Annidamento (nidificazione)

- Una routine può chiamare al suo interno un'altra routine (annidamento, nesting)

```
PROG: CALL SUB1          LR ← PC

SUB1: CALL SUB2          LR ← PC
```

- Prima di chiamare la seconda routine bisogna salvare il contenuto del registro di collegamento per non perderlo al momento della seconda chiamata
- Con un numero qualsiasi di chiamate annidate, l'indirizzo di rientro è l'ultimo generato. La situazione è la stessa per tutti i rientri che concludono mano a mano le chiamate annidate
- I rientri sono in ordine inverso rispetto alle chiamate, quindi l'ultimo indirizzo di rientro salvato sarà il primo da ripristinare, ovvero in ordine LIFO last-in-first-out
- Alcuni processori come parte dell'esecuzione della chiamata a routine impilano l'indirizzo di ritorno sulla pila e aggiornano SP, quindi l'istruzione di return preleva l'indirizzo di ritorno dalla pila, senza aver bisogno di un registro di collegamento. Si dice che il meccanismo di chiamata **si serve della pila in *modo implicito***

Passaggio di parametri

- Vi sono due modi principali di passaggio di parametri e/o valori di ritorno
  - tramite registri, o
  - nella pila
- Quale dei due modi viene scelto dipende dal **programma** (si possono scegliere modi diversi per chiamate diverse)
- L'uso dei registri per passare i parametri è molto rapido e pratico, tuttavia i registri sono in numero limitato
- L'uso della pila permette di passare qualsiasi numero di parametri

Passaggio di parametri nei registri

- Il sottoprogramma calcola la somma della lista di numeri. Il programma chiamante passa al sottoprogramma la dimensione della lista N in R2, e la posizione della lista in R4. Il sottoprogramma restituisce il risultato lasciandolo in R3.

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

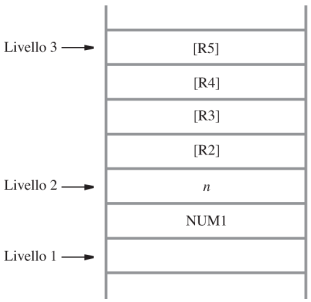
- Il programma chiamante memorizza il risultato in R3 all'indirizzo SOMMA
- Il sottoprogramma usa R5 per leggere un valore dalla memoria (prima istruzione in CICLO)
- Poiché R5 potrebbe essere usato dal programma chiamante, il sottoprogramma appena inizia salva R5 sulla pila, e quindi lo ripristina prima di rientrare al chiamante

Passaggio di parametri in pila

- Il programma chiamante passa NUM1 e N al sottoprogramma impilandoli. Al rientro dal sottoprogramma, legge dalla pila il risultato e lo conserva in SOMMA, quindi svuota la pila. Livello 2 è il livello in cui si trova la pila quando inizia l'esecuzione del sottoprogramma

Assumere che la cima della pila sia a livello 1 in Figura 2.19

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)
⋮		



Passaggio di parametri in pila

```
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 n
          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
```

- Il sottoprogramma salva in pila i registri che userà (R2, R3, R4, R5), così la pila raggiunge il livello 3. Si usa la modalità di indirizzamento con indice e spiazzamento
- Quindi, il sottoprogramma legge i parametri dalla pila (num elementi e puntatore lista)
- Al termine del calcolo della somma mette il risultato in pila (sovrascrivendo il puntatore alla lista), e ripristina i registri

Note su passaggio di parametri

- Alcuni calcolatori hanno istruzioni speciali per caricare e memorizzare più registri in pila. Es.

```
StoreMultiple R2-R5, -(SP)
```

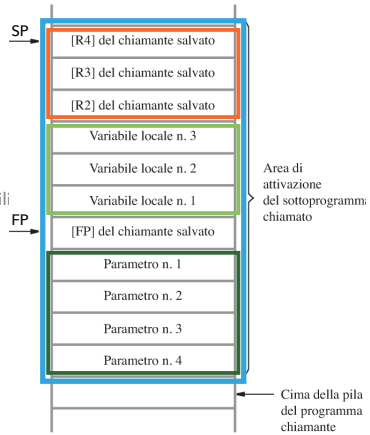
- memorizza i registri sorgenti da R2, R3, R4 e R5 in pila, e SP è aggiornato in modo appropriato, ovvero decrementato di 4 prima di memorizzare ciascun registro
- Analogamente l'istruzione seguente carica dalla pila i valori che erano stati conservati

```
LoadMultiple R2-R5, (SP)+
```

- Metodi di passaggio di parametri: il primo parametro passato, `NUM1` è l'indirizzo del primo elemento della lista, quindi è un riferimento; il secondo parametro passato, n, è il numero di elementi. Pertanto, in senso logico, il primo parametro è passato *per riferimento*, il secondo è passato *per valore*

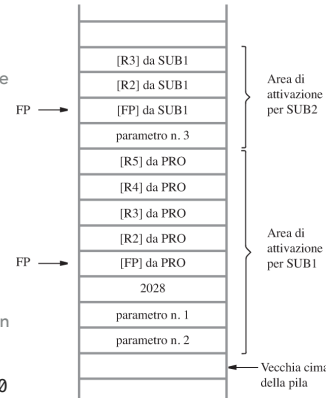
Area di attivazione

- La parte sommitale della pila, contenente dall'alto: registri salvati, eventuali variabili locali, e parametri passati, è detta **area di attivazione** (stack frame, o activation record) della routine
- La cima della pila è sempre puntata da SP
- Il registro puntatore **FP** (frame pointer), puntatore all'area di attivazione, è utile per accedere ai parametri passati alla routine
- FP punta all'interno dell'area di attivazione: i parametri passati si possono leggere con indice e spiazzamento: `4(FP)`, etc.; e le variabili locali con `-4(FP)`, etc.
- FP ha valore costante per tutta l'esecuzione della routine
- Il programma chiamante impila i parametri, quindi chiama la routine
- **La routine impila** FP (in uso da parte del chiamante), e **aggiorna** FP **con il valore di** SP, quindi impila variabili locali e registri da salvare
- Alla fine dell'esecuzione, la routine dealloca registri salvati, variabili locali e valore di FP salvato, ripristinando FP, quindi esegue return
- Il programma chiamante ha la responsabilità di deallocare i parametri



Aree di attivazione con annidamento

- Le aree di attivazione di sottoprogrammi annidati sono contenute nella pila una sopra l'altra. I valori di FP conservati in pila sono concatenati, ovvero il valore di FP conservato sopra punta a quello sotto, etc.
- Si abbia il programma principale `PR0` che chiama la routine `SUB1`, che a sua volta chiama la routine `SUB2`, il passaggio di parametri avviene tramite pila, il collegamento si serve di LR
  - L'area di attivazione di `SUB2` è sopra quella di `SUB1`. Viene mostrata la pila nel momento di massima espansione
  - Da sotto: `PR0` impila i parametri passati a `SUB1`; quindi `SUB1` impila LR, FP, registri R2-R5, e dopo alcuni calcoli, un parametro passato a `SUB2`; quindi `SUB2` impila FP, registri R2-R3
  - `SUB1` impila LR poiché deve poi chiamare una routine e la pila non è usata in modo implicito
  - I parametri sono letti dalla routine con `Load Rn, X(FP)`, con `X>0`
  - Il compito di eliminare i parametri tocca al chiamante



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		

Secondo sottoprogramma

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

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

Ulteriori istruzioni macchina (S. 2.8)

- Fino adesso sono state usate istruzioni `Load`, `Store`, `Move`, `Clear`, `Add`, `Subtract`, `Branch`, `Call`, `Return`
- Altre istruzioni utili sono `And`, `Or`, `Not`

`And R4, R2, R3`

- calcola l'AND (prodotto) bit a bit degli operandi nei registri R2 e R3 e mette il risultato in R4
- Per azzerare i 3 byte a sinistra di R2, si usa `And R2, R2, #0xFF`
  - `FF` è l'esadecimale per `11111111`, che viene esteso a sinistra con `0` per 3 byte (per riempire una parola di memoria). Con l'AND bit a bit la parte di R2 che rimane invariata è solo il byte più a destra

```
And  R2, R2, #0xFF
Move R3, #0x5A                carica 5A in R3, 5A è il codice ASCII del carattere Z
Branch_If_[R2]=[R3] TROVATOZ
```