# Passaggio di Parametri per Valore o Indirizzo

Come in C, l'assembler permette di passare un dato per valore (copia) o per indirizzo, nel secondo caso rendendo modificabile il dato stesso da dentro una subroutine. Ovviamente nei due casi dovrà essere diverso il modo di accedere al dato.

**Passaggio per valore (mediante registro), risultato restituito mediante un registro**

```
            ORG     $4000
Dato:       DC.W    $10

SommaV      ADD     #14,D1
            RTS

Start:      MOVE    Dato,D1
            JSR     SommaV
            END     Start
```

**Passaggio per indirizzo (mediante registro), risultato scritto in memoria**

```
            ORG     $4000
Dato:       DC.W    $10

SommaI:     ADD     #14,(A1)
            RTS

Start:      LEA     Dato,A1
            JSR     SommaI
            END     Start
```

# Passaggio di Parametri mediante Aree Dati

E’ possibile organizzare il programma affinchè il passaggio dei parametri dal programma chiamante ad un sottoprogramma avvenga utilizzando un’area di memoria (diversa dallo stack) che il programma chiamante riempie con i dati (e in cui poi legge i risultati), e da cui il sottoprogramma chiamato legge i dati passati dal chiamante.

La definizione dell’area dati utilizzata può essere fatta in due modi:
• o l’area è fissata a priori e quindi il chiamante sa già dove andare a reperire i dati,
• oppure l’area viene di volta in volta allocata dal chiamante, il quale poi passa al chiamato l’indirizzo dell’area mediante un registro.

In ogni caso, l’area dati per il passaggio dei parametri dovrà sempre prevedere una zona per i parameri di ingresso, ed una zona per i parametri di uscita.

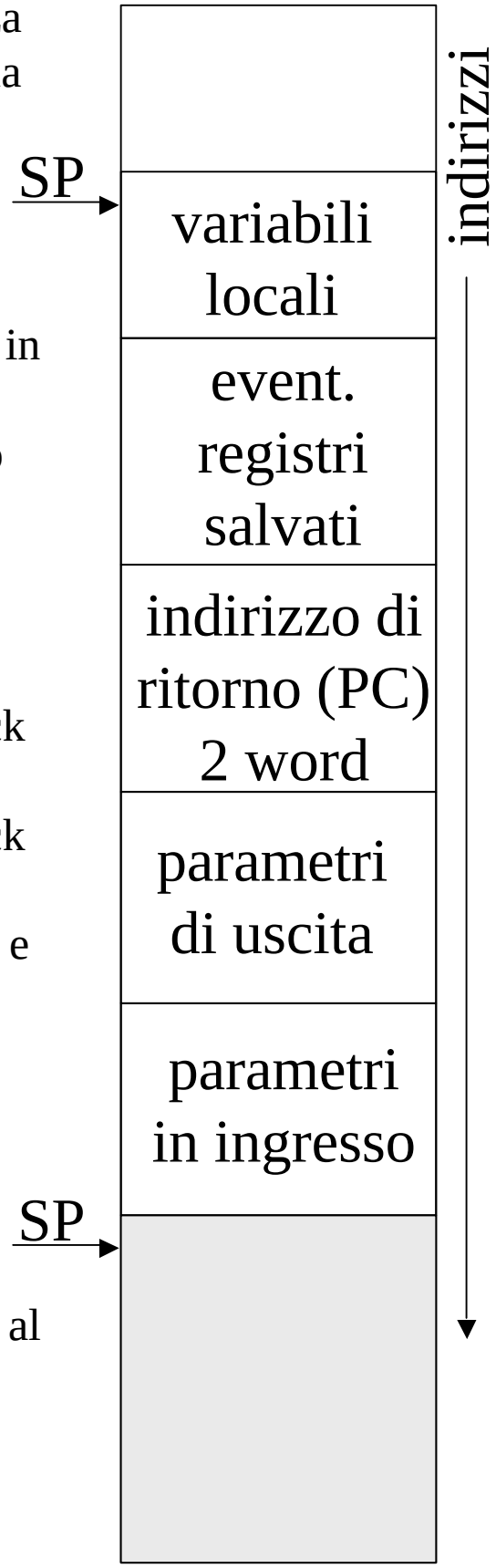
OSS.
Utilizzare una stessa area di memoria come area per il passaggio di parametri ad una subroutine, impedisce di effettuare chiamate ricorsive, perchè sovrascriverebbero i parametri delle chiamate precedenti.

# Passaggio di Parametri mediante Stack di Sistema

Nella chiamata a sottoprogramma che utilizza lo stack per il passaggio dei parametri e per la collocazione delle variabili locali, il programmatore deve seguire le fasi qui indicate:

- Immediatamente prima della chiamata a sottoprogramma JSR, il chiamante aggiunge in cima (puntata da SP) allo stack di sistema i parametri da passare al sottoprogramma, e lo spazio per i risultati da restituire.
- La chiamata a sottoprogramma JSR, aggiunge in cima allo stack l'indirizzo di ritorno dal sottoprogramma (il corrente PC).
- Il sottoprogramma colloca in cima allo stack il valore dei rigestri da salvare.
- Il sottoprogramma colloca in cima allo stack le proprie variabili locali.
- Il sottoprogramma esegue il proprio codice e salva il risultato nello stack, nello spazio allocato dal chiamante.
- Il sottoprogramma libera lo stack dalle variabili locali.
- Il sottoprogramma invoca la RTS, che fa terminare la funzione, carica dallo stack l'indirizzo di ritorno e restituisce il controllo al chiamante.
- Il chiamante utilizza il risultato di ritorno e libera lo stack dai parametri passati al chiamato.

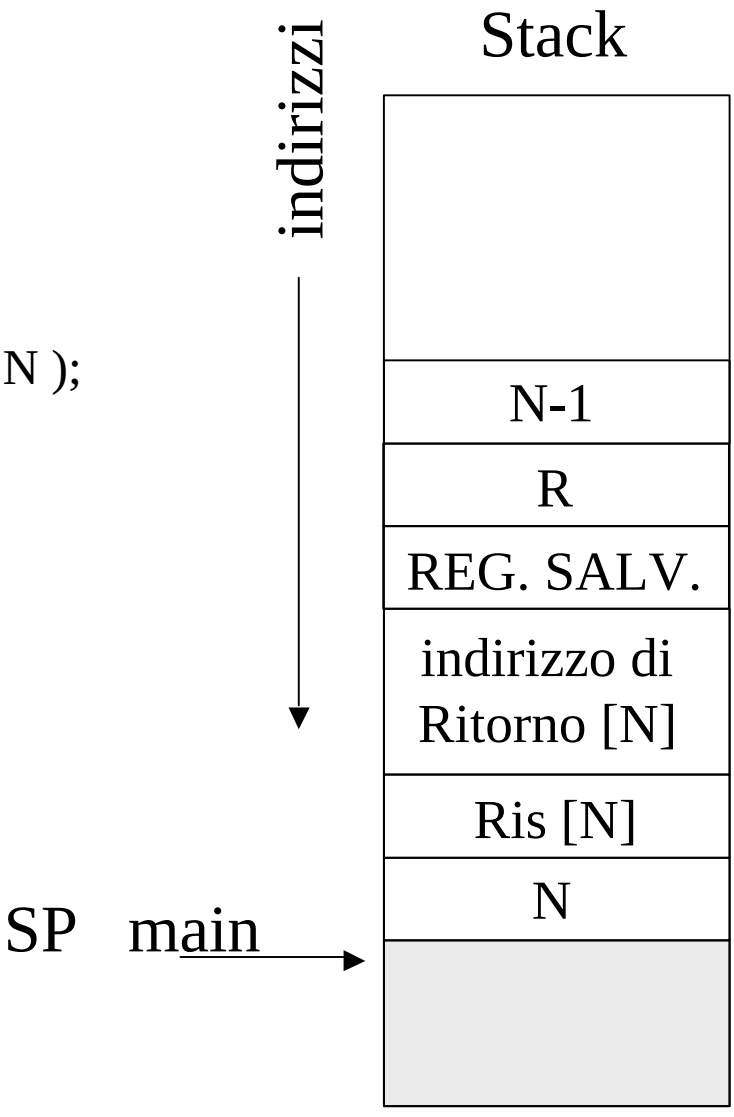
In questo modo è possibile effettuare chiamate ricorsive.



# Esempio di Funzione Ricorsiva: Fattoriale

implementazione C

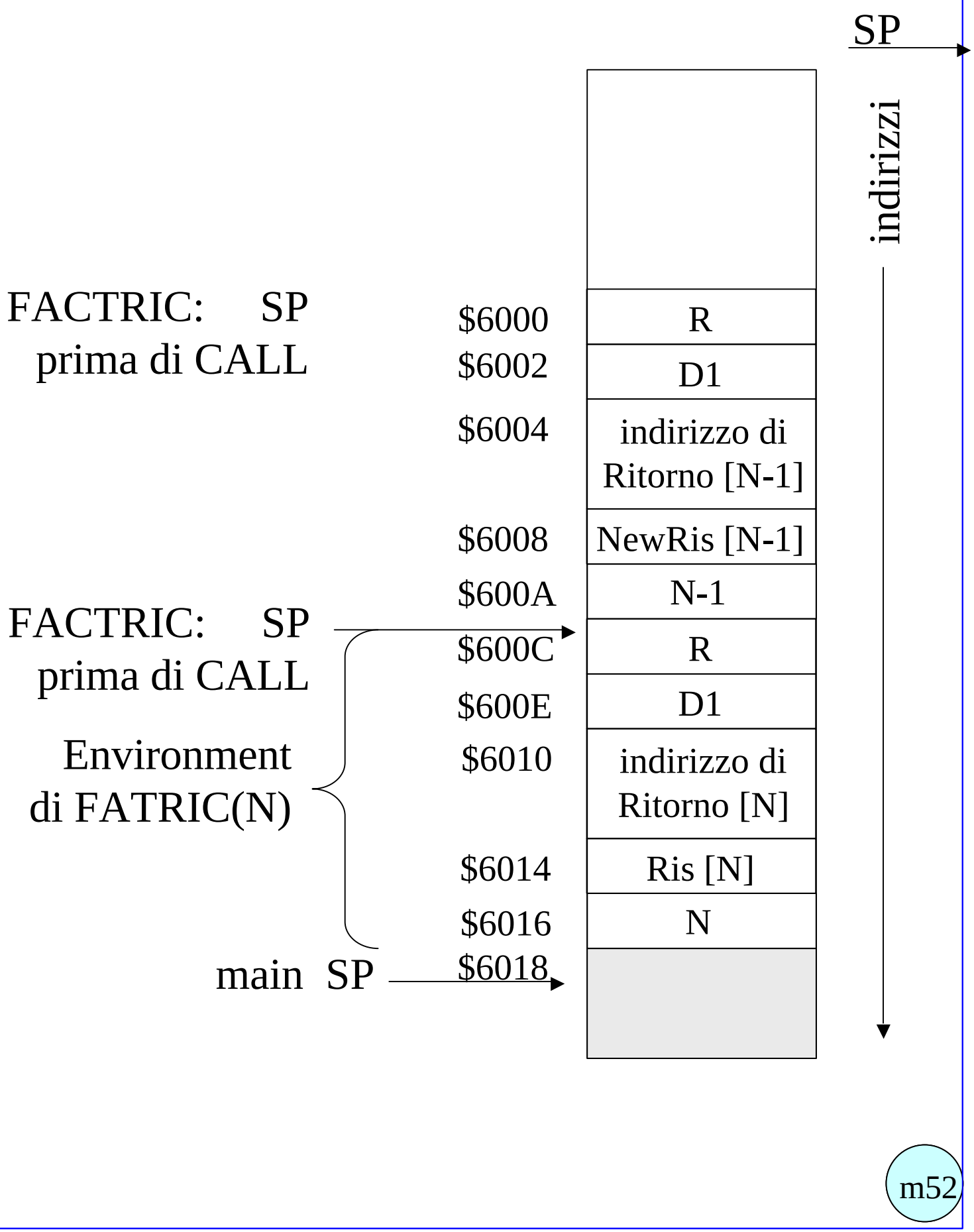
```
int FATRIC(int N)
{
        int R;
        if (N==0)
                R = 1;
        else
                R = N * FATRIC( N-1 );
        return ( R );
}

int   N = 5;
int   RESULT;
void main(void)
{
        RESULT = FATRIC ( N );
}
```



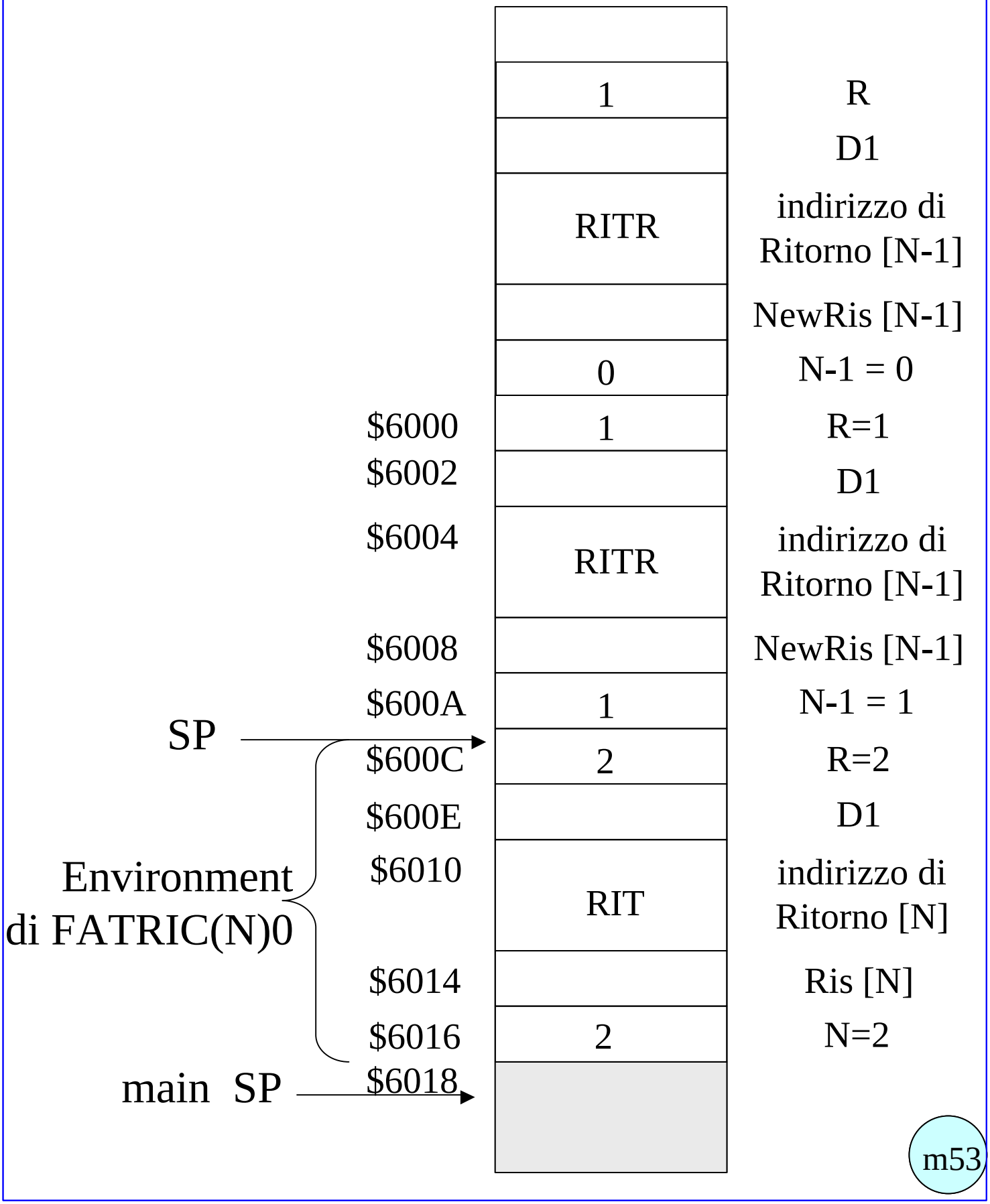
# Esempio di Funzione Ricorsiva: Fattoriale

```
N:          DC.W    $4
RESULT:     DS      $2

            ORG         $1000
FATRIC:     MOVEM   D1,-(SP)    salvo D1
            SUBQ.L  #2,SP       Spazio per R sullo stack
            MOVE    10(SP),D1   N in D1
            BEQ     FINERIC     Se D1 e' zero finisco
RIC:        MOVE    D1,(SP)     N che e' in D1 va in R
            SUBQ    #1,D1       D1 vale N-1
CALL:       MOVEM   D1,-(SP)    N-1 sullo stack
            SUBQ.L  #2,SP       spazio per NewRis
            JSR     FATRIC      chiamata ricorsiva
RITR:       MOVE    (SP),D1     NewRis in D1
            ADDQ.L  #4,SP       elimino NewRis e N-1
            MULU    (SP),D1     D1 = R*NewRis = N*NewRis
            MOVE    D1,(SP)     R = R*NewRis cioe' R=N*NewRis
            BRA     RITORNO
FINERIC:    MOVE    #1,(SP)     1 in R
RITORNO:    MOVE    (SP),8(SP)  R in Ris
            ADDQ.L  #2,SP       elimino R
            MOVEM   (SP)+,D1    ripristino D1
            RTS                 ritorno al chiamante

START:      NOP
            MOVE    N,-(SP)     Spazio per N sullo stack
            SUBQ.L  #2,SP       Spazio per Ris sullo stack
            JSR     FATRIC
RIT:        MOVEM   (SP)+,D1    Ris in D1, ed elimino da stack
            ADDQ.L  #2,SP       elimino N dallo stack
            MOVE    D1,RESULT
FINISH:     STOP    #$2700
            END     START
```

# Esempio di Funzione Ricorsiva



# Stack dopo l'unica esecuzione dell'istruzione all'indirizzo FINERIC (per N=2) (solo i valori interessanti)



# L’istruzione Decrement and Branch Always

Calcola il prodotto scalare di due vettori

```
N          EQU      $00A

           $ORG     $3000
A          DC.W     1,2,3,4,0,9,8,7,6,5
B          DC.W     -1,2,-3,4,-5,6,-7,8,-9,10
C          DS.L     1

           $ORG     $4000
START:     MOVEA.L  #A,A0
           MOVEA.L  #B,A1
           MOVE     #N,D0
           SUBQ     #1,D0
           CLR      D2
LOOP:      MOVE     (A0)+,D1
           MULS     (A1)+,D1
           ADD      D1,D2
           DBRA     D0,LOOP   decrementa D0, salta se >=0
           MOVE     D2,C
           END      START
```

# Esempio, ricerca di un carattere in una stringa

cerca l'indirizzo del token e lo mette in ADDR se lo trova

```
TOKEN   EQU       ':'

            $ORG     $3000
ADDR     DS.L       1
STRING  DC.B      'QUI QUO : QUA'
TAPPO   DC.B      $0

            $ORG     $4000
START:   MOVEA.L   #STRING,A0
            MOVE.B     #TOKEN,D0
LOOP      CMP.B        #0,(A0)
            BEQ           END
            CMP.B.       (A0)+,D0
            BNE           LOOP
            SUBQ.L      #1,A0
            MOVEA.L   #ADDR,A1
            MOVE.L     A0,(A1)
END          START
```

# Esempio, Fibonacci ricorsivo

```
* esercizo di assembler per sistemi 1
* calcolo del numero di fibonacci
* mediante procedura ricorsiva, in cui il
* parametro n viene passato nel registro D0
* lo stack viene usato solo per appoggio,
* per memorizzare il risultato di F(n-1)
* mentre viene calcolato F(n-2)
*
*   F(0) := 1
*   F(1) := 1
*   F(n) := F(n-1) + F(n-2) per n>=2
*

            ORG       $1000
N:          DC.L      $4
RESULT:     DC.L      $0


FIBRIC:     NOP
            CMPI.L    #1,D0
            BGT       RICORS
FINE01:     MOVE.L    #1,D0
            RTS
RICORS:     NOP
            MOVEM.L   D0,-(SP)        salvo N sullo stack
            SUBQ.L    #1,D0
            JSR       FIBRIC
            MOVEM.L   D0,-(SP)        salvo F(N-1) sullo stack
            MOVE.L    4(SP),D0        copio N in D1
            SUBQ.L    #2,D0
            JSR       FIBRIC          F(N-2) -> D1
            ADD.L     (SP),D0         D1 = F(N-1) + F(N-2)
            ADDQ.L     #8,SP          elimino dallo stack N e F(N-1)
            RTS                       ritorno al chiamante

*       Main Program
START:      MOVE.L    N,D0
            JSR       FIBRIC
            MOVE.L    D0,RESULT
FINISH:     STOP      #$2700
            END       START
```

# Esempio, Fibonacci iterativo (1)

```
* esercizo di assembler per sistemi 1, calcolo del numero di fibonacci, procedura iterativa
*   F(0) := 1                F(1) := 1        F(n) := F(n-1) + F(n-2) per n>=2

        ORG     $1000
N:      DC.W    $4
RESULT: DC.W    $0


FIBO:   NOP
TEST0:  CMPI.W  #0,4(SP)
        BEQ     FINE01
TEST1:  CMPI.W  #1,4(SP)
        BNE     OTHER
FINE01: MOVE.W  #1,4(SP)
        RTS
OTHER:  MOVEM.L D0,-(SP)
        MOVEM.L D1,-(SP)
        MOVEM.L A0,-(SP)
        CLR.L   D0
        CLR.L   D1
        MOVE.W  16(SP),D0   N in D0
        MOVE.W  D0,D1       N in D1
        ADDQ.W  #1,D1
        MULU    #2,D1
        SUB.L   D1,SP       USO 2*N byte nello stack
        MOVE.W  #1,(SP)
        MOVE.W  #1,2(SP)
        SUBQ.W  #2,D0       N := N-2
        MOVE.L  SP,A0
LOOP:   MOVE.W  (A0),D1
        ADD.W   2(A0),D1
        MOVE.W  D1,4(A0)
        CMPI.W  #0,D0
        BEQ     ENDLOOP
        ADDQ.L  #2,A0
        SUBQ.W  #1,D0
        BRA     LOOP
```

# Esempio, Fibonacci iterativo (2)

```
ENDLOOP: ADDQ.L   #4,A0
        CLR.L    D1
        MOVE.W   18(A0),D1    salvo N in D1
        ADDQ.W   #1,D1
        MULU     #2,D1
        MOVE.W   (A0),18(A0)  copio F(N)

CTRL:    ADD.L    D1,SP
        MOVEM.L  (SP)+,A0
        MOVEM.L  (SP)+,D1
        MOVEM.L  (SP)+,D0
        RTS                ritorno al chiamante

*        Main Program
START:   NOP
        MOVE.W   N,-(SP)
        JSR      FIBO
        MOVE.W   (SP)+,RESULT
FINISH:  STOP     #$2700
        END      START
```

# Set di Istruzioni del MC68000