
	Add R3, R3, R5	Aggiungi questo numero alla somma	
	Add R4, R4, #4	Incrementa di 4 il puntatore	
	Subtract R2, R2, #1	Decrementa il contatore	
	Branch_if_[R2]>0 CICLO		
	Load R5, (SP)	Ripristina il contenuto di R5	
	Add SP, SP, #4		
	Return	Rientra al programma chiamante	

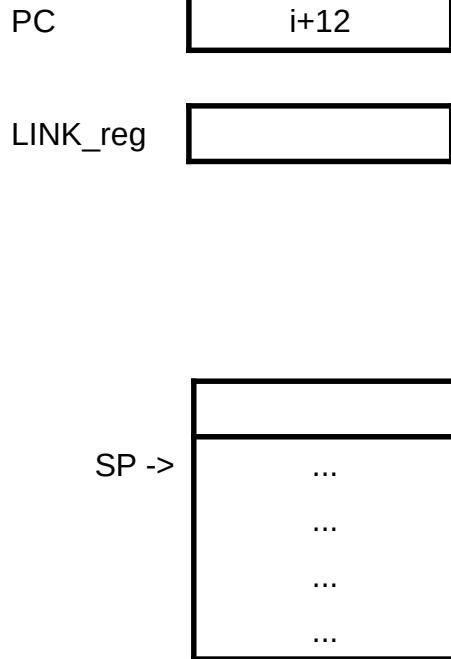
# Esempio passaggio tramite registri



Programma chiamante			
Load	R2, N	Il parametro 1 è la dimensione della lista	
Move	R4, #NUM1	Il parametro 2 è la locazione della lista	
Call	TOTALE	Chiama il sottoprogramma	
Store	R3, SOMMA	Immagazzina il risultato	
Sottoprogramma			
TOTALE:	Subtract	SP, SP, #4	Salva in pila R5
	Store	R5, (SP)	
	Clear	R3	Inizializza la somma a 0
CICLO:	Load	R5, (R4)	Preleva il prossimo numero
	Add	R3, R3, R5	Aggiungi questo numero alla somma
	Add	R4, R4, #4	Incrementa di 4 il puntatore
	Subtract	R2, R2, #1	Decrementa il contatore
	Branch_if_[R2]>0	CICLO	
	Load	R5, (SP)	Ripristina il contenuto di R5
	Add	SP, SP, #4	
	Return		Rientra al programma chiamante



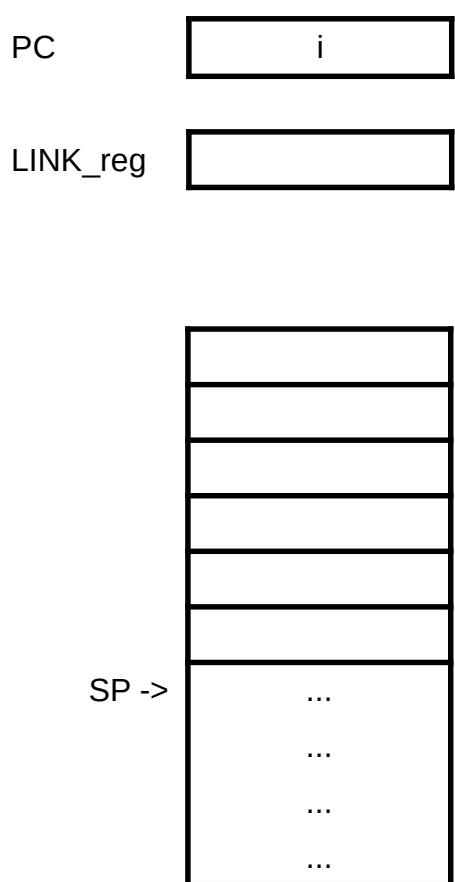
# Esempio passaggio tramite registri



Programma chiamante			
Load	R2, N	Il parametro 1 è la dimensione della lista	
Move	R4, #NUM1	Il parametro 2 è la locazione della lista	
Call	TOTALE	Chiama il sottoprogramma	
Store	R3, SOMMA	Immagazzina il risultato	
Sottoprogramma			
TOTALE:	Subtract	SP, SP, #4	Salva in pila R5
	Store	R5, (SP)	
	Clear	R3	Inizializza la somma a 0
CICLO:	Load	R5, (R4)	Preleva il prossimo numero
	Add	R3, R3, R5	Aggiungi questo numero alla somma
	Add	R4, R4, #4	Incrementa di 4 il puntatore
	Subtract	R2, R2, #1	Decrementa il contatore
	Branch_if_[R2]>0	CICLO	
	Load	R5, (SP)	Ripristina il contenuto di R5
	Add	SP, SP, #4	
	Return		Rientra al programma chiamante



# Esempio passaggio tramite pila



	Move	R2, #NUM1	Impila i parametri
	Subtract	SP, SP, #4	
	Store	R2, (SP)	
	Load	R2, N	
	Subtract	SP, SP, #4	
	Store	R2, (SP)	
	Call	TOTALE	Chiama il sottoprogramma (cima della pila a livello 2)
	Load	R2, 4(SP)	Spila il risultato e conservalo in SOMMA
	Store	R2, SOMMA	
	Add	SP, SP, #8	Ripristina la cima della pila (cima della pila a livello 1)
	:		
	TOTALE :	Subtract	Salva in pila i registri
		SP, SP, #16	
		Store	R2, 12(SP)
		Store	R3, 8(SP)
		Store	R4, 4(SP)
		Store	R5, (SP) (cima della pila a livello 3)
		Load	R2, 16(SP) Inizializza il contatore a <i>n</i>
		Load	R4, 20(SP) Inizializza il puntatore alla lista
		Clear	R3 Inizializza la somma a 0
	CICLO:	Load	R5, (R4) Preleva il prossimo numero
		Add	R3, R3, R5 Aggiungi questo numero alla somma
		Add	R4, R4, #4 Incrementa di 4 il puntatore
		Subtract	R2, R2, #1 Decrementa il contatore
		Branch_if_[R2]>0	CICLO
		Store	R3, 20(SP) Impila il risultato
		Load	R5, (SP) Ripristina i registri
		Load	R4, 4(SP)
		Load	R3, 8(SP)
		Load	R2, 12(SP)
		Add	SP, SP, #16 (cima della pila a livello 2)
		Return	Rientra al programma chiamante

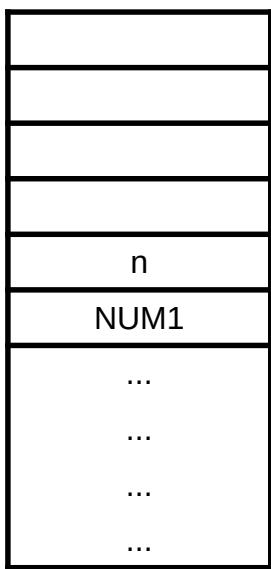
# Esempio passaggio tramite pila

PC

i+22

LINK\_reg

SP ->



Move	R2, #NUM1	Impila i parametri
Subtract	SP, SP, #4	
Store	R2, (SP)	
Load	R2, N	
Subtract	SP, SP, #4	
Store	R2, (SP)	
Call	TOTALE	Chiama il sottoprogramma (cima della pila a livello 2)
Load	R2, 4(SP)	Spila il risultato e conservalo in SOMMA
Store	R2, SOMMA	
Add	SP, SP, #8	Ripristina la cima della pila (cima della pila a livello 1)
:		
TOTALE :	Subtract	SP, SP, #16
	Store	R2, 12(SP)
	Store	R3, 8(SP)
	Store	R4, 4(SP)
	Store	R5, (SP)
	Load	R2, 16(SP)
	Load	R4, 20(SP)
	Clear	R3
CICLO:	Load	R5, (R4)
	Add	R3, R3, R5
	Add	R4, R4, #4
	Subtract	R2, R2, #1
	Branch_if_[R2]>0	CICLO
	Store	R3, 20(SP)
	Load	R5, (SP)
	Load	R4, 4(SP)
	Load	R3, 8(SP)
	Load	R2, 12(SP)
	Add	SP, SP, #16
	Return	(cima della pila a livello 2)
		Rientra al programma chiamante

# Esempio passaggio tramite pila

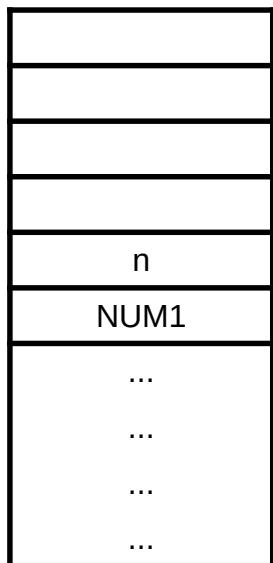
PC

TOTALE

LINK\_reg

i+24

SP ->



	Move	R2, #NUM1	Impila i parametri
	Subtract	SP, SP, #4	
	Store	R2, (SP)	
	Load	R2, N	
	Subtract	SP, SP, #4	
	Store	R2, (SP)	
	Call	TOTALE	Chiama il sottoprogramma (cima della pila a livello 2)
	Load	R2, 4(SP)	Spila il risultato e conservalo in SOMMA
	Store	R2, SOMMA	
	Add	SP, SP, #8	Ripristina la cima della pila (cima della pila a livello 1)
	:		
TOTALE :	Subtract	SP, SP, #16	Salva in pila i registri
	Store	R2, 12(SP)	
	Store	R3, 8(SP)	
	Store	R4, 4(SP)	
	Store	R5, (SP)	(cima della pila a livello 3)
	Load	R2, 16(SP)	Inizializza il contatore a <i>n</i>
	Load	R4, 20(SP)	Inizializza il puntatore alla lista
	Clear	R3	Inizializza la somma a 0
CICLO:	Load	R5, (R4)	Preleva il prossimo numero
	Add	R3, R3, R5	Aggiungi questo numero alla somma
	Add	R4, R4, #4	Incrementa di 4 il puntatore
	Subtract	R2, R2, #1	Decrementa il contatore
	Branch_if_[R2]>0	CICLO	
	Store	R3, 20(SP)	Impila il risultato
	Load	R5, (SP)	Ripristina i registri
	Load	R4, 4(SP)	
	Load	R3, 8(SP)	
	Load	R2, 12(SP)	
	Add	SP, SP, #16	(cima della pila a livello 2)
	Return		Rientra al programma chiamante

# Esempio passaggio tramite pila

PC

TOTALE+20

LINK\_reg

i+24

SP ->

[R5]
[R4]
[R3]
[R2]
n
NUM1
...
...
...
...

Move	R2, #NUM1	Impila i parametri
Subtract	SP, SP, #4	
Store	R2, (SP)	
Load	R2, N	
Subtract	SP, SP, #4	
Store	R2, (SP)	
Call	TOTALE	Chiama il sottoprogramma (cima della pila a livello 2)
Load	R2, 4(SP)	Spila il risultato e conservalo in SOMMA
Store	R2, SOMMA	
Add	SP, SP, #8	Ripristina la cima della pila (cima della pila a livello 1)
:		
TOTALE :	Subtract	SP, SP, #16
	Store	R2, 12(SP)
	Store	R3, 8(SP)
	Store	R4, 4(SP)
	Store	R5, (SP)
	Load	R2, 16(SP)
	Load	R4, 20(SP)
	Clear	R3
CICLO:	Load	R5, (R4)
	Add	R3, R3, R5
	Add	R4, R4, #4
	Subtract	R2, R2, #1
	Branch_if_[R2]>0	CICLO
	Store	R3, 20(SP)
	Load	R5, (SP)
	Load	R4, 4(SP)
	Load	R3, 8(SP)
	Load	R2, 12(SP)
	Add	SP, SP, #16
	Return	(cima della pila a livello 2)
		Rientra al programma chiamante



# Esempio passaggio tramite pila

PC

TOTALE+56

LINK\_reg

i+24

SP ->

[R5]
[R4]
[R3]
[R2]
n
[R3] da TOTALE
...
...
...
...

Move	R2, #NUM1	Impila i parametri
Subtract	SP, SP, #4	
Store	R2, (SP)	
Load	R2, N	
Subtract	SP, SP, #4	
Store	R2, (SP)	
Call	TOTALE	Chiama il sottoprogramma (cima della pila a livello 2)
Load	R2, 4(SP)	Spila il risultato e conservalo in SOMMA
Store	R2, SOMMA	
Add	SP, SP, #8	Ripristina la cima della pila (cima della pila a livello 1)
:		
TOTALE :	Subtract	SP, SP, #16
	Store	R2, 12(SP)
	Store	R3, 8(SP)
	Store	R4, 4(SP)
	Store	R5, (SP)
	Load	R2, 16(SP)
	Load	R4, 20(SP)
	Clear	R3
CICLO:	Load	R5, (R4)
	Add	R3, R3, R5
	Add	R4, R4, #4
	Subtract	R2, R2, #1
	Branch_if_[R2]>0	CICLO
	Store	R3, 20(SP)
	Load	R5, (SP)
	Load	R4, 4(SP)
	Load	R3, 8(SP)
	Load	R2, 12(SP)
	Add	SP, SP, #16
	Return	(cima della pila a livello 2)
		Rientra al programma chiamante



# Esempio passaggio tramite pila

PC

TOTALE+76

LINK\_reg

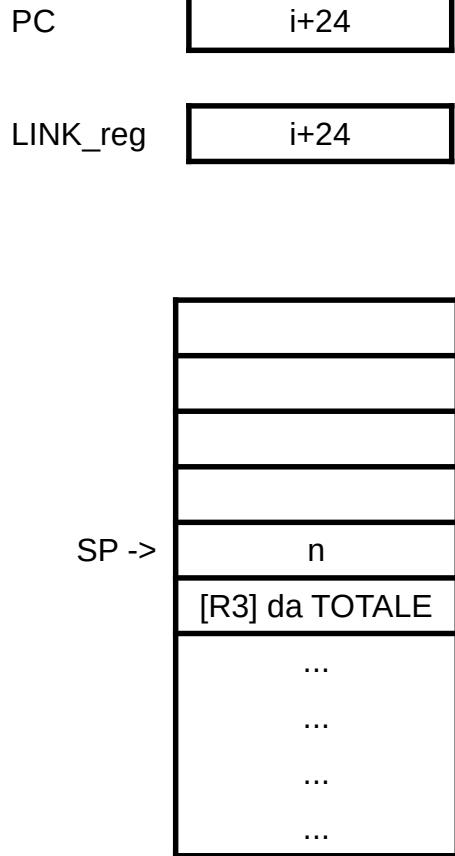
i+24

SP ->



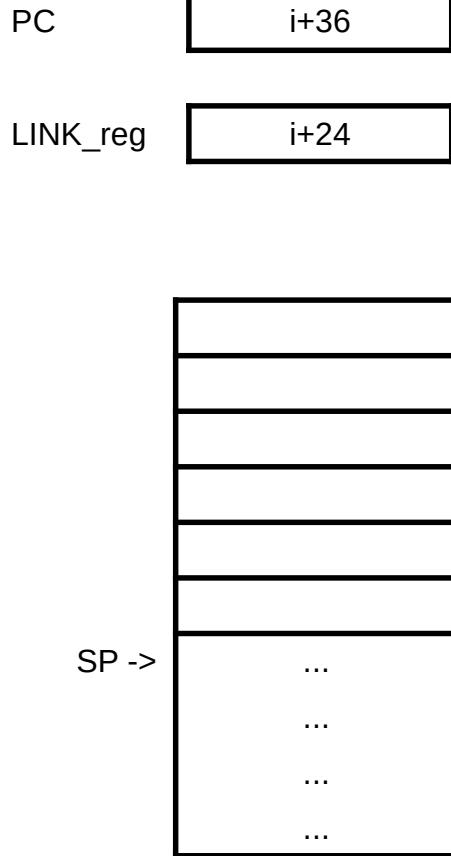
	Move	R2, #NUM1	Impila i parametri
	Subtract	SP, SP, #4	
	Store	R2, (SP)	
	Load	R2, N	
	Subtract	SP, SP, #4	
	Store	R2, (SP)	
	Call	TOTALE	Chiama il sottoprogramma (cima della pila a livello 2)
	Load	R2, 4(SP)	Spila il risultato e conservalo in SOMMA
	Store	R2, SOMMA	
	Add	SP, SP, #8	Ripristina la cima della pila (cima della pila a livello 1)
	:		
TOTALE :	Subtract	SP, SP, #16	Salva in pila i registri
	Store	R2, 12(SP)	
	Store	R3, 8(SP)	
	Store	R4, 4(SP)	
	Store	R5, (SP)	(cima della pila a livello 3)
	Load	R2, 16(SP)	Inizializza il contatore a <i>n</i>
	Load	R4, 20(SP)	Inizializza il puntatore alla lista
	Clear	R3	Inizializza la somma a 0
CICLO:	Load	R5, (R4)	Preleva il prossimo numero
	Add	R3, R3, R5	Aggiungi questo numero alla somma
	Add	R4, R4, #4	Incrementa di 4 il puntatore
	Subtract	R2, R2, #1	Decrementa il contatore
	Branch_if_[R2]>0	CICLO	
	Store	R3, 20(SP)	Impila il risultato
	Load	R5, (SP)	Ripristina i registri
	Load	R4, 4(SP)	
	Load	R3, 8(SP)	
	Load	R2, 12(SP)	
	Add	SP, SP, #16	(cima della pila a livello 2)
	Return		Rientra al programma chiamante

# Esempio passaggio tramite pila



	Move	R2, #NUM1	Impila i parametri
	Subtract	SP, SP, #4	
	Store	R2, (SP)	
	Load	R2, N	
	Subtract	SP, SP, #4	
	Store	R2, (SP)	
	Call	TOTALE	Chiama il sottoprogramma (cima della pila a livello 2)
	Load	R2, 4(SP)	Spila il risultato e conservalo in SOMMA
	Store	R2, SOMMA	
	Add	SP, SP, #8	Ripristina la cima della pila (cima della pila a livello 1)
	:		
	TOTALE :	Subtract	Salva in pila i registri
		SP, SP, #16	
		Store	R2, 12(SP)
		Store	R3, 8(SP)
		Store	R4, 4(SP)
		Store	R5, (SP)
		Load	R2, 16(SP)
		Load	R4, 20(SP)
		Clear	Inizializza il contatore a 0
	CICLO:	Load	Inizializza il puntatore alla lista
		R3	Inizializza la somma a 0
		Load	Preleva il prossimo numero
		Add	Aggiungi questo numero alla somma
		Add	Incrementa di 4 il puntatore
		Subtract	Decrementa il contatore
		Branch_if_[R2]>0	CICLO
		Store	Impila il risultato
		Load	Ripristina i registri
		Load	
		Return	(cima della pila a livello 2)
			Rientra al programma chiamante

# Esempio passaggio tramite pila

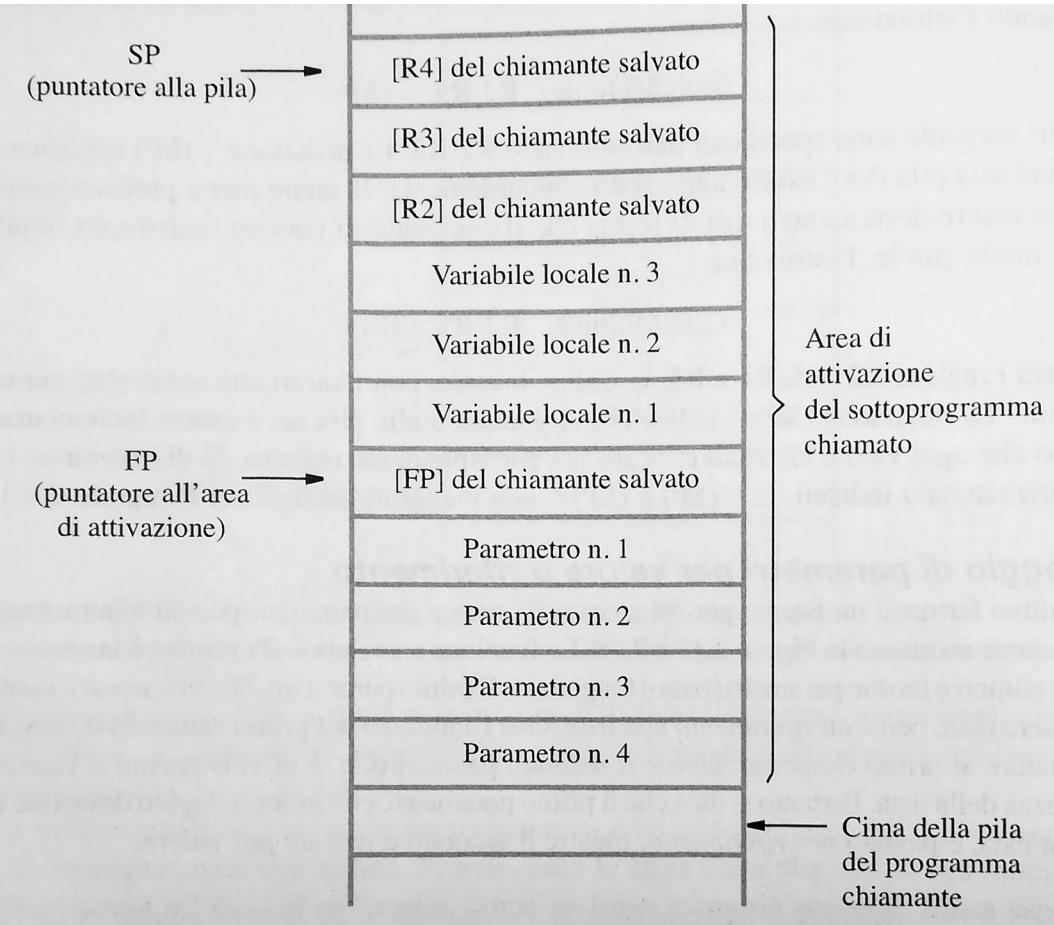


→

Move	R2, #NUM1	Impila i parametri
Subtract	SP, SP, #4	
Store	R2, (SP)	
Load	R2, N	
Subtract	SP, SP, #4	
Store	R2, (SP)	
Call	TOTALE	Chiama il sottoprogramma (cima della pila a livello 2)
Load	R2, 4(SP)	Spila il risultato e conservalo in SOMMA
Store	R2, SOMMA	
Add	SP, SP, #8	Ripristina la cima della pila (cima della pila a livello 1)
:		
TOTALE :	Subtract	SP, SP, #16
	Store	R2, 12(SP)
	Store	R3, 8(SP)
	Store	R4, 4(SP)
	Store	R5, (SP)
	Load	R2, 16(SP)
	Load	R4, 20(SP)
	Clear	R3
CICLO:	Load	R5, (R4)
	Add	R3, R3, R5
	Add	R4, R4, #4
	Subtract	R2, R2, #1
	Branch_if_[R2]>0	CICLO
	Store	R3, 20(SP)
	Load	R5, (SP)
	Load	R4, 4(SP)
	Load	R3, 8(SP)
	Load	R2, 12(SP)
	Add	SP, SP, #16
	Return	(cima della pila a livello 2)
		Rientra al programma chiamante

# Area di attivazione in pila (stack frame)

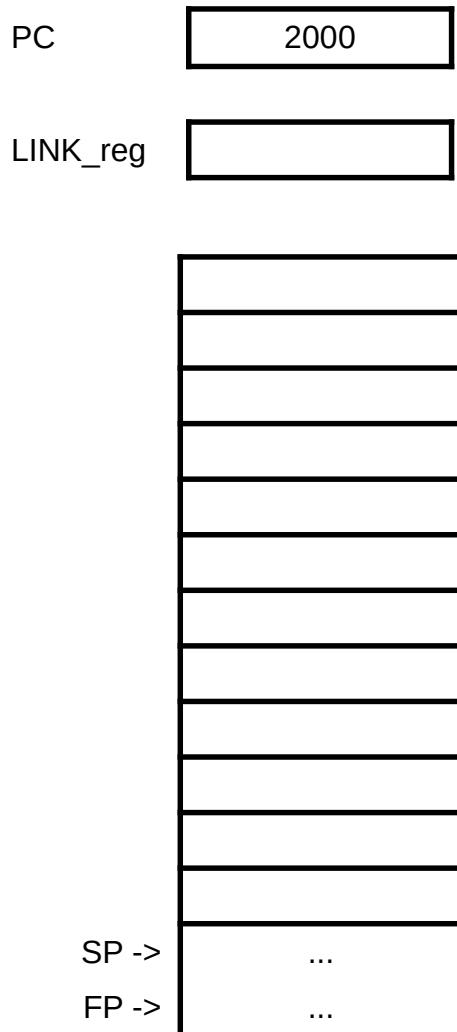
- Il blocco di memoria nella pila riservato al sottoprogramma è chiamato **Area di attivazione (Stack Frame)**
- Il **Frame Pointer FP** è un registro che punta allo Stack Frame del sottoprogramma in esecuzione
- Lo Stack Frame contiene i parametri, il FP del programma chiamante, le variabili locali e valori di registri salvati
- Il FP punta alla parola dove è memorizzato il FP del programma chiamante



# Annidamento di sottoprogramma

- Nel caso si abbiano diversi sottoprogrammi annidati, prima di chiamare una seconda routine è necessario salvare il contenuto del registro LINK\_reg per recuperarlo in seguito
- Si può usare la pila per salvare gli indirizzi di rientro delle chiamate annidate all'interno dell'area di attivazione dei programmi chiamanti
- Il LINK\_reg è comunque sempre usato dalle funzioni di Call e Return
- Alcune architetture non usano il LINK\_reg, ma salvano l'indirizzo di rientro solo nella pila. Questa strategia è chiamata **pila a modo implicito**

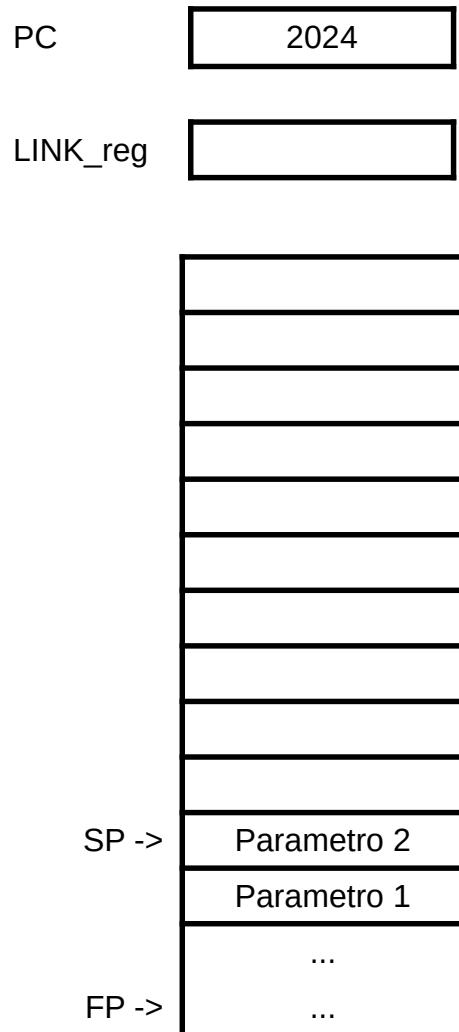
# Esempio sottoprogrammi annidati



Locazione di memoria		Istruzioni		Commenti
<b>Programma principale</b>				
			:	
2000	PRO:	Load	R2, PARAM2	Impila i parametri
2004		Subtract	SP, SP, #4	
2008		Store	R2, (SP)	
2012		Load	R2, PARAM1	
2016		Subtract	SP, SP, #4	
2020		Store	R2, (SP)	
2024		Call	SUB1	Chiama il sottoprogramma
2028		Load	R2, (SP)	Immagazzina il risultato
2032		Store	R2, RIS	
2036		Add	SP, SP, #8	Ripristina il livello della pila
2040			prossima istruzione	
			:	
<b>Primo sottoprogramma</b>				
2100	SUB1:	Subtract	SP, SP, #24	Salva in pila i registri
2104		Store	LINK_reg,20(SP)	
2108		Store	FP, 16(SP)	
2112		Store	R2, 12(SP)	
2116		Store	R3, 8(SP)	
2120		Store	R4, 4(SP)	
2124		Store	R5, (SP)	
2128		Add	FP, SP, #16	Inizializza il puntatore all'area di attivazione
2132		Load	R2, 8(FP)	Preleva il primo parametro
2136		Load	R3, 12(FP)	Preleva il secondo parametro
		:		
		Load	R4, PARAM3	Impila un parametro da passare a SUB2
		Subtract	SP, SP, #4	
		Store	R4, (SP)	
		Call	SUB2	
		Load	R4, (SP)	Spila il risultato ottenuto da SUB2
		Add	SP, SP, #4	
		:		
		Store	R5, 8(FP)	Impila la risposta
		Load	R5, (SP)	Ripristina i registri
		Load	R4, 4(SP)	
		Load	R3, 8(SP)	
		Load	R2, 12(SP)	
		Load	FP, 16(SP)	
		Load	LINK_reg,20(SP)	
		Add	SP, SP, #24	
		Return		Rientro al programma principale

Continua nella parte b

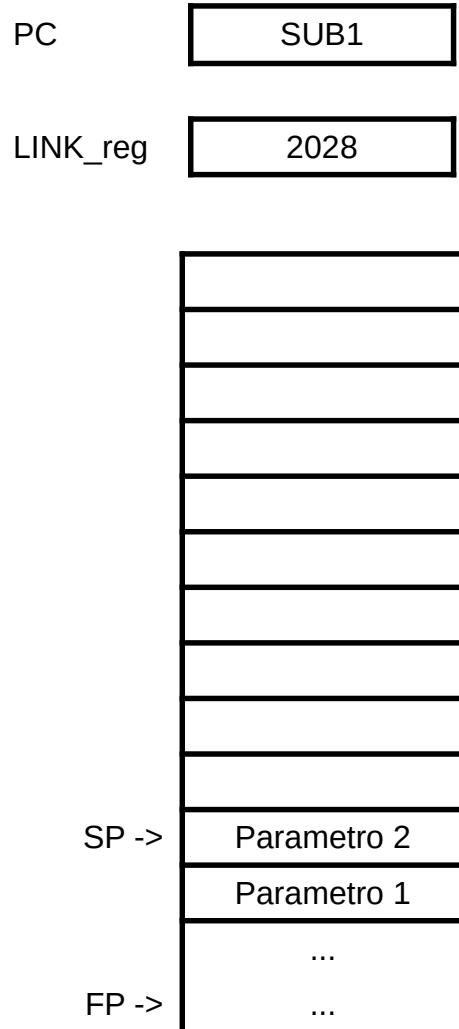
# Esempio sottoprogrammi annidati



Locazione di memoria		Istruzioni		Commenti
Programma principale				
		:		
2000	PRO:	Load R2, PARAM2		Impila i parametri
2004		Subtract SP, SP, #4		
2008		Store R2, (SP)		
2012		Load R2, PARAM1		
2016		Subtract SP, SP, #4		
2020		Store R2, (SP)		
2024		Call SUB1		Chiama il sottoprogramma
2028		Load R2, (SP)		Immagazzina il risultato
2032		Store R2, RIS		
2036		Add SP, SP, #8		Ripristina il livello della pila
2040		prossima istruzione		
		:		
Primo sottoprogramma				
2100	SUB1:	Subtract SP, SP, #24		Salva in pila i registri
2104		Store LINK_reg,20(SP)		
2108		Store FP, 16(SP)		
2112		Store R2, 12(SP)		
2116		Store R3, 8(SP)		
2120		Store R4, 4(SP)		
2124		Store R5, (SP)		
2128		Add FP, SP, #16		Inizializza il puntatore all'area di attivazione
2132		Load R2, 8(FP)		Preleva il primo parametro
2136		Load R3, 12(FP)		Preleva il secondo parametro
		:		
		Load R4, PARAM3		Impila un parametro da passare a SUB2
		Subtract SP, SP, #4		
		Store R4, (SP)		
		Call SUB2		
		Load R4, (SP)		Spila il risultato ottenuto da SUB2
		Add SP, SP, #4		
		:		
		Store R5, 8(FP)		Impila la risposta
		Load R5, (SP)		Ripristina i registri
		Load R4, 4(SP)		
		Load R3, 8(SP)		
		Load R2, 12(SP)		
		Load FP, 16(SP)		
		Load LINK_reg,20(SP)		
		Add SP, SP, #24		
		Return		Rientro al programma principale

Continua nella parte b

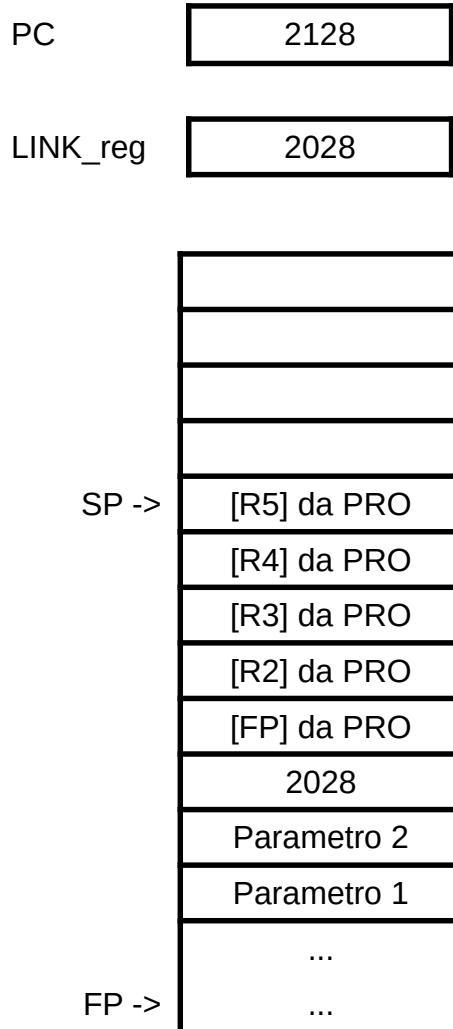
# Esempio sottoprogrammi annidati



Locazione di memoria		Istruzioni	Commenti
<b>Programma principale</b>			
2000	PRO:	Load R2, PARAM2	Impila i parametri
2004		Subtract SP, SP, #4	
2008		Store R2, (SP)	
2012		Load R2, PARAM1	
2016		Subtract SP, SP, #4	
2020		Store R2, (SP)	
2024		Call SUB1	Chiama il sottoprogramma
2028		Load R2, (SP)	Immagazzina il risultato
2032		Store R2, RIS	
2036		Add SP, SP, #8	Ripristina il livello della pila
2040		prossima istruzione	
:			
<b>Primo sottoprogramma</b>			
2100	SUB1:	Subtract SP, SP, #24	Salva in pila i registri
2104		Store LINK_reg,20(SP)	
2108		Store FP, 16(SP)	
2112		Store R2, 12(SP)	
2116		Store R3, 8(SP)	
2120		Store R4, 4(SP)	
2124		Store R5, (SP)	
2128		Add FP, SP, #16	Inizializza il puntatore all'area di attivazione
2132		Load R2, 8(FP)	Preleva il primo parametro
2136		Load R3, 12(FP)	Preleva il secondo parametro
:			
		Load R4, PARAM3	Impila un parametro da passare a SUB2
		Subtract SP, SP, #4	
		Store R4, (SP)	
		Call SUB2	
		Load R4, (SP)	Spila il risultato ottenuto da SUB2
		Add SP, SP, #4	
:			
		Store R5, 8(FP)	Impila la risposta
		Load R5, (SP)	Ripristina i registri
		Load R4, 4(SP)	
		Load R3, 8(SP)	
		Load R2, 12(SP)	
		Load FP, 16(SP)	
		Load LINK_reg,20(SP)	
		Add SP, SP, #24	
		Return	Rientro al programma principale

Continua nella parte b

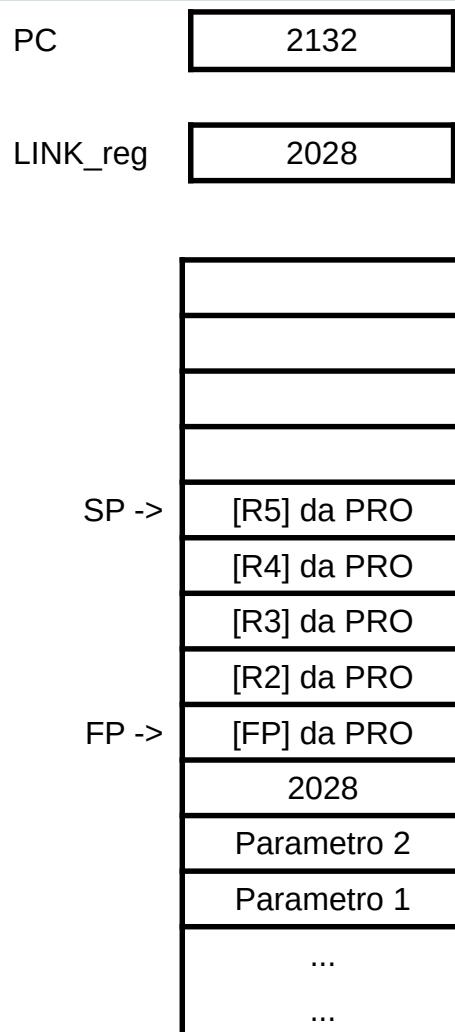
# Esempio sottoprogrammi annidati



Locazione di memoria		Istruzioni		Commenti
<b>Programma principale</b>				
		:		
2000	PRO:	Load R2, PARAM2		Impila i parametri
2004		Subtract SP, SP, #4		
2008		Store R2, (SP)		
2012		Load R2, PARAM1		
2016		Subtract SP, SP, #4		
2020		Store R2, (SP)		
2024		Call SUB1		Chiama il sottoprogramma
2028		Load R2, (SP)		Immagazzina il risultato
2032		Store R2, RIS		
2036		Add SP, SP, #8		Ripristina il livello della pila
2040		prossima istruzione		
		:		
<b>Primo sottoprogramma</b>				
2100	SUB1:	Subtract SP, SP, #24		Salva in pila i registri
2104		Store LINK_reg,20(SP)		
2108		Store FP, 16(SP)		
2112		Store R2, 12(SP)		
2116		Store R3, 8(SP)		
2120		Store R4, 4(SP)		
2124		Store R5, (SP)		
2128		Add FP, SP, #16		Inizializza il puntatore all'area di attivazione
2132		Load R2, 8(FP)		Preleva il primo parametro
2136		Load R3, 12(FP)		Preleva il secondo parametro
		:		
		Load R4, PARAM3		Impila un parametro da passare a SUB2
		Subtract SP, SP, #4		
		Store R4, (SP)		
		Call SUB2		
		Load R4, (SP)		Spila il risultato ottenuto da SUB2
		Add SP, SP, #4		
		:		
		Store R5, 8(FP)		Impila la risposta
		Load R5, (SP)		Ripristina i registri
		Load R4, 4(SP)		
		Load R3, 8(SP)		
		Load R2, 12(SP)		
		Load FP, 16(SP)		
		Load LINK_reg,20(SP)		
		Add SP, SP, #24		
		Return		Rientro al programma principale

Continua nella parte b

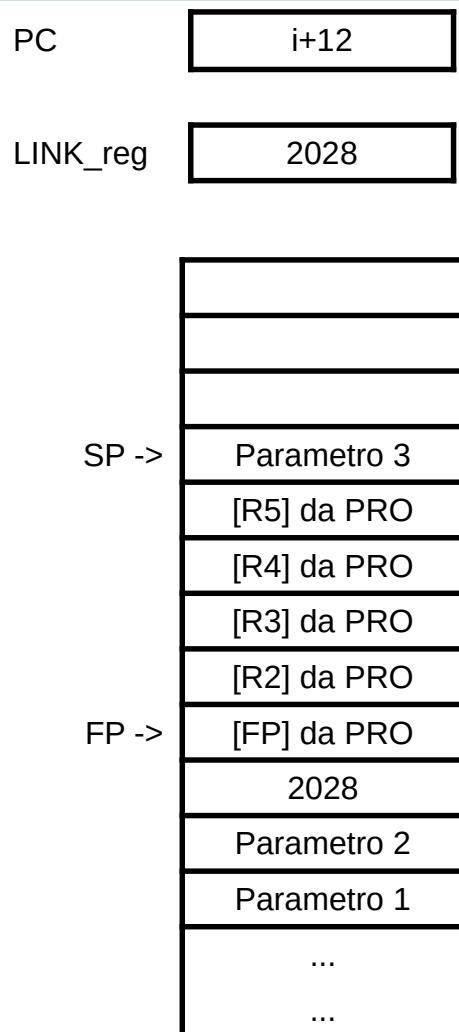
# Esempio sottoprogrammi annidati



Locazione di memoria		Istruzioni		Commenti
<b>Programma principale</b>				
		:		
2000	PRO:	Load R2, PARAM2		Impila i parametri
2004		Subtract SP, SP, #4		
2008		Store R2, (SP)		
2012		Load R2, PARAM1		
2016		Subtract SP, SP, #4		
2020		Store R2, (SP)		
2024		Call SUB1		Chiama il sottoprogramma
2028		Load R2, (SP)		Immagazzina il risultato
2032		Store R2, RIS		
2036		Add SP, SP, #8		Ripristina il livello della pila
2040		prossima istruzione		
		:		
<b>Primo sottoprogramma</b>				
2100	SUB1:	Subtract SP, SP, #24		Salva in pila i registri
2104		Store LINK_reg,20(SP)		
2108		Store FP, 16(SP)		
2112		Store R2, 12(SP)		
2116		Store R3, 8(SP)		
2120		Store R4, 4(SP)		
2124		Store R5, (SP)		
2128		Add FP, SP, #16		Inizializza il puntatore all'area di attivazione
2132		Load R2, 8(FP)		Preleva il primo parametro
2136		Load R3, 12(FP)		Preleva il secondo parametro
		:		
		Load R4, PARAM3		Impila un parametro da passare a SUB2
		Subtract SP, SP, #4		
		Store R4, (SP)		
		Call SUB2		
		Load R4, (SP)		Spila il risultato ottenuto da SUB2
		Add SP, SP, #4		
		:		
		Store R5, 8(FP)		Impila la risposta
		Load R5, (SP)		Ripristina i registri
		Load R4, 4(SP)		
		Load R3, 8(SP)		
		Load R2, 12(SP)		
		Load FP, 16(SP)		
		Load LINK_reg,20(SP)		
		Add SP, SP, #24		
		Return		Rientro al programma principale

Continua nella parte b

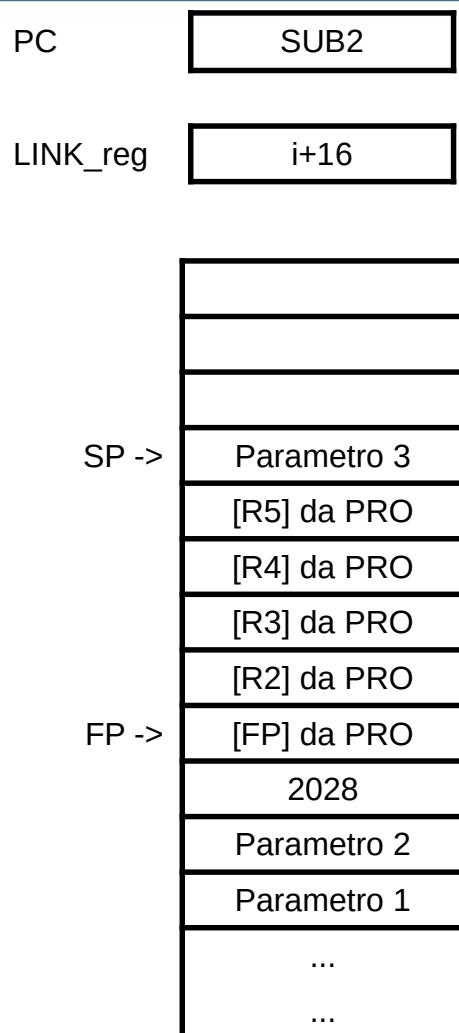
# Esempio sottoprogrammi annidati



Locazione di memoria		Istruzioni	Commenti
<b>Programma principale</b>			
		:	
2000	PRO:	Load R2, PARAM2	Impila i parametri
2004		Subtract SP, SP, #4	
2008		Store R2, (SP)	
2012		Load R2, PARAM1	
2016		Subtract SP, SP, #4	
2020		Store R2, (SP)	
2024		Call SUB1	Chiama il sottoprogramma
2028		Load R2, (SP)	Immagazzina il risultato
2032		Store R2, RIS	
2036		Add SP, SP, #8	Ripristina il livello della pila
2040		prossima istruzione	
		:	
<b>Primo sottoprogramma</b>			
2100	SUB1:	Subtract SP, SP, #24	Salva in pila i registri
2104		Store LINK_reg,20(SP)	
2108		Store FP, 16(SP)	
2112		Store R2, 12(SP)	
2116		Store R3, 8(SP)	
2120		Store R4, 4(SP)	
2124		Store R5, (SP)	
2128		Add FP, SP, #16	Inizializza il puntatore all'area di attivazione
2132		Load R2, 8(FP)	Preleva il primo parametro
2136		Load R3, 12(FP)	Preleva il secondo parametro
		:	
		Load R4, PARAM3	Impila un parametro da passare a SUB2
		Subtract SP, SP, #4	
		Store R4, (SP)	
		Call SUB2	
		Load R4, (SP)	Spila il risultato ottenuto da SUB2
		Add SP, SP, #4	
		:	
		Store R5, 8(FP)	Impila la risposta
		Load R5, (SP)	Ripristina i registri
		Load R4, 4(SP)	
		Load R3, 8(SP)	
		Load R2, 12(SP)	
		Load FP, 16(SP)	
		Load LINK_reg,20(SP)	
		Add SP, SP, #24	
		Return	Rientro al programma principale

Continua nella parte b

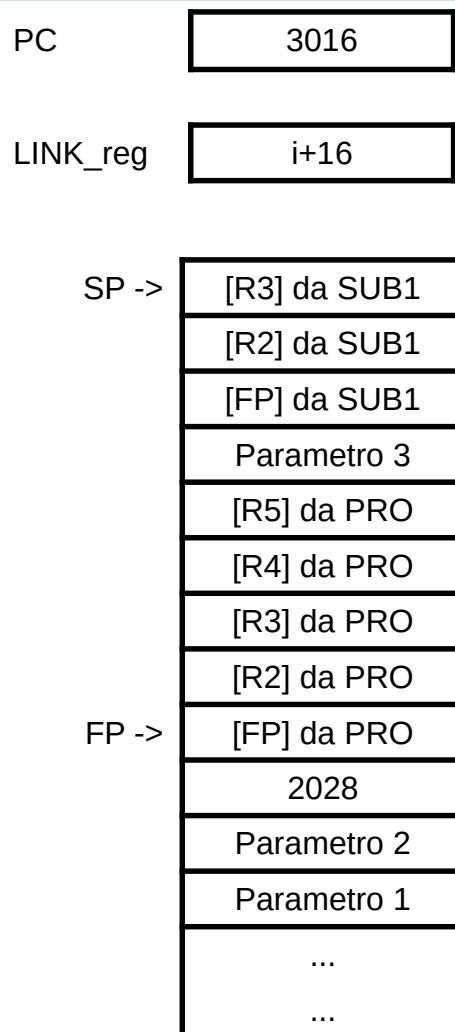
# Esempio sottoprogrammi annidati



Locazione di memoria		Istruzioni		Commenti
<b>Programma principale</b>				
		:		
2000	PRO:	Load R2, PARAM2		Impila i parametri
2004		Subtract SP, SP, #4		
2008		Store R2, (SP)		
2012		Load R2, PARAM1		
2016		Subtract SP, SP, #4		
2020		Store R2, (SP)		
2024		Call SUB1		Chiama il sottoprogramma
2028		Load R2, (SP)		Immagazzina il risultato
2032		Store R2, RIS		
2036		Add SP, SP, #8		Ripristina il livello della pila
2040		prossima istruzione		
		:		
<b>Primo sottoprogramma</b>				
2100	SUB1:	Subtract SP, SP, #24		Salva in pila i registri
2104		Store LINK_reg,20(SP)		
2108		Store FP, 16(SP)		
2112		Store R2, 12(SP)		
2116		Store R3, 8(SP)		
2120		Store R4, 4(SP)		
2124		Store R5, (SP)		
2128		Add FP, SP, #16		Inizializza il puntatore all'area di attivazione
2132		Load R2, 8(FP)		Preleva il primo parametro
2136		Load R3, 12(FP)		Preleva il secondo parametro
		:		
		Load R4, PARAM3		Impila un parametro da passare a SUB2
		Subtract SP, SP, #4		
		Store R4, (SP)		
		Call SUB2		
		Load R4, (SP)		Spila il risultato ottenuto da SUB2
		Add SP, SP, #4		
		:		
		Store R5, 8(FP)		Impila la risposta
		Load R5, (SP)		Ripristina i registri
		Load R4, 4(SP)		
		Load R3, 8(SP)		
		Load R2, 12(SP)		
		Load FP, 16(SP)		
		Load LINK_reg,20(SP)		
		Add SP, SP, #24		
		Return		Rientro al programma principale

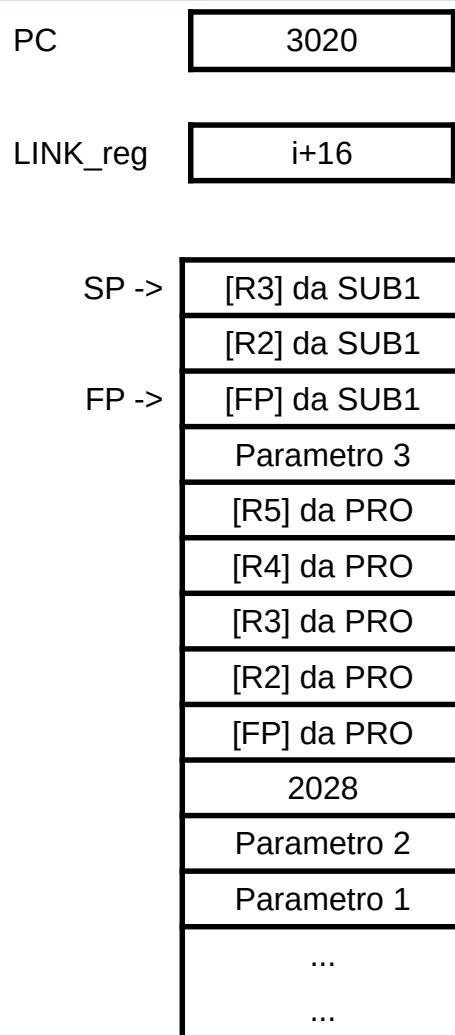
Continua nella parte b

# Esempio sottoprogrammi annidati



Locazione di memoria	Istruzioni	Commenti
<b>Secondo sottoprogramma</b>		
3000	SUB2: Subtract SP, SP, #12	Salva in pila i registri
3004	Store FP, 8(SP)	
	Store R2, 4(SP)	
	Store R3, (SP)	
	Add FP, SP, #8	Inizializza il puntatore all'area di attivazione
	Load R2, 4(FP)	Preleva il parametro
	:	
	Store R3, 4(FP)	Impila il risultato di SUB2
	Load R3, (SP)	Ripristina i registri
	Load R2, 4(SP)	
	Load FP, 8(SP)	
	Add SP, SP, #12	
	Return	Rientro al sottoprogramma 1

# Esempio sottoprogrammi annidati



Locazione di memoria	Istruzioni	Commenti
<b>Secondo sottoprogramma</b>		
3000	SUB2: Subtract SP, SP, #12	Salva in pila i registri
3004	Store FP, 8(SP)	
	Store R2, 4(SP)	
	Store R3, (SP)	
	Add FP, SP, #8	Inizializza il puntatore all'area di attivazione
	Load R2, 4(FP)	Preleva il parametro
	:	
	Store R3, 4(FP)	Impila il risultato di SUB2
	Load R3, (SP)	Ripristina i registri
	Load R2, 4(SP)	
	Load FP, 8(SP)	
	Add SP, SP, #12	
	Return	Rientro al sottoprogramma 1

# Esempio sottoprogrammi annidati

PC      j+4

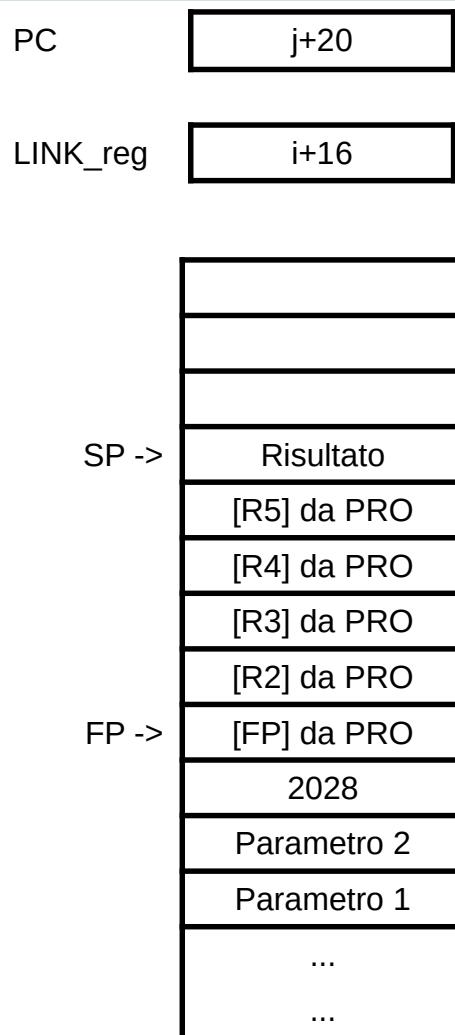
LINK\_reg    i+16

SP ->	[R3] da SUB1
	[R2] da SUB1
FP ->	[FP] da SUB1
	Risultato
	[R5] da PRO
	[R4] da PRO
	[R3] da PRO
	[R2] da PRO
	[FP] da PRO
	2028
	Parametro 2
	Parametro 1
	...
	...

Locazione di memoria	Istruzioni	Commenti
<b>Secondo sottoprogramma</b>		
3000	SUB2:              Subtract SP, SP, #12	Salva in pila i registri
3004	Store FP, 8(SP)	
	Store R2, 4(SP)	
	Store R3, (SP)	
	Add FP, SP, #8	Inizializza il puntatore all'area di attivazione
	Load R2, 4(FP)	Preleva il parametro
	:	
	Store R3, 4(FP)	Impila il risultato di SUB2
	Load R3, (SP)	Ripristina i registri
	Load R2, 4(SP)	
	Load FP, 8(SP)	
	Add SP, SP, #12	
	Return	Rientro al sottoprogramma 1

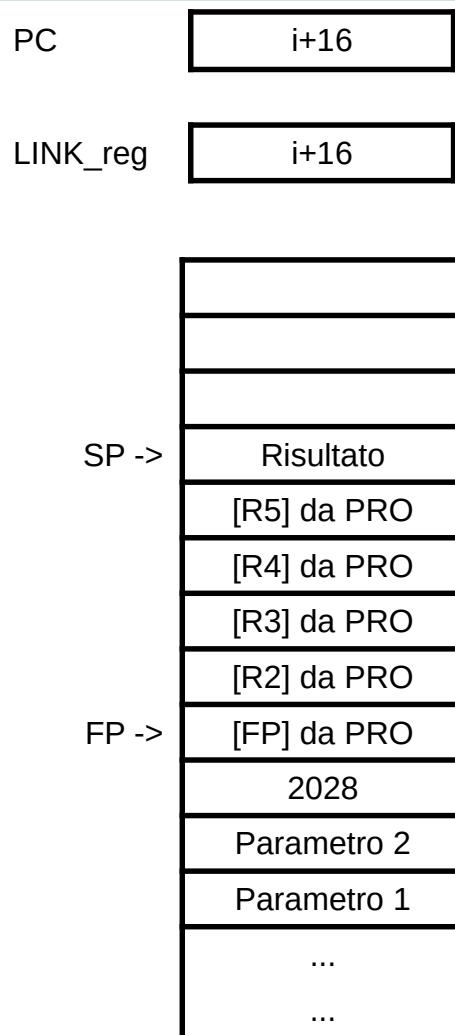


# Esempio sottoprogrammi annidati



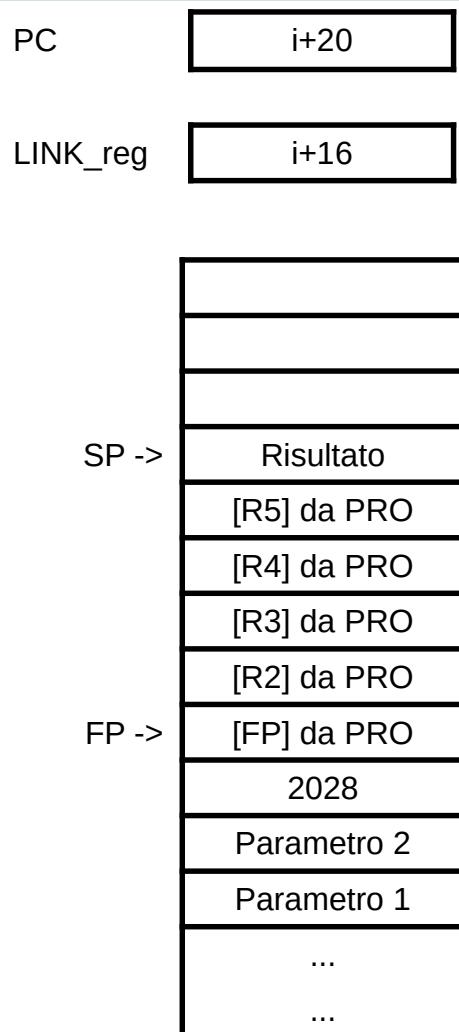
Locazione di memoria	Istruzioni	Commenti
<b>Secondo sottoprogramma</b>		
3000	SUB2: Subtract SP, SP, #12	Salva in pila i registri
3004	Store FP, 8(SP)	
	Store R2, 4(SP)	
	Store R3, (SP)	
	Add FP, SP, #8	Inizializza il puntatore all'area di attivazione
	Load R2, 4(FP)	Preleva il parametro
	:	
	Store R3, 4(FP)	Impila il risultato di SUB2
	Load R3, (SP)	Ripristina i registri
	Load R2, 4(SP)	
	Load FP, 8(SP)	
	Add SP, SP, #12	
	Return	Rientro al sottoprogramma 1

# Esempio sottoprogrammi annidati



Locazione di memoria	Istruzioni	Commenti
<b>Secondo sottoprogramma</b>		
3000	SUB2:      Subtract SP, SP, #12	Salva in pila i registri
3004	Store      FP, 8(SP)	
	Store      R2, 4(SP)	
	Store      R3, (SP)	
	Add      FP, SP, #8	Inizializza il puntatore all'area di attivazione
	Load      R2, 4(FP)	Preleva il parametro
	:	
	Store      R3, 4(FP)	Impila il risultato di SUB2
	Load      R3, (SP)	Ripristina i registri
	Load      R2, 4(SP)	
	Load      FP, 8(SP)	
	Add      SP, SP, #12	
	Return	Rientro al sottoprogramma 1

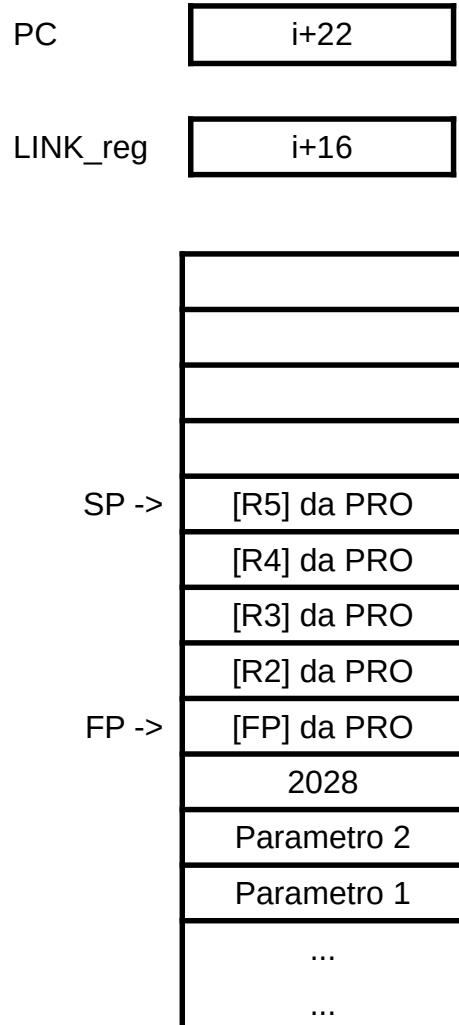
# Esempio sottoprogrammi annidati



Locazione di memoria		Istruzioni	Commenti
<b>Programma principale</b>			
		:	
2000	PRO:	Load R2, PARAM2	Impila i parametri
2004		Subtract SP, SP, #4	
2008		Store R2, (SP)	
2012		Load R2, PARAM1	
2016		Subtract SP, SP, #4	
2020		Store R2, (SP)	
2024		Call SUB1	Chiama il sottoprogramma
2028		Load R2, (SP)	Immagazzina il risultato
2032		Store R2, RIS	
2036		Add SP, SP, #8	Ripristina il livello della pila
2040		prossima istruzione	
		:	
<b>Primo sottoprogramma</b>			
2100	SUB1:	Subtract SP, SP, #24	Salva in pila i registri
2104		Store LINK_reg,20(SP)	
2108		Store FP, 16(SP)	
2112		Store R2, 12(SP)	
2116		Store R3, 8(SP)	
2120		Store R4, 4(SP)	
2124		Store R5, (SP)	
2128		Add FP, SP, #16	Inizializza il puntatore all'area di attivazione
2132		Load R2, 8(FP)	Preleva il primo parametro
2136		Load R3, 12(FP)	Preleva il secondo parametro
		:	
		Load R4, PARAM3	Impila un parametro da passare a SUB2
		Subtract SP, SP, #4	
		Store R4, (SP)	
		Call SUB2	
		Load R4, (SP)	Spila il risultato ottenuto da SUB2
		Add SP, SP, #4	
		:	
		Store R5, 8(FP)	Impila la risposta
		Load R5, (SP)	Ripristina i registri
		Load R4, 4(SP)	
		Load R3, 8(SP)	
		Load R2, 12(SP)	
		Load FP, 16(SP)	
		Load LINK_reg,20(SP)	
		Add SP, SP, #24	
		Return	Rientro al programma principale

Continua nella parte b

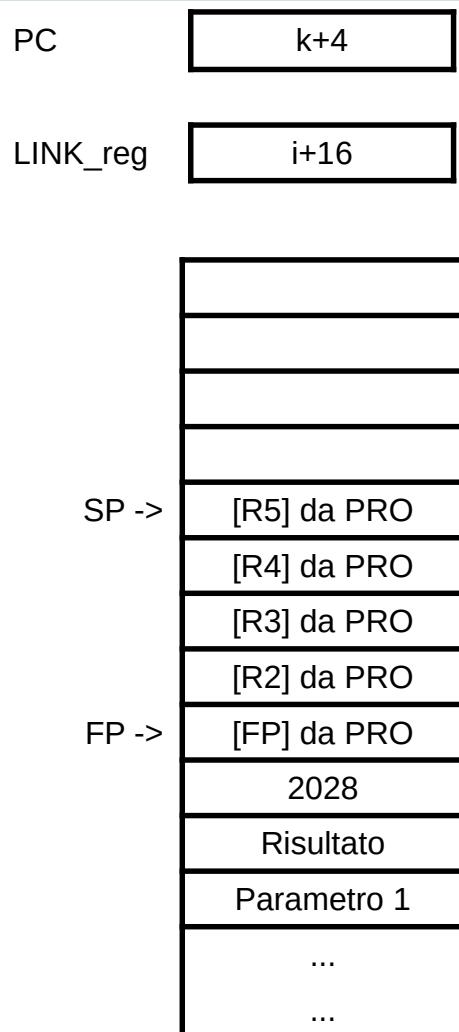
# Esempio sottoprogrammi annidati



Locazione di memoria		Istruzioni	Commenti
<b>Programma principale</b>			
		:	
2000	PRO:	Load R2, PARAM2	Impila i parametri
2004		Subtract SP, SP, #4	
2008		Store R2, (SP)	
2012		Load R2, PARAM1	
2016		Subtract SP, SP, #4	
2020		Store R2, (SP)	
2024		Call SUB1	Chiama il sottoprogramma
2028		Load R2, (SP)	Immagazzina il risultato
2032		Store R2, RIS	
2036		Add SP, SP, #8	Ripristina il livello della pila
2040		prossima istruzione	
		:	
<b>Primo sottoprogramma</b>			
2100	SUB1:	Subtract SP, SP, #24	Salva in pila i registri
2104		Store LINK_reg,20(SP)	
2108		Store FP, 16(SP)	
2112		Store R2, 12(SP)	
2116		Store R3, 8(SP)	
2120		Store R4, 4(SP)	
2124		Store R5, (SP)	
2128		Add FP, SP, #16	Inizializza il puntatore all'area di attivazione
2132		Load R2, 8(FP)	Preleva il primo parametro
2136		Load R3, 12(FP)	Preleva il secondo parametro
		:	
		Load R4, PARAM3	Impila un parametro da passare a SUB2
		Subtract SP, SP, #4	
		Store R4, (SP)	
		Call SUB2	
		Load R4, (SP)	Spila il risultato ottenuto da SUB2
		Add SP, SP, #4	
		:	
		Store R5, 8(FP)	Impila la risposta
		Load R5, (SP)	Ripristina i registri
		Load R4, 4(SP)	
		Load R3, 8(SP)	
		Load R2, 12(SP)	
		Load FP, 16(SP)	
		Load LINK_reg,20(SP)	
		Add SP, SP, #24	
		Return	Rientro al programma principale

Continua nella parte b

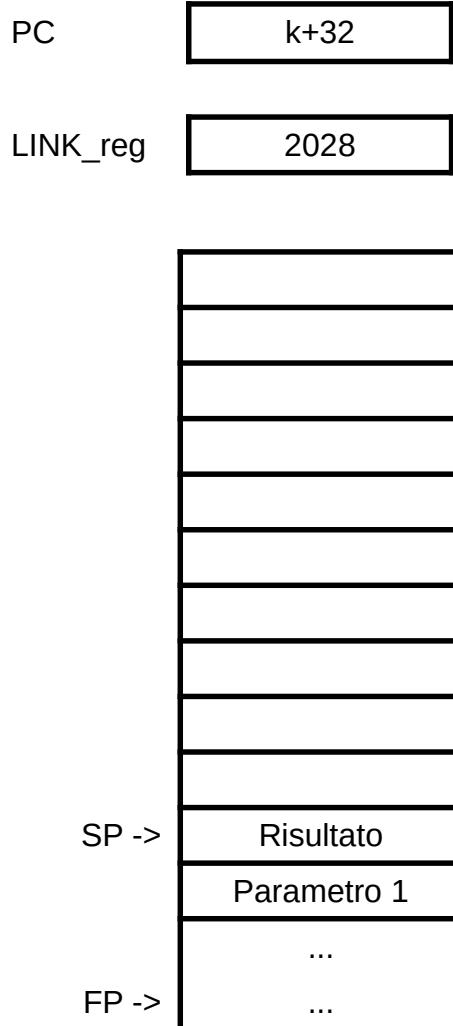
# Esempio sottoprogrammi annidati



Locazione di memoria		Istruzioni	Commenti
<b>Programma principale</b>			
		:	
2000	PRO:	Load R2, PARAM2	Impila i parametri
2004		Subtract SP, SP, #4	
2008		Store R2, (SP)	
2012		Load R2, PARAM1	
2016		Subtract SP, SP, #4	
2020		Store R2, (SP)	
2024		Call SUB1	Chiama il sottoprogramma
2028		Load R2, (SP)	Immagazzina il risultato
2032		Store R2, RIS	
2036		Add SP, SP, #8	Ripristina il livello della pila
2040		prossima istruzione	
		:	
<b>Primo sottoprogramma</b>			
2100	SUB1:	Subtract SP, SP, #24	Salva in pila i registri
2104		Store LINK_reg,20(SP)	
2108		Store FP, 16(SP)	
2112		Store R2, 12(SP)	
2116		Store R3, 8(SP)	
2120		Store R4, 4(SP)	
2124		Store R5, (SP)	
2128		Add FP, SP, #16	Inizializza il puntatore all'area di attivazione
2132		Load R2, 8(FP)	Preleva il primo parametro
2136		Load R3, 12(FP)	Preleva il secondo parametro
		:	
		Load R4, PARAM3	Impila un parametro da passare a SUB2
		Subtract SP, SP, #4	
		Store R4, (SP)	
		Call SUB2	
		Load R4, (SP)	Spila il risultato ottenuto da SUB2
		Add SP, SP, #4	
		:	
		Store R5, 8(FP)	Impila la risposta
		Load R5, (SP)	Ripristina i registri
		Load R4, 4(SP)	
		Load R3, 8(SP)	
		Load R2, 12(SP)	
		Load FP, 16(SP)	
		Load LINK_reg,20(SP)	
		Add SP, SP, #24	
		Return	Rientro al programma principale

Continua nella parte b

# Esempio sottoprogrammi annidati

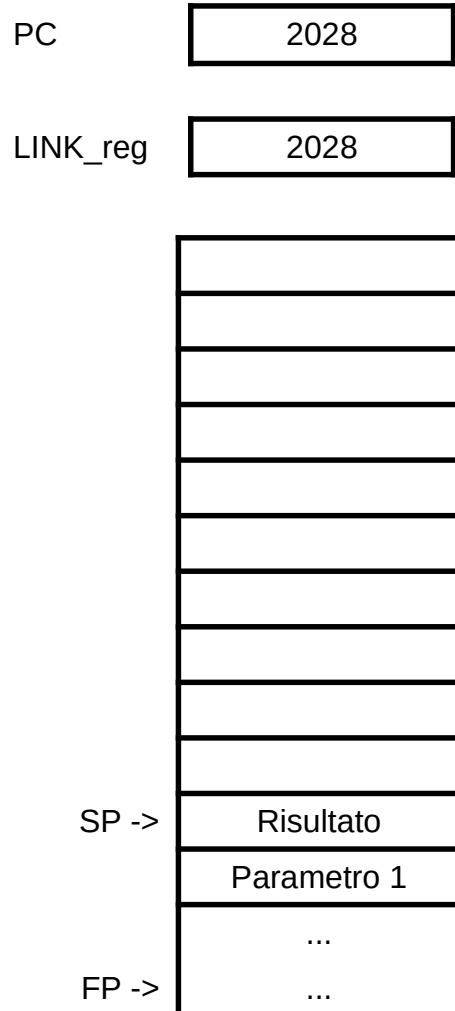


Locazione di memoria		Istruzioni	Commenti
<b>Programma principale</b>			
2000	PRO:	Load R2, PARAM2	Impila i parametri
2004		Subtract SP, SP, #4	
2008		Store R2, (SP)	
2012		Load R2, PARAM1	
2016		Subtract SP, SP, #4	
2020		Store R2, (SP)	
2024		Call SUB1	Chiama il sottoprogramma
2028		Load R2, (SP)	Immagazzina il risultato
2032		Store R2, RIS	
2036		Add SP, SP, #8	Ripristina il livello della pila
2040		prossima istruzione	
:			
<b>Primo sottoprogramma</b>			
2100	SUB1:	Subtract SP, SP, #24	Salva in pila i registri
2104		Store LINK_reg,20(SP)	
2108		Store FP, 16(SP)	
2112		Store R2, 12(SP)	
2116		Store R3, 8(SP)	
2120		Store R4, 4(SP)	
2124		Store R5, (SP)	
2128		Add FP, SP, #16	Inizializza il puntatore all'area di attivazione
2132		Load R2, 8(FP)	Preleva il primo parametro
2136		Load R3, 12(FP)	Preleva il secondo parametro
:			
		Load R4, PARAM3	Impila un parametro da passare a SUB2
		Subtract SP, SP, #4	
		Store R4, (SP)	
		Call SUB2	
		Load R4, (SP)	Spila il risultato ottenuto da SUB2
		Add SP, SP, #4	
:			
		Store R5, 8(FP)	Impila la risposta
		Load R5, (SP)	Ripristina i registri
		Load R4, 4(SP)	
		Load R3, 8(SP)	
		Load R2, 12(SP)	
		Load FP, 16(SP)	
		Load LINK_reg,20(SP)	
		Add SP, SP, #24	
		Return	Rientro al programma principale

Continua nella parte b

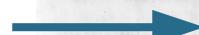


# Esempio sottoprogrammi annidati

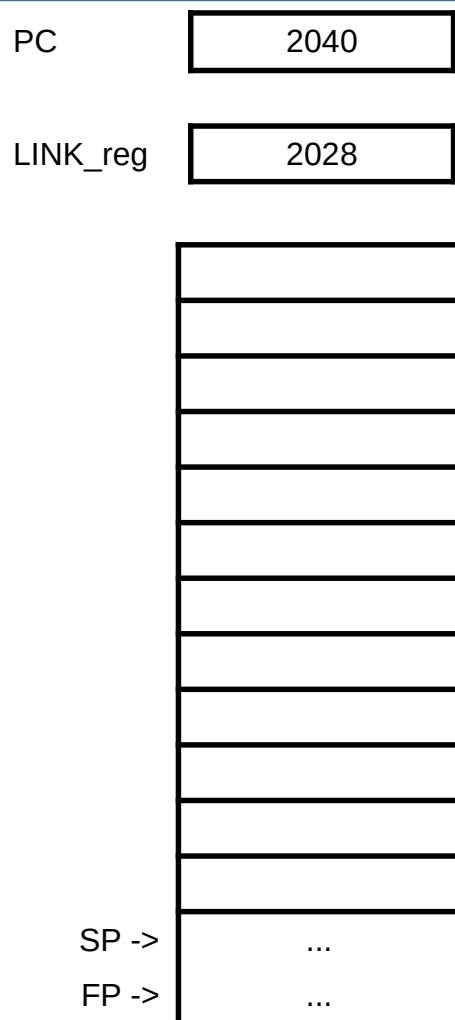


Locazione di memoria		Istruzioni	Commenti
<b>Programma principale</b>			
2000	PRO:	Load R2, PARAM2	Impila i parametri
2004		Subtract SP, SP, #4	
2008		Store R2, (SP)	
2012		Load R2, PARAM1	
2016		Subtract SP, SP, #4	
2020		Store R2, (SP)	
2024		Call SUB1	Chiama il sottoprogramma
2028		Load R2, (SP)	Immagazzina il risultato
2032		Store R2, RIS	
2036		Add SP, SP, #8	Ripristina il livello della pila
2040		prossima istruzione	
:			
<b>Primo sottoprogramma</b>			
2100	SUB1:	Subtract SP, SP, #24	Salva in pila i registri
2104		Store LINK_reg,20(SP)	
2108		Store FP, 16(SP)	
2112		Store R2, 12(SP)	
2116		Store R3, 8(SP)	
2120		Store R4, 4(SP)	
2124		Store R5, (SP)	
2128		Add FP, SP, #16	Inizializza il puntatore all'area di attivazione
2132		Load R2, 8(FP)	Preleva il primo parametro
2136		Load R3, 12(FP)	Preleva il secondo parametro
:			
		Load R4, PARAM3	Impila un parametro da passare a SUB2
		Subtract SP, SP, #4	
		Store R4, (SP)	
		Call SUB2	
		Load R4, (SP)	Spila il risultato ottenuto da SUB2
		Add SP, SP, #4	
:			
		Store R5, 8(FP)	Impila la risposta
		Load R5, (SP)	Ripristina i registri
		Load R4, 4(SP)	
		Load R3, 8(SP)	
		Load R2, 12(SP)	
		Load FP, 16(SP)	
		Load LINK_reg,20(SP)	
		Add SP, SP, #24	
		Return	Rientro al programma principale

Continua nella parte b



# Esempio sottoprogrammi annidati



Locazione di memoria		Istruzioni		Commenti
<b>Programma principale</b>				
2000	PRO:	Load	R2, PARAM2	Impila i parametri
2004		Subtract	SP, SP, #4	
2008		Store	R2, (SP)	
2012		Load	R2, PARAM1	
2016		Subtract	SP, SP, #4	
2020		Store	R2, (SP)	
2024		Call	SUB1	Chiama il sottoprogramma
2028		Load	R2, (SP)	Immagazzina il risultato
2032		Store	R2, RIS	
2036		Add	SP, SP, #8	Ripristina il livello della pila
2040				prossima istruzione
				:
<b>Primo sottoprogramma</b>				
2100	SUB1:	Subtract	SP, SP, #24	Salva in pila i registri
2104		Store	LINK_reg,20(SP)	
2108		Store	FP, 16(SP)	
2112		Store	R2, 12(SP)	
2116		Store	R3, 8(SP)	
2120		Store	R4, 4(SP)	
2124		Store	R5, (SP)	
2128		Add	FP, SP, #16	Inizializza il puntatore all'area di attivazione
2132		Load	R2, 8(FP)	Preleva il primo parametro
2136		Load	R3, 12(FP)	Preleva il secondo parametro
				:
		Load	R4, PARAM3	Impila un parametro da passare a SUB2
		Subtract	SP, SP, #4	
		Store	R4, (SP)	
		Call	SUB2	
		Load	R4, (SP)	Spila il risultato ottenuto da SUB2
		Add	SP, SP, #4	
				:
		Store	R5, 8(FP)	Impila la risposta
		Load	R5, (SP)	Ripristina i registri
		Load	R4, 4(SP)	
		Load	R3, 8(SP)	
		Load	R2, 12(SP)	
		Load	FP, 16(SP)	
		Load	LINK_reg,20(SP)	
		Add	SP, SP, #24	
		Return		Rientro al programma principale

Continua nella parte b