Sottoprogrammi in linguaggio assembly del Motorola 68000

Daniele Paolo Scarpazza
daniele.scarpazza@elet.polimi.it

Politecnico di Milano
Ultimo aggiornamento: 10 Maggio 2005

Bibliografia

- Sezioni 4.9, 5.6 del libro di testo:

  Hamacher, Vranesic & Zaky

  Introduzione all'architettura dei calcolatori

  McGraw-Hill Science

- Esempi importanti di traduzione da C ad assembly:

  `www.eventhelix.com/RealtimeMantra/Basics/CToAssemblyTranslation.htm`

- Esempi sull'uso del record di attivazione:

  `www.cwru.edu/cse/eeap/282/25_stack_frames.html`

Strumenti

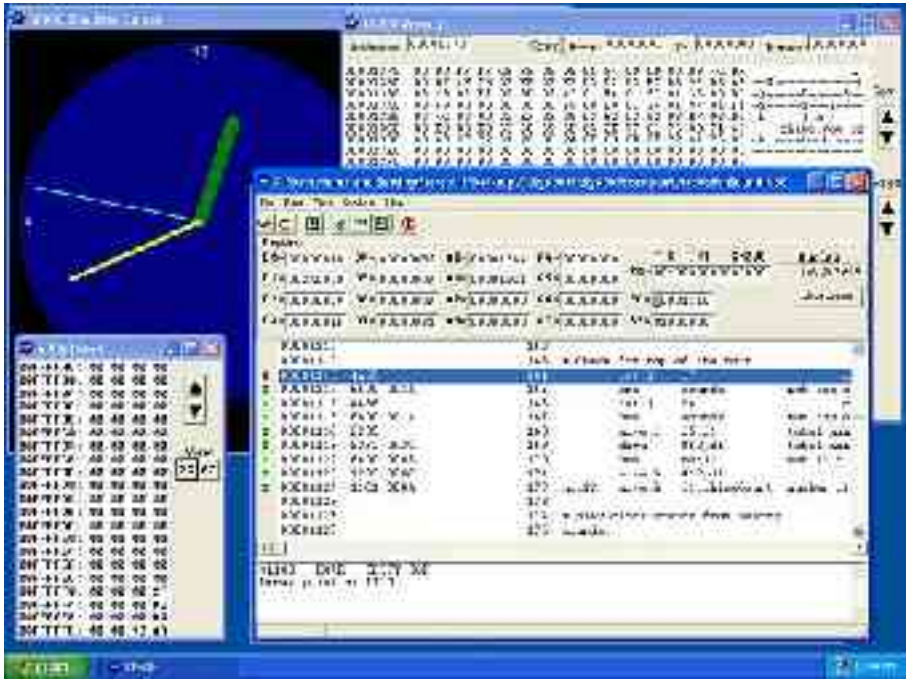
- Strumento da usare:

  EASy68K

  Editor/Assembler/Simulator for the 68000

  Disponibile presso:

  `http://www.monroeccc.edu/ckelly/easy68k.htm`



Modello di chiamata di sottoprogramma

- Interessa le seguenti scelte:
  - Dove vanno posti i parametri in ingresso?
    (nei registri? nella pila? in entrambi?)
  - In quale ordine si caricano i parametri sulla pila?
    (solitamente le push appaiono in ordine inverso)
  - Chi deve salvare i registri?
    (il chiamante? il chiamato? quali? partizionamento *caller-save* e *callee-save* registers?)
  - Come si passa il valore di ritorno?
    (in un registro? sulla pila? come?
    solitamente sovrascrivendo i parametri)
  - Chi abbatte la pila al ritorno?
    (il chiamante? il chiamato?)
  - Si usa una catena di record di attivazione (*stack frames*) oppure no?
- Le scelte confluiscono nella cosiddetta
  Application Binary Interface (ABI) di un certo sistema operativo;
  in assenza di sistema operativo, le scelte sono libere;

Chiamata di sottoprogramma

- Istruzione di salto a sottoprogramma:
  `BSR` (*branch to subroutine*) salto “vicino”
  `JSR` (*jump to subroutine*) salto “lontano”
  salto incondizionato che impila il *program counter*
  e salta all'indirizzo specificato;
- Istruzione di ritorno da sottoprogramma:
  `RTS` (*return from subroutine*)
  disimpila il *program counter*
  e salta all'indirizzo appena disimpilato;

Passaggio dei parametri nei registri

- Esempio di programma *(~Fig. 5.11, remake di 5.8)*
  che calcola la somma di un array di interi a 32 bit.
- Parametri di ingresso: in A2 l'indirizzo e in D1 la lunghezza dell'array;
- Valore di ritorno: lascia la somma calcolata in D0;

```
          ORG       $2000
SUM       EQU       $2000                    * we store the sum here
ARRAY     DC.L      1,2,3,4,5,6,7,8,9,10     * an array of 10 integers
LENGTH    EQU       10                       * the array length

          ORG       $1000
DO_SUM    ********************************************************
          SUBQ.L    #1,D1                    * initialize the loop index to N-1
          CLR.L     D0                       * use D0 as accumulator; clear it
LOOP      ADD.L     (A2)+, D0                * sum current cell to D0
          DBRA      D1, LOOP                 * decrement index D1 and loop
          RTS                                * return from subroutine

START     ********************************************************
          MOVEA.L   #ARRAY, A2               * set array location parameter to A2
          MOVE.L    #LENGTH, D1              * set array length parameter to D1
          BSR       DO_SUM                   * call the subroutine
          MOVE.L    D0, SUM                  * save the final sum to variable
          STOP      #$2000
          END       START
```

Passaggio dei parametri sulla pila

- Modifica del programma precedente *(~Fig. 5.12)*
  con passaggio di parametri e del valore di ritorno sulla pila;
- La gestione dello stack è esplicita e non fa uso di *frame pointer*;
- Attenzione al modello di chiamata scelto:
  - push dei parametri nello stesso ordine in cui sono “pensati”,
    cioè prima la lunghezza dell'array (primo parametro)
    e poi indirizzo dell'array (secondo parametro);
    di solito succede il contrario;
  - valore di ritorno che sovrascrive il primo parametro;
  - salvataggio dei registri da parte del chiamato;
    il chiamato salva i soli registri che lui sa di usare;
  - il chiamante rimuove i parametri dalla pila dopo la chiamata;

Passaggio dei parametri sulla pila

- Modifica del programma precedente *(~Fig. 5.12)*
  con passaggio di parametri e del valore di ritorno sulla pila;
- La gestione dello stack è esplicita e priva di frame pointer;

```
            ORG         $2000
SUM         EQU         $2000                   * the variable where we store the sum
ARRAY       DC.L 1,2,3,4,5,6,7,8,9,10           * an array of 10 integers
LENGTH      EQU         10                      * the array length

            ORG         $1000
DO_SUM      MOVEM.L     D0-D1/A2,-(A7)          * save D0, D1 and A2 on the stack
            MOVE.L      16(A7), D1              * retrieve array length from the stack
            SUBQ.L      #1,D1                   * initialize the loop index to N-1
            MOVEA.L     20(A7), A2              * retrieve array location from the stack
            CLR.L       D0                      * use D0 as accumulator; clear it
      LOOP  ADD.L       (A2)+, D0               * sum current cell to D0
            DBRA        D1, LOOP                * decrement index D1 and loop
            MOVE.L      D0, 20(A7)              * save the result onto the stack
            MOVEM.L     (A7)+,D0-D1/A2          * restores the registers from the stack
            RTS                                 * return from subroutine

START       MOVE.L      #ARRAY,  -(A7)          * push array location parameter
            MOVE.L      #LENGTH, -(A7)          * push array length parameter
            BSR         DO_SUM                  * call the subroutine
            MOVE.L      4(A7), SUM              * retrieve the result from the stack
            ADD.L       #8, A7                  * reposition the stack pointer
            STOP        #$2000
            END         START
```

Parametri sulla pila: simulazione

Stato della pila:

Codice:

```
START   MOVE.L    #ARRAY, -(A7)      * array location
        MOVE.L    #LENGTH,-(A7)      * array length
        BSR       DO_SUM             * call
        MOVE.L    4(A7), SUM         * result
        ADD.L     #8, A7             * reposition SP
        ...

DO_SUM  MOVEM.L   D0-D1/A2,-(A7)     * save regs
        MOVE.L    16(A7), D1         * array length
        MOVEA.L   20(A7), A2         * array location
        ...                          * loop
        MOVE.L    D0, 20(A7)         * write result
        MOVEM.L   (A7)+,D0-D1/A2     * restores regs
        RTS                          * return
```

Parametri sulla pila: simulazione

Stato della pila:

Codice:

```
START   MOVE.L    #ARRAY, -(A7)     * array location
        MOVE.L    #LENGTH,-(A7)     * array length
        BSR       DO_SUM            * call
        MOVE.L    4(A7), SUM        * result
        ADD.L     #8, A7            * reposition SP
        ...

DO_SUM  MOVEM.L   D0-D1/A2,-(A7)    * save regs
        MOVE.L    16(A7), D1        * array length
        MOVEA.L   20(A7), A2        * array location
        ...                         * loop
        MOVE.L    D0, 20(A7)        * write result
        MOVEM.L   (A7)+,D0-D1/A2    * restores regs
        RTS                         * return
```

Parametri sulla pila: simulazione

Stato della pila:

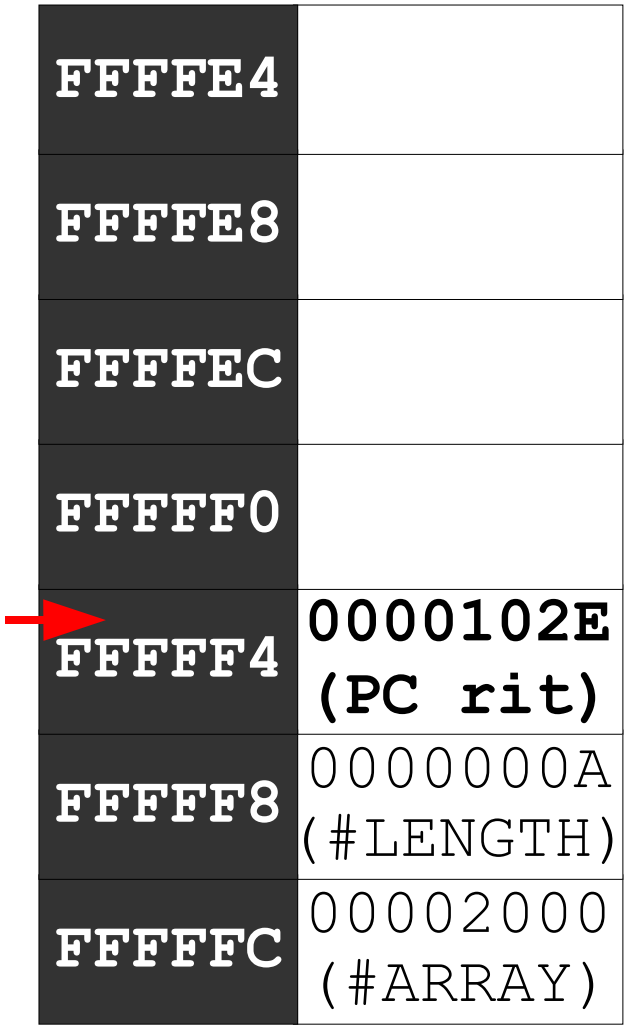
Codice:

```
START   MOVE.L    #ARRAY, -(A7)     * array location
        MOVE.L    #LENGTH,-(A7)     * array length
        BSR       DO_SUM            * call
        MOVE.L    4(A7), SUM        * result
        ADD.L     #8, A7            * reposition SP
        ...

DO_SUM  MOVEM.L   D0-D1/A2,-(A7)    * save regs
        MOVE.L    16(A7), D1        * array length
        MOVEA.L   20(A7), A2        * array location
        ...                         * loop
        MOVE.L    D0, 20(A7)        * write result
        MOVEM.L   (A7)+,D0-D1/A2    * restores regs
        RTS                         * return
```

Parametri sulla pila: simulazione

Stato della pila:



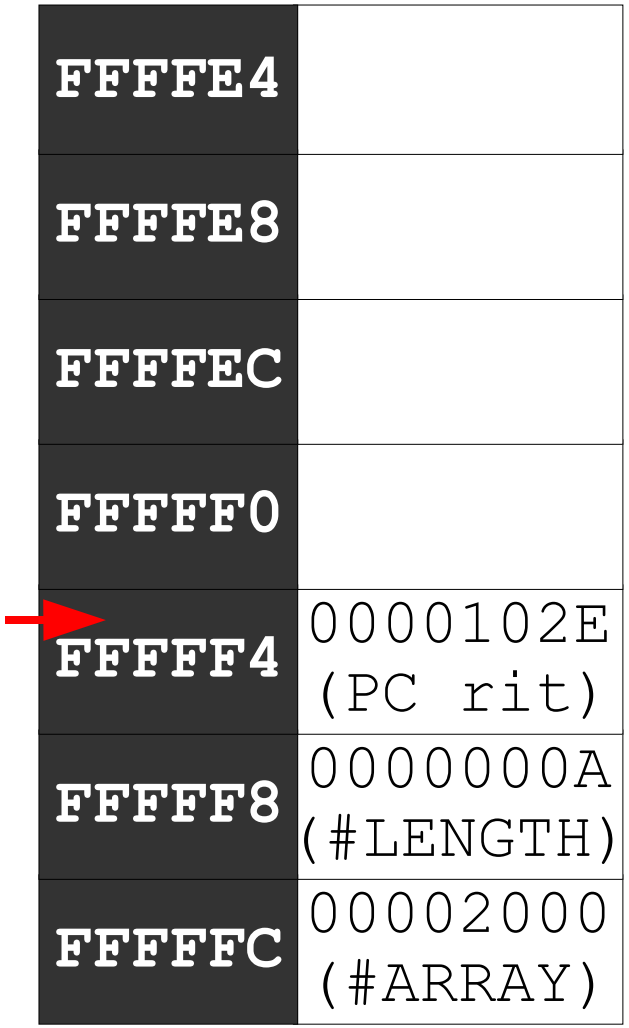
Codice:

```
START   MOVE.L    #ARRAY, -(A7)     * array location
        MOVE.L    #LENGTH,-(A7)     * array length
        BSR       DO_SUM            * call
        MOVE.L    4(A7), SUM        * result
        ADD.L     #8, A7            * reposition SP
        ...

DO_SUM  MOVEM.L   D0-D1/A2,-(A7)    * save regs
        MOVE.L    16(A7), D1        * array length
        MOVEA.L   20(A7), A2        * array location
        ...                         * loop
        MOVE.L    D0, 20(A7)        * write result
        MOVEM.L   (A7)+,D0-D1/A2    * restores regs
        RTS                         * return
```

Parametri sulla pila: simulazione

Stato della pila:



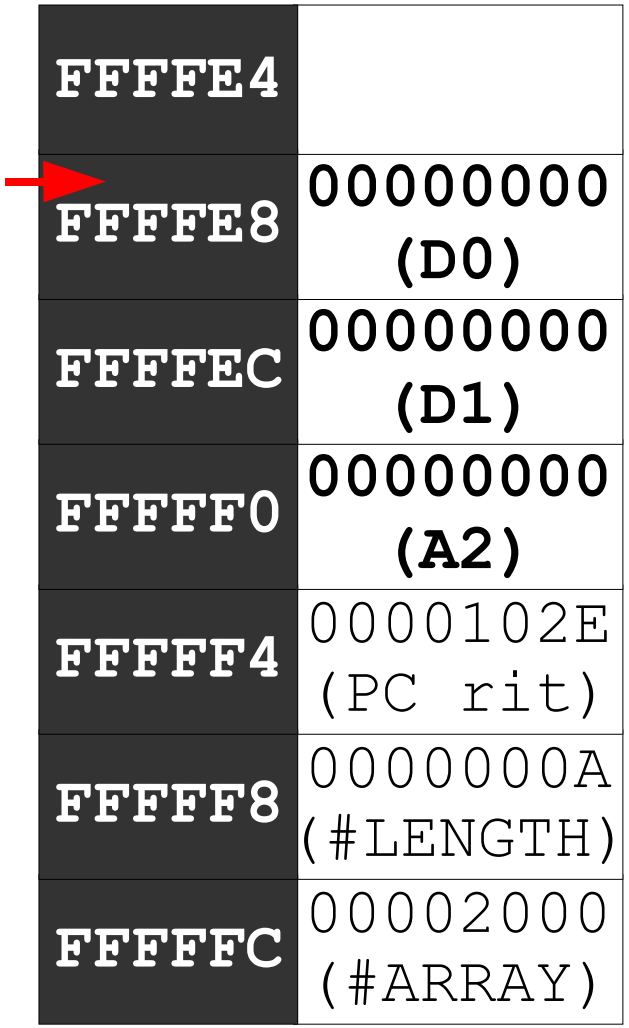
Codice:

```
START   MOVE.L    #ARRAY, -(A7)     * array location
        MOVE.L    #LENGTH,-(A7)     * array length
        BSR       DO_SUM            * call
        MOVE.L    4(A7), SUM        * result
        ADD.L     #8, A7            * reposition SP
        ...

DO_SUM  MOVEM.L   D0-D1/A2,-(A7)    * save regs
        MOVE.L    16(A7), D1        * array length
        MOVEA.L   20(A7), A2        * array location
        ...                         * loop
        MOVE.L    D0, 20(A7)        * write result
        MOVEM.L   (A7)+,D0-D1/A2    * restores regs
        RTS                         * return
```

Parametri sulla pila: simulazione

Stato della pila:



Codice:

```
START   MOVE.L    #ARRAY, -(A7)     * array location
        MOVE.L    #LENGTH,-(A7)     * array length
        BSR       DO_SUM            * call
        MOVE.L    4(A7), SUM        * result
        ADD.L     #8, A7            * reposition SP
        ...

DO_SUM  MOVEM.L   D0-D1/A2,-(A7)    * save regs
        MOVE.L    16(A7), D1        * array length
        MOVEA.L   20(A7), A2        * array location
        ...                         * loop
        MOVE.L    D0, 20(A7)        * write result
        MOVEM.L   (A7)+,D0-D1/A2    * restores regs
        RTS                         * return
```

Parametri sulla pila: simulazione

Stato della pila:

Codice:

```
START   MOVE.L    #ARRAY, -(A7)     * array location
        MOVE.L    #LENGTH,-(A7)     * array length
        BSR       DO_SUM            * call
        MOVE.L    4(A7), SUM        * result
        ADD.L     #8, A7            * reposition SP
        ...

DO_SUM  MOVEM.L   D0-D1/A2,-(A7)    * save regs
        MOVE.L    16(A7), D1        * array length
        MOVEA.L   20(A7), A2        * array location
        ...                         * loop
        MOVE.L    D0, 20(A7)        * write result
        MOVEM.L   (A7)+,D0-D1/A2    * restores regs
        RTS                         * return
```

16(A7) = $FFFFE8 + $10 = $FFFFF8

Parametri sulla pila: simulazione

Stato della pila:

Codice:

```
START   MOVE.L    #ARRAY, -(A7)     * array location
        MOVE.L    #LENGTH,-(A7)     * array length
        BSR       DO_SUM            * call
        MOVE.L    4(A7), SUM        * result
        ADD.L     #8, A7            * reposition SP
        ...

DO_SUM  MOVEM.L   D0-D1/A2,-(A7)    * save regs
        MOVE.L    16(A7), D1        * array length
        MOVEA.L   20(A7), A2        * array location
        ...                         * loop
        MOVE.L    D0, 20(A7)        * write result
        MOVEM.L   (A7)+,D0-D1/A2    * restores regs
        RTS                         * return
```

20(A7) = $FFFFE8 + $12 = $FFFFF8

Parametri sulla pila: simulazione

Stato della pila:

Codice:

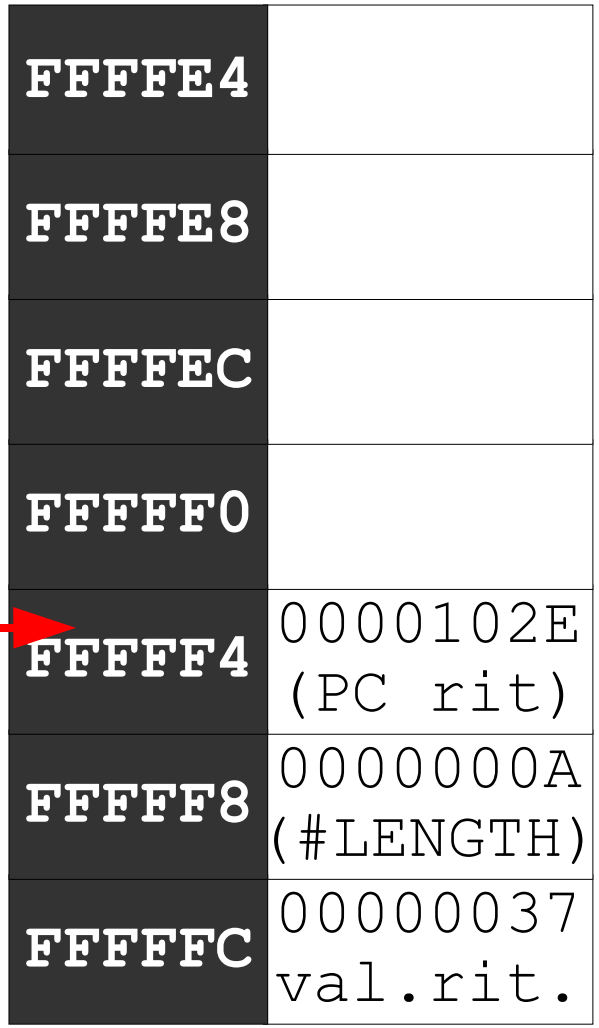
```
START   MOVE.L    #ARRAY, -(A7)      * array location
        MOVE.L    #LENGTH,-(A7)      * array length
        BSR       DO_SUM             * call
        MOVE.L    4(A7), SUM         * result
        ADD.L     #8, A7             * reposition SP
        ...

DO_SUM  MOVEM.L   D0-D1/A2,-(A7)     * save regs
        MOVE.L    16(A7), D1         * array length
        MOVEA.L   20(A7), A2         * array location
        ...                          * loop
        MOVE.L    D0, 20(A7)         * write result
        MOVEM.L   (A7)+,D0-D1/A2     * restores regs
        RTS                          * return
```

20(A7) = $FFFFE8 + $12 = $FFFFFC

Parametri sulla pila: simulazione

Stato della pila:



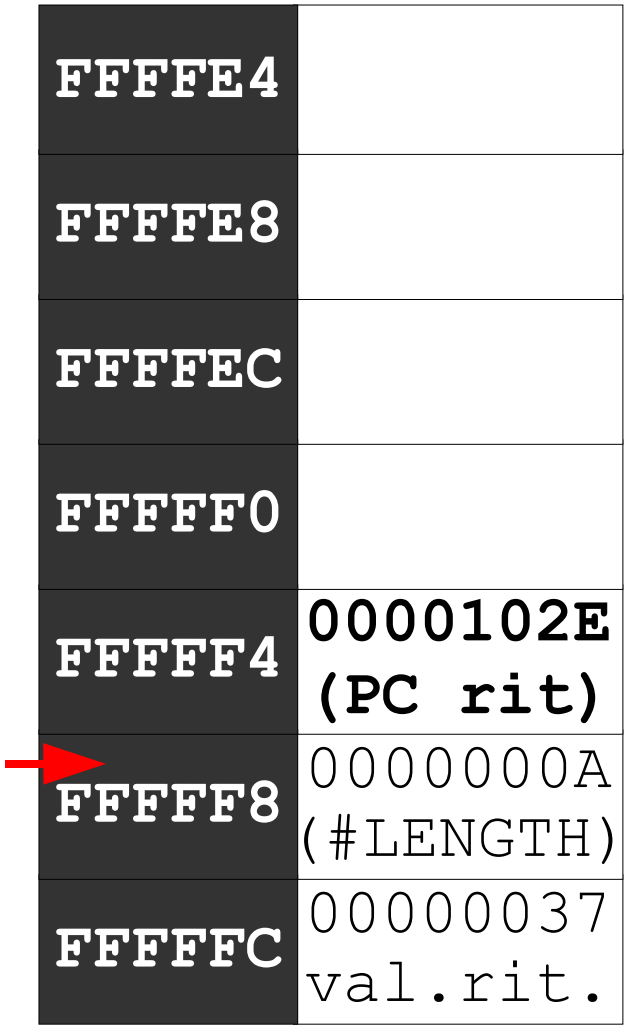
Codice:

```
START   MOVE.L    #ARRAY, -(A7)     * array location
        MOVE.L    #LENGTH,-(A7)     * array length
        BSR       DO_SUM            * call
        MOVE.L    4(A7), SUM        * result
        ADD.L     #8, A7            * reposition SP
        ...

DO_SUM  MOVEM.L   D0-D1/A2,-(A7)    * save regs
        MOVE.L    16(A7), D1        * array length
        MOVEA.L   20(A7), A2        * array location
        ...                         * loop
        MOVE.L    D0, 20(A7)        * write result
        MOVEM.L   (A7)+,D0-D1/A2    * restores regs
        RTS                         * return
```

Parametri sulla pila: simulazione

Stato della pila:



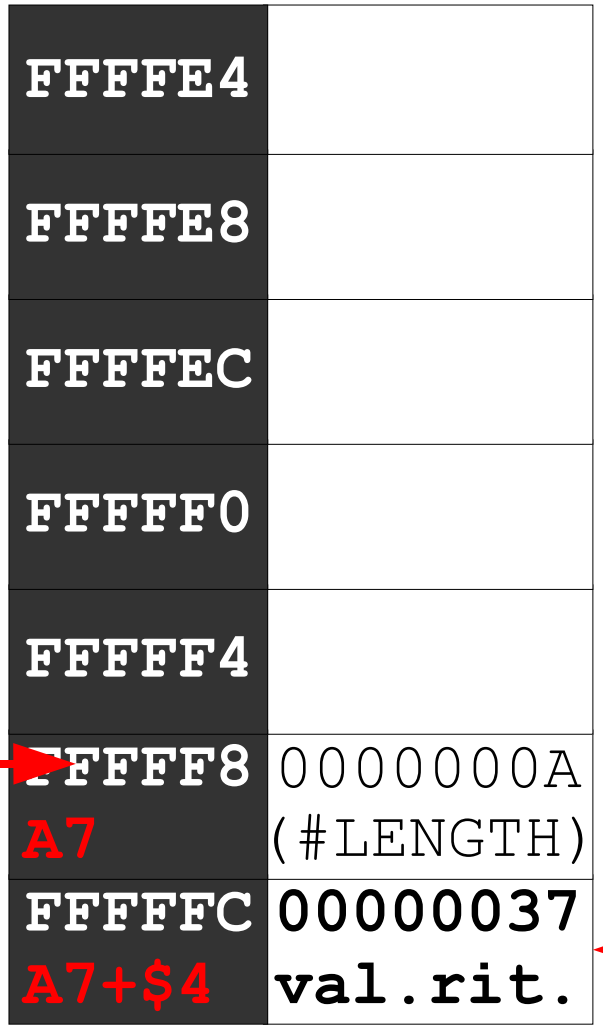
Codice:

```
START   MOVE.L    #ARRAY, -(A7)     * array location
        MOVE.L    #LENGTH,-(A7)     * array length
        BSR       DO_SUM            * call
        MOVE.L    4(A7), SUM        * result
        ADD.L     #8, A7            * reposition SP
        ...

DO_SUM  MOVEM.L   D0-D1/A2,-(A7)    * save regs
        MOVE.L    16(A7), D1        * array length
        MOVEA.L   20(A7), A2        * array location
        ...                         * loop
        MOVE.L    D0, 20(A7)        * write result
        MOVEM.L   (A7)+,D0-D1/A2    * restores regs
        RTS                         * return
```

Parametri sulla pila: simulazione

Stato della pila:



Codice:

```
START   MOVE.L    #ARRAY, -(A7)     * array location
        MOVE.L    #LENGTH,-(A7)     * array length
        BSR       DO_SUM            * call
        MOVE.L    4(A7), SUM        * result
        ADD.L     #8, A7            * reposition SP
        ...

DO_SUM  MOVEM.L   D0-D1/A2,-(A7)    * save regs
        MOVE.L    16(A7), D1        * array length
        MOVEA.L   20(A7), A2        * array location
        ...                         * loop
        MOVE.L    D0, 20(A7)        * write result
        MOVEM.L   (A7)+,D0-D1/A2    * restores regs
        RTS                         * return
```

4(A7) = $FFFFF8 + $4 = $FFFFFC

Parametri sulla pila: simulazione

Stato della pila:

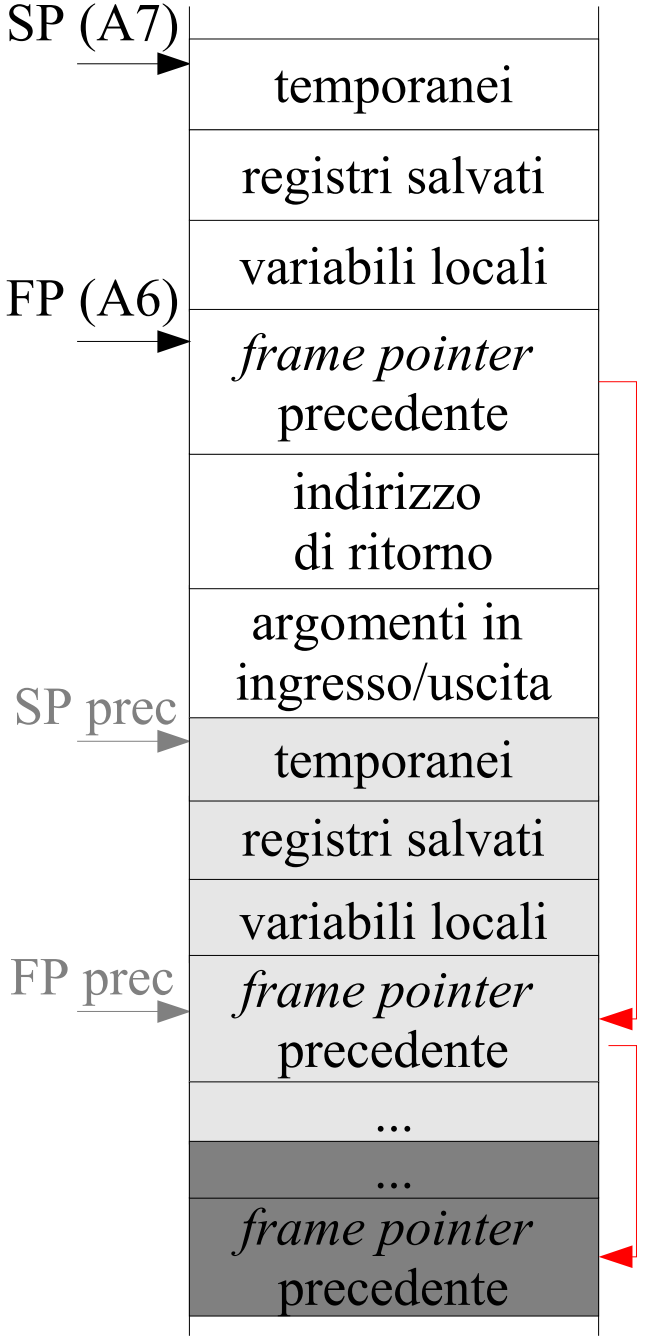
Codice:

```
START   MOVE.L    #ARRAY, -(A7)     * array location
        MOVE.L    #LENGTH,-(A7)     * array length
        BSR       DO_SUM            * call
        MOVE.L    4(A7), SUM        * result
        ADD.L     #8, A7            * reposition SP
        ...

DO_SUM  MOVEM.L   D0-D1/A2,-(A7)    * save regs
        MOVE.L    16(A7), D1        * array length
        MOVEA.L   20(A7), A2        * array location
        ...                         * loop
        MOVE.L    D0, 20(A7)        * write result
        MOVEM.L   (A7)+,D0-D1/A2    * restores regs
        RTS                         * return
```

A7+8 = $FFFFF8 + $8 = $1000000

Uso dei record di attivazione

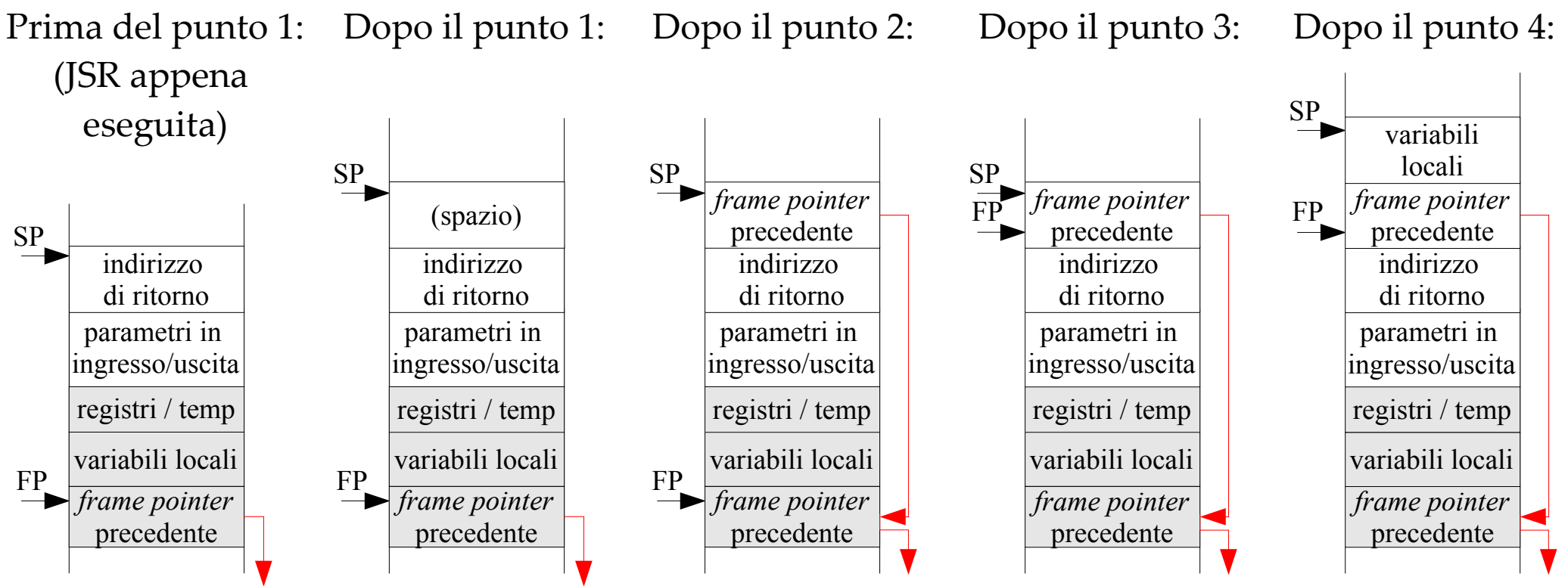


- Una migliore realizzazione del meccanismo di chiamata a sottoprogramma fa uso dei **record di attivazione** (*stack frame*);
- Nel 68000 esistono istruzioni dedicate per allocare e distruggere i record di attivazione: LINK e UNLK;
- I record di attivazione formano una lista concatenata (in rosso nella figura);
- Per convenzione, il puntatore al record di attivazione corrente (*frame pointer)* è nel registro A6;
- Creazione del frame: **`LINK A6, #-N`**
  (N è la dimensione delle variabili locali)
- Distruzione del frame: **`UNLK A6`**
- I parametri hanno spiazzamento **positivo**, esempio: il primo parametro .L sta a **`8(A6)`**
- Le variabili locali hanno spiazzamento **negativo**, esempio: la prima var. locale .L sta a **`-4(A6)`**

Effetto dell' istruzione LINK

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

1.	[SP]	=	[SP] – 4	riservo una long word 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)

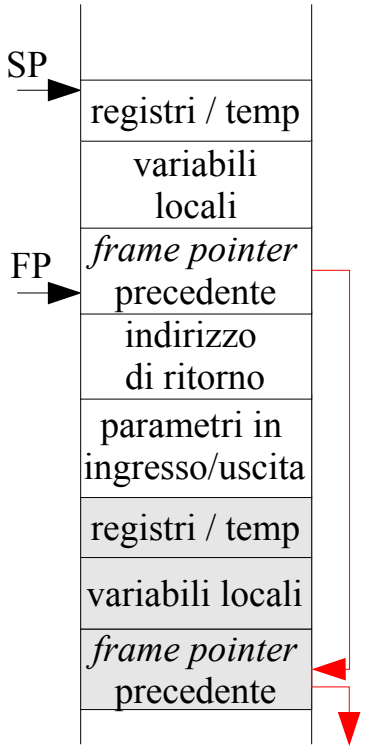


Effetto dell' istruzione UNLK

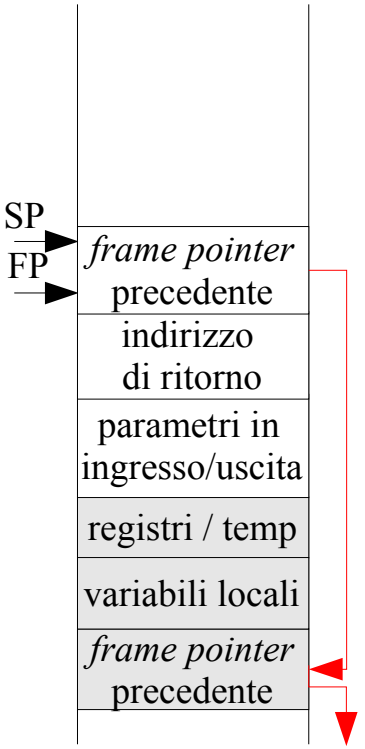
- Istruzione **`UNLK A6`**

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 long word

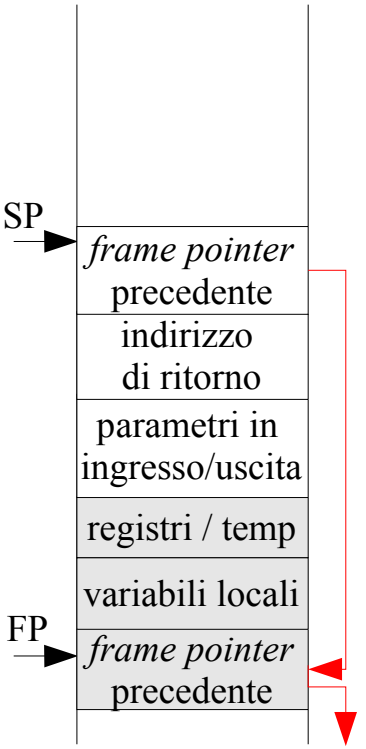
Prima del punto 1:



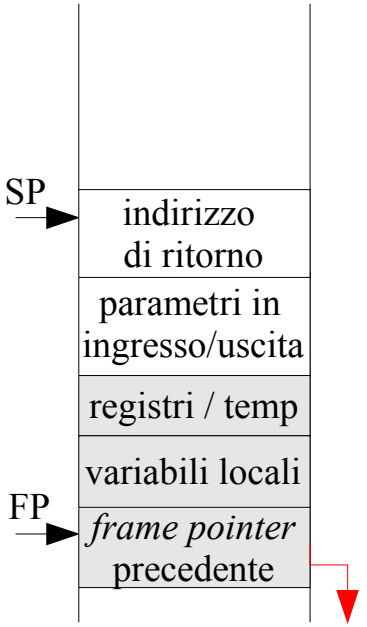
Dopo il punto 1:



Dopo il punto 2:



Dopo il punto 3:
(pronto per RTS)



Schema di funzione in assembly

```
FUNZIONE:
    LINK      A6, #-N            * Riservo spazio per le var locali
    MOVEM.L   D1-...,-(SP)       * Salvo i registri che uso

    ...
    * Preparazione a chiamata di funzione con due parametri
    MOVE.L    ..., -(A7)         * Push secondo parametro
    MOVE.L    ..., -(A7)         * Push primo parametro
    JSR       ALTRA_FUNZIONE     * Chiamata
    ADDQ.L    #8, A7             * Pop parametri
    MOVEQ.L   D0, ...            * Uso il valore di ritorno di ALTRA_FUNZIONE
    ...

    MOVE.L    ..., D0            * Imposto il valore di ritorno di FUNZIONE
    MOVEM.L   (SP)+,D1-...       * Ripristino i registri
    UNLK      A6                 * Elimino var. locali
    RTS                          * Ritorno al chiamante
```

Esempio di traduzione di funzione

Tradurre in assembly di un programma in linguaggio C:

```
int CallingFunction(int x)
{
   int y;
   CalledFunction(1,2);
   return 5;
}


void CalledFunction(int param1, int param2)
{
   int local1, local2;
   local1 = param2;
}
```

Applichiamo le seguenti convenzioni:

- per semplicità non salviamo i registri;
- il chiamato lascia il valore di ritorno in D0;
- il chiamante abbatte la pila dopo la chiamata;
- usiamo la catena dei record di attivazione;

Esempio di traduzione di funzione

Traduzione in assembly del programma di esempio:

```
int CallingFunction(int x)
{
   int y;                  LINK    A6, #-4            * Riservo spazio per y

   CalledFunction(1,2);    MOVE.L  #2, -(A7)          * Push secondo parametro
                           MOVE.L  #1, -(A7)          * Push primo parametro
                           JSR     _CalledFunction    * Chiamata
                           ADDQ.L  #8, A7             * Pop parametri
   return 5;               MOVEQ.L #5, D0             * Valore di ritorno in D0
                           UNLK    A6                 * Elimino var. locali
                           RTS                        * Ritorno al chiamante
}

void CalledFunction(int param1, int param2)
{
   int local1, local2;     LINK    A6, #-8            * Riservo spazio per var.
   local1 = param2;        MOVE.L  12(A6), -4(A6)     * Assegnamento par.>var.
                           UNLK    A6                 * Elimino var. locali
                           RTS                        * Ritorno al chiamante
}
```

Esempio completo

- È a disposizione un esercizio completo che illustra i meccanismi di chiamata a sottoprogramma, interamente svolto.