AXO - Architettura dei Calcolatori e Sistemi Operativi

sottoprogrammi e 68000

istruzioni di modifica del flusso di esecuzione
chiamata a sottoprogramma

- nei linguaggi di alto livello, per esempio C, la chiamata a sottoprogramma ha come effetto la creazione di un record di attivazione sullo stack (pila), associato alla chiamata e contenente
  - i parametri attuali
  - l'indirizzo di ritorno al chiamante
  - informazioni per gestire lo spazio allocato per il record di attivazione
  - le variabili locali del sottoprogramma
  - l'eventuale valore restituito

chiamata a sottoprogramma

- in ISA, la chiamata a sottoprogramma di alto livello viene espansa in più istruzioni macchina, eseguite in "ambiti diversi"
  - il chiamante gestisce la parte relativa al passaggio dei parametri.
  - il chiamante attiva il sottoprogramma tramite **l'istruzione di chiamata ISA** (JSR o BSR per il Motorola 68000)
  - l'esecuzione dell'istruzione di chiamata ISA gestisce la parte relativa al salvataggio dell'indirizzo di ritorno (**PC**) sullo stack, e attiva il sottoprogramma
  - il sottoprogramma gestisce l'allocazione sulla pila delle variabili locali e dell'eventuale valore restituito

modello di chiamata a sottoprogramma

- la chiamata a sottoprogramma segue sempre l'"espansione" vista, ma il modello per la chiamata a sottoprogrammi non è identico in tutti i processori
  - alcuni processori vincolano la modalità
  - in altri è possibile operare delle scelte (68000)
- dove si salvano i valori dei parametri attuali: registri e/o pila ?
- in quale ordine vanno passati i parametri ?
- salvataggio dei registri usati dal sottoprogramma: chi li salva ?
- valore restituito: dove viene passato ? in pila ? in registro ?
- chi dealloca lo spazio in pila eventualmente utilizzato, per parametri e variabili locali ?
- si definisce l'area di attivazione ? come viene gestita ?

istruzioni 68000 per chiamata/ritorno da sottoprog. senza l'utilizzo dell'area di attivazione

- istruzione di salto a sottoprogramma

BSR (*branch to subroutine*) salto "vicino"
JSR (*jump to subroutine*) salto "lontano"

```
SP  ← [SP] - 4
[SP] ← [PC]
PC  ← i.e. dest.
```

salto incondizionato che salva sullo stack il *program counter (indirizzo di ritorno) e salta all'indirizzo specificato*

- istruzione di ritorno da sottoprogramma

RFS (*return from subroutine*)

```
PC  ← [[SP]]
SP  ← [SP] + 4
```

estrae dalla pila il *program counter*
e salta all'indirizzo appena disimpilato

passaggio di parametri e valore restituito (senza area di attivazione) - 1

passaggio dei parametri

- l'operazione viene svolta dal chiamante
- su registri (eseguita dalla maggior parte dei processori attuali)
  - il numero di registri è limitato e si esaurisce rapidamente
  - in chiamate ricorsive è necessario salvare poi sullo stack questi registri (oltre a eventuali altri registri del processore – vedi dopo)
- sullo stack (è la scelta che consideriamo)
  - con salvataggio esplicito da parte del chiamante
  - passando i parametri nell'ordine in cui vengono elencati (il primo parametro è il primo ad essere salvato in pila)

passaggio di parametri e valore restituito (senza area di attivazione) - 2

- valore restituito
  - su registro
  - sullo stack sovrascrivendo il primo parametro passato (è la scelta che consideriamo)
    - nel caso in cui il sottoprogramma non abbia parametri, il chiamante deve comunque allocare sulla pila una parola per il valore restituito
    - il chiamante deve prevedere, dopo l'istruzione di chiamata, un'istruzione che toglie il valore restituito dalla cima dello stack e lo salva in memoria o registro
- salvataggio dei registri e deallocazione dello stack occupato da parametri e variabili locali
  - se un sottoprogramma usa registri del processore in cui ci sono valori utili per il chiamante al ritorno, questi registri devono essere salvati; l'operazione viene fatta dal sottoprogramma (dal chiamato)
  - la deallocazione dello stack per la parte dei parametri viene fatta dal chiamante, mentre quella relativa alle variabili locali viene fatta dal chiamato

esempio – Hamacher 5.12

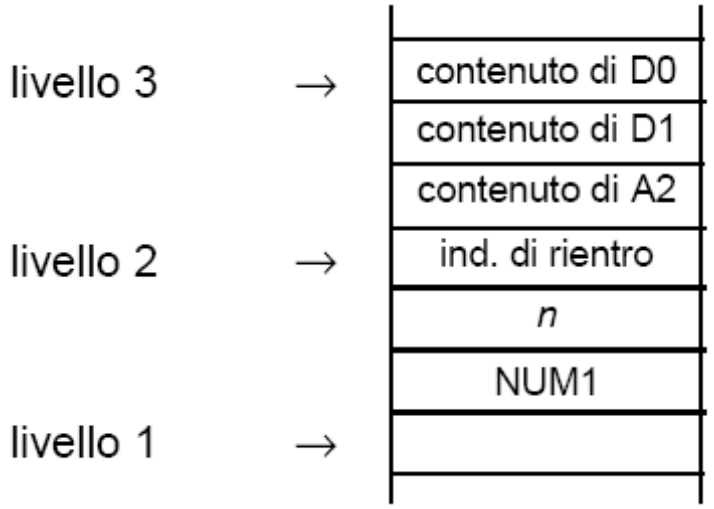
(la cima della pila si trova al livello 1)

programma chiamante

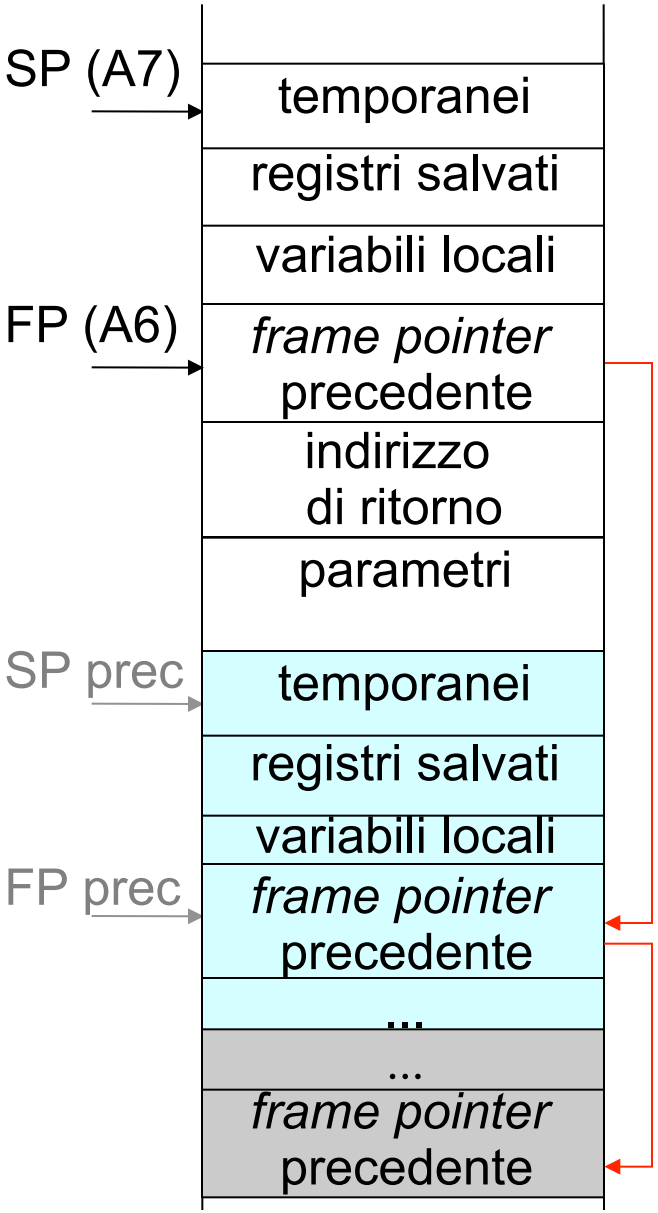
```
                MOVE.L    #NUM1, −(A7)      impila il parametro
                MOVE.L    N, −(A7)          impila il parametro
                BSR       SOMMA_LISTA       chiama la routine
                MOVE.L    4(A7), SOMMA      memorizza il risultato
                ADDI.L    #8, A7            abbassa la pila
                . . .     . . .
```

sottoprogramma chiamato

```
SOMMA_LISTA     MOVEM.L   D0-D1/A2, −(A7)   salva i registri
                MOVE.L    16(A7), D1        predisponi il contatore
                SUBQ.L    #1, D1            decrementa il contatore
                MOVEA.L   20(A7), A2        punta alla lista
                CLR.L     D0                azzera la somma iniziale
CICLO           ADD.W     (A2)+, D0         addiziona l'elemento
                DBRA      D1, CICLO         se non hai finito ripeti
                MOVE.L    D0, 20(A7)        memorizza la somma
                MOVEM.L   (A7)+, D0-D1/A2   ripristina i registri
                RTS                         rientra al programma
```



uso dell'area di attivazione

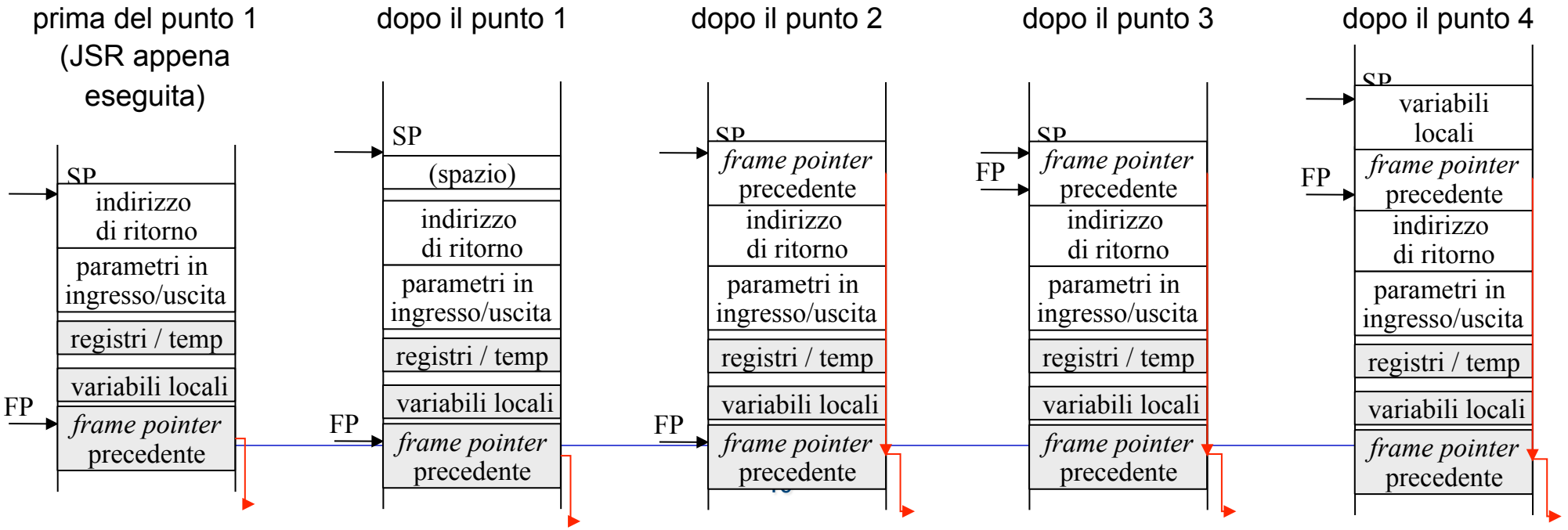


- una migliore realizzazione del meccanismo di chiamata a sottoprogramma fa uso delle **aree di attivazione** (*stack frame*)
- in 68000 esistono istruzioni dedicate per allocare e deallocare aree di attivazione: LINK e UNLK
- le aree di attivazione formano una lista concatenata (in rosso nella figura)
- per convenzione, il puntatore all'area di attivazione corrente (*frame pointer*) *è nel* ***registro*** **A6**
- creazione del frame `LINK A6, #-N`
  (N è la dimensione delle variabili locali del chiamato)
- deallocazione del frame `UNLK A6`
- i **parametri** hanno spiazzamento **positivo rispetto a FP**
  esempio: il primo parametro .L sta a `8(A6)`
- le **variabili locali** hanno spiazzamento **negativo rispetto a FP**
  esempio: la prima var. locale .L sta a `-4(A6)`

effetto dell' istruzione LINK

- **istruzione `LINK A6, #-N`**

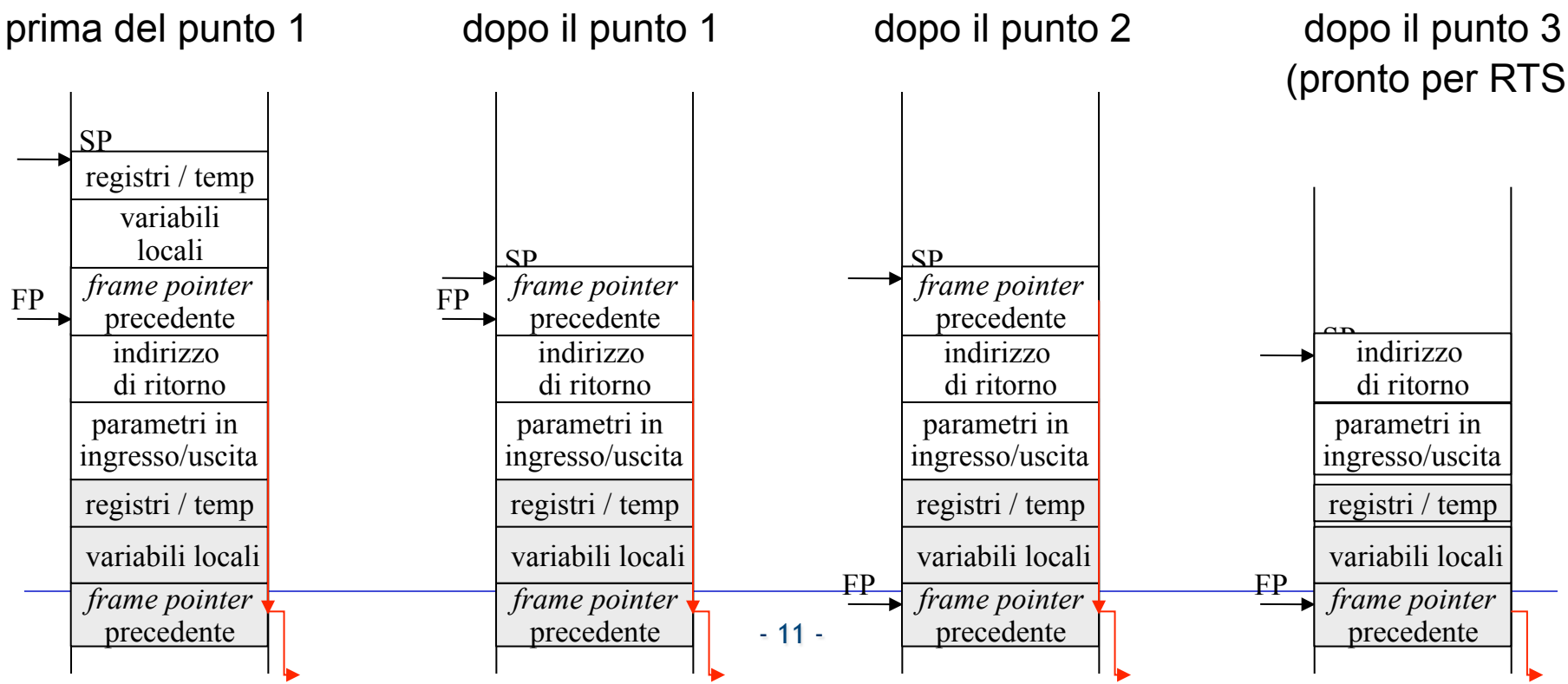
1.	[SP]	←	[SP] – 4	**riservo una parola lunga sulla pila**
2.	[M([SP])]	←	[A6]	**salvo vecchio *frame pointer***
3.	[A6]	←	[SP]	**il *frame pointer* ora punta alla cima della pila**
4.	[SP]	←	[SP] + N	**riservo spazio per le variabili locali (attenzione: –N è solitamente negativo)**



effetto dell' istruzione UNLK

- **istruzione `UNLK A6`**

1. **[SP] ← [A6]** — **la nuova cima della pila è il FP precedente**
2. **[A6] ← [M([SP])]** — **ripristino il vecchio *frame pointer***
3. **[SP] ← [SP] + 4** — **abbasso la cima della pila di una parola lunga**



schema di sottoprogramma

```
SUB1:
    LINK    A6, #-N             // riservo spazio per le var locali di SUB1
    MOVEM.L D1-...,-(SP)        // salvo i registri che uso
    ...
    // preparazione a chiamata di un'altra funzione con due parametri
    MOVE.L  ..., -(SP) // push primo parametro
    MOVE.L  ..., -(SP)          // push secondo parametro
    BRS     SUB2                // chiamata
    ADDA.L  #4, SP              // pop (n-1)parametri * dim parametri
    MOVE.L  (SP)+, ...          // spilo e uso il valore in uscita di SUB2
                                // sovrascritto al primo parametro
    ...
    MOVE.L ..., spiaz.(A6)      // salvo in stack il valore restituito da SUB1
      in posizione (+4 per ind.rit. + 4(n), n numero di parametri, 4 dim
      parametri) rispetto a FP
    MOVEM.L (SP)+,D1-...        // ripristino i registri
    UNLK    A6                  // elimino var. locali
    RTS                         // ritorno al chiamante
```