LABORATORIO DI ARCHITETTURA DEI CALCOLATORI

lezione n° 16

Prof. Rosario Cerbone

rosario.cerbone@uniparthenope.it

http://digilander.libero.it/rosario.cerbone

a.a. 2007-2008

Passaggio di Parametri mediante Aree Dati

- E' possibile organizzare il programma affinché 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.

Passaggio di Parametri mediante Aree Dati

- La definizione dell'area dati utilizzata può essere fatta in due modi:
  - l'area è fissata a priori e quindi il chiamante sa già dove andare a reperire i dati
  - 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.
- Utilizzare una stessa area di memoria come area per il passaggio di parametri ad una subroutine, impedisce di effettuare chiamate ricorsive, perché sovrascriverebbero i parametri delle chiamate precedenti.

Passaggio di Parametri mediante Aree Dati Procedimento

- I parametri di scambio vengono allocati in un'area di memoria
- Il programma chiamante pone nel registro A0 il base address dell'area di memoria adottata
- La subroutine accede ai parametri usando come based addressing il contenuto di A0, e gli spiazzamenti definiti come costanti:
- **PAR1 EQU 0**
- **PAR2 EQU 4**
- **PAR3 EQU 8**
- **...**

(supponendo che i parametri siano tutti di tipo long word)

Passaggio di Parametri mediante Aree Dati Esempio 1

```
                ORG $8000
ADDN1  EQU      0                   * costanti di spiazzamento
ADDN2  EQU      4                   * dei parametri
ADDN3  EQU      8                   * di ingresso al sottoprogramma
RES    EQU      12                  * spiazzamento del parametro di uscita
* Sottoprogramma
SOMMA MOVE.L    ADDN1(A0),D0        * in D0 il 1° addendo
      ADD.L     ADDN2(A0),D0        * viene sommato al 2° addendo
      ADD.L     ADDN3(A0),D0        * ed al 3° addendo
      MOVE.L    D0,RES(A0)   * la somma viene copiata in memoria
      RTS                           * ritorna al programma principale
* Programma principale
START LEA.L     PARMS,A0    * in A0 pone indirizzo base dell'area parametri
      MOVE.L    A,ADDN1(A0) * A0 punta al 1° addendo
      MOVE.L    B,ADDN2(A0) * A0+4 punta al 2° addendo
      MOVE.L    C,ADDN3(A0) * A0+8 punta al 3° addendo
      JSR       SOMMA               * salta al sottoprogramma
      MOVE.L    RES(A0),D1   * in D1 il parametro di uscita (la somma)
                STOP                #$2700
*Area Dati
                ORG                 $8500
A               DC.L                10
B               DC.L                5
C               DC.L                3
PARMS           DS.L                4
                END                 START
```

Passaggio di Parametri mediante Stack di Sistema

Una stessa area di memoria a stack può essere usata per

- salvare gli indirizzi di ritorno
- passare i parametri di scambio
- memorizzare variabili temporanee

L'area a stack in cui vengono memorizzati i parametri di scambio e l'indirizzo di ritorno è detta *record di attivazione* o **stack frame**

Il sottoprogramma accede al record di attivazione ***mediante indirizzamento indiretto con displacement***, usando un registro A come **frame pointer** (**FP)**

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, si opera nel modo seguente:
  - Immediatamente prima della chiamata a sottoprogramma JSR, il chiamante aggiunge in cima allo stack di sistema (puntata da SP), i parametri da passare al sottoprogramma, e lo spazio per i risultati da restituire.

Passaggio di Parametri mediante Stack di Sistema

- La chiamata a sottoprogramma JSR, aggiunge in cima allo stack l'indirizzo di ritorno dal sottoprogramma (il corrente PC).
- Il sottoprogramma :
  - Sceglie un registro A da usare come Frame Pointer
  - Inizializza FP a SP
  - colloca in cima allo stack il valore dei registri da salvare.
  - colloca in cima allo stack le proprie variabili locali.
  - esegue il proprio codice e salva il risultato nello stack, nello spazio allocato dal chiamante.

Passaggio di Parametri mediante Stack di Sistema

- libera lo stack dalle variabili locali.
- 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.

Passaggio di Parametri mediante Stack di Sistema

- È possibile passare i parametri di input/output in aree organizzate a stack.
- È una forma di allocazione dinamica della memoria.
- Lo stack tende a riempirsi all'ingresso di un sottoprogramma e torna a svuotarsi all'uscita di esso.
- Ogni sottoprogramma utilizzerà solo una serie di locazioni prossime alla testa dello stack, ignorando locazioni più in profondità appartenenti ad altri sottoprogrammi di livello inferiore.

Esempio 2

```
* CALCOLO LOGARITMO CON PARAMETRI NELLO STACK

          ORG $8100
X         DC.W    31
BASE      DC.W    3
RESULT    DS.W    1
START     MOVE.W X,-(SP)          CARICO I PARAMETRI INPUT
          MOVE.W BASE,-(SP)
          SUB.L   #2,SP           ALLOCO SPAZIO PARAMETRI OUT
          JSR LOG
          MOVE.W (SP)+,RESULT     PRELEVO OUT
          ADD.L   #4,SP           ELIMINO PARAMETRI INPUT
          STOP #$2700
```

Esempio 2

```
X_P       EQU     20
BASE_P    EQU     18
RESULT_P  EQU     16              OFFSET PER I PARAMETRI SULLO STACK
INDEX     EQU     2
POT       EQU     0
LOG       MOVEM.L D0-D1,-(SP)             SALVO REG.
          SUB.L   #4,SP                   SPAZIO VAR. LOCALI
          MOVE.W  X_P(SP),D0              ACQ. PARAM.
          MOVE.W  BASE_P(SP),D1
          MOVE.W  #1,INDEX(SP)
          MOVE.W  D1,POT(SP)              INIZIALIZZO POT. A BASE
CONFRONTA CMP.W   POT(SP),D0
          BLE FINE
          MULU.W  POT(SP),D1
          MOVE.W  D1,POT(SP)
          MOVE.W  BASE_P(SP),D1
          ADDQ.W  #1,INDEX(SP)
          BRA CONFRONTA
FINE      MOVE.W  INDEX(SP),RESULT_P(SP)  RIS=INDEX
          ADD.L   #4,SP                   ELIMINA VAR. LOCALI
          MOVEM.L (SP)+,D0-D1             RIPR. REGISTRI
          RTS
          END     START
```

Passaggio di Parametri mediante Stack di Sistema

- Perché un sottoprogramma possa invocare ricorsivamente se stesso, occorre che sia i parametri effettivi, sia le variabili locali siano allocate in un'area di memoria (detta *record di attivazione*) definita al momento della chiamata al sottoprogramma.
- Per facilitare la allocazione sullo stack del record di attivazione di un sottoprogramma, i progettisti del 68000 hanno dotato il processore delle istruzioni LINK A,*im e* UNLK A.
- L'istruzione LINK salva il contenuto del registro A sullo stack, carica in A il valore aggiornato dello Stack Pointer, ed infine incrementa SP dell'offset *im*. Sommando il valore negativo *im* al contenuto di SP, l'istruzione LINK riserva un'area di memoria di *im* byte sulla cima dello stack. Quest'area è utilizzata per l'allocazione delle variabili locali del sottoprogramma.
- L'istruzione UNLK A ripristina il contenuto del registro A e lo stato dello stack.

Link and allocate

- Syntassi:
- LINK An,#<displacement> (word)
- Funzionamento:
  - Esegue push su stack del contenuto del registro indirizzo specificato
  - Il registro indirizzo specificato viene caricato con il nuovo valore dello stack pointer
  - Il displacement viene esteso in segno e sommato a SP. Questo valore viene assegnato a SP.
  - Durante l'esecuzione SP varia, mentre FP rimane costante

UNLK Unlink and deallocate

Sintassi: **UNLK An**

- Funzionamento:
  - Ripristina il contenuto originario del registro An
  - Ripristina lo stato dello stack

Esempio 3

- Calcolare la somma di due numeri (C = A + B) utilizzando una subroutine SOMMA per l'elaborazione principale.
- I parametri di scambio alla subroutine vengano passati attraverso lo stack

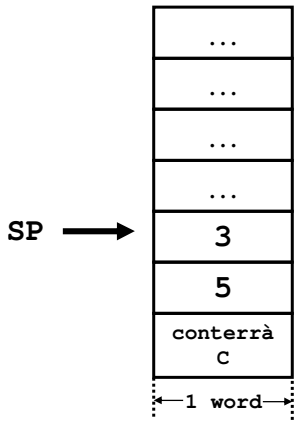
Esempio 3

```
        ORG     $8000
START   MOVE.W  #3,D0       Copia A in D0
        MOVE.W  #5,D1       Copia B in D1
        SUBQ.L  #2,SP       Alloca sullo stack una word per C
        MOVE.W  D1,-(SP)    Push B sullo stack
        MOVE.W  D0,-(SP)    Push A sullo stack
        JSR     SOMMA       Salta alla subroutine
        MOVE.W  4(SP),D2    Copia il risultato in D2
        ADD.L   #6,SP       Dealloca lo spazio
                            sullo stack
        MOVE.W  D2,RES      Copia il risultato in memoria
        STOP    #$2700      Termina
* Area dati
        ORG     $8800
RES     DS.W    1           Riserva una word per il risultato
```

Esempio 3 - Sottoprogramma

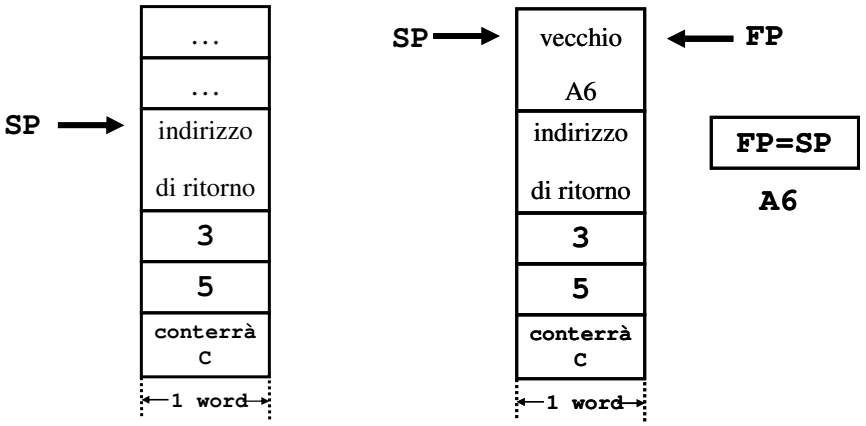
```
        ORG     $8400
OFFA    EQU     8           Offset di A rispetto a FP
OFFB    EQU     10          Offset di B rispetto a FP
OFFC    EQU     12          Offset di C rispetto a FP
SOMMA   LINK    A6,#0       Imposta il frame-pointer in A6:
                            * non alloca variabili locali
        MOVE.L  D0,-(SP)    Push D0 per poterlo utilizzare
        MOVE.W  OFFA(A6),D0 Copia A in D0
        ADD.W   OFFB(A6),D0 Addiziona B a D0
        MOVE.W  D0,OFFC(A6) Salva il risultato sullo stack
        MOVE.L  (SP)+,D0    Ripristina il registro D0
        UNLK    A6          Ripristina il registro A6
        RTS                 Ritorna al programma chiamante
        END     START
```

Esempio 3 - Stato dello stack



prima della chiamata
al sottoprogramma

Stato dello stack

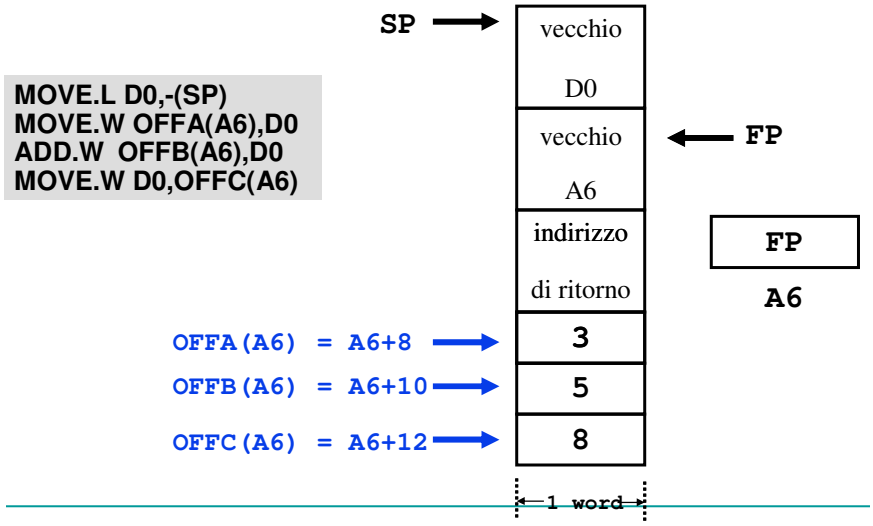


prima
dell'istruzione LINK

dopo dell'istruzione
LINK

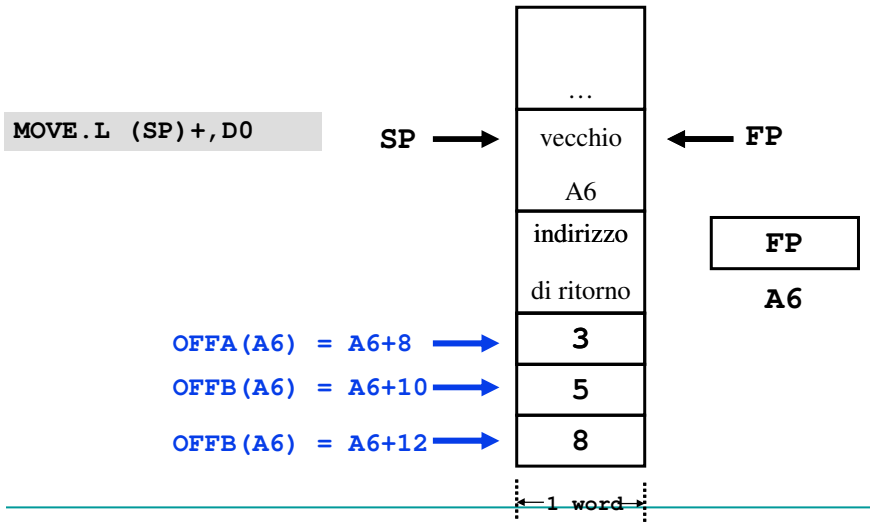
Stato dello stack

```
MOVE.L D0,-(SP)
MOVE.W OFFA(A6),D0
ADD.W  OFFB(A6),D0
MOVE.W D0,OFFC(A6)
```



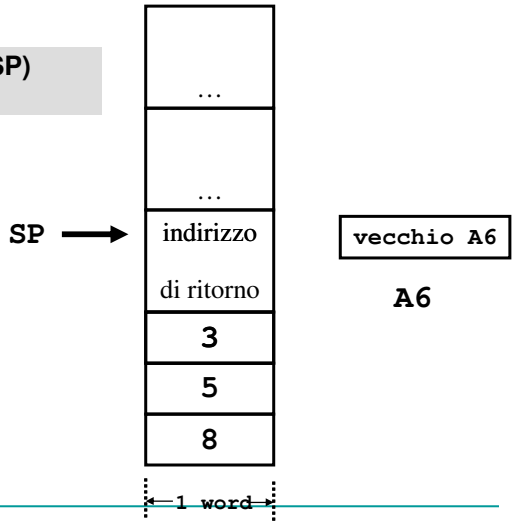
Stato dello stack

```
MOVE.L (SP)+,D0
```



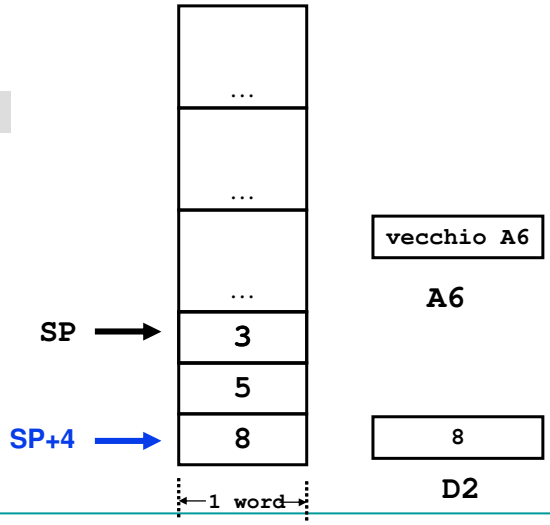
Stato dello stack

```
UNLK    A6  *SP = A6; A6 = -(SP)
RTS
```



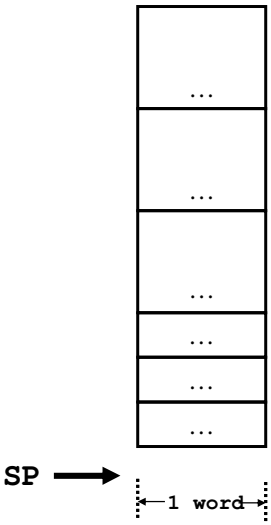
Ritorno al programma chiamante

```
JSR     SOMMA
MOVE.W 4(SP),D2
```



Ritorno al programma chiamante

```
JSR       SOMMA
MOVE.W    4(SP),D2
ADD.L     #6,SP
MOVE.W    D2,RES
STOP      #$2700
```



Gestione dello stack: conclusioni e osservazioni

- L'utilizzo dello stack per il passaggio dei parametri effettivi ad un sottoprogramma è una forma di allocazione dinamica usata sia da programmatori assembler che dai compilatori di linguaggi di alto livello come il Pascal ed il C.
- E' possibile pensare ad una convenzione per il passaggio dei parametri: tale convenzione assegna al chiamante la responsabilità di allocare sullo stack lo spazio richiesto dai parametri di output e nel caricare successivamente sullo stack i valori dei parametri di input.
- Il sottoprogramma crea uno *stack frame* mediante l'istruzione LINK, che inizializza un *frame pointer* ed alloca sullo stack spazio per le variabili locali. Il sottoprogramma può accedere ai parametri di scambio ed alle variabili locali mediante indirizzamento indiretto attraverso il *frame pointer*.
- Per l'accesso ai parametri di scambio di ingresso/uscita si usano valori di spiazzamento positivi. Eventuali variabili locali del sottoprogramma dovrebbero essere allocate nell'area al di sopra del *frame pointer*, e quindi sarebbero accedute mediante displacement a valori negativi.

Esempio 4

```
Calcola Z = X^Y mediante sottoprogramma POWR
*****************************************************************
* Programma principale
        ORG     $8000
MAIN            ADDA.L  #-2,SP          riserva 2 byte per Z
                MOVE    Y,-(SP)         push esponente Y (word)
                MOVE    X,-(SP)         push base X (word)
                JSR     POWR            chiama subroutine POWR
                ADDQ    #4,SP           rimuovi X e Y dallo stack
                MOVE    (SP)+,Z         copia parametro di output
                STOP    #$2700

* Allocazione variabili X, Y e Z
X               DC.W    3
Y               DC.W    4
Z               DS.W    1
*****************************************************************
```

Esempio 4

```
* Subroutine POWR: calcola A^B
* Definizione di costanti di spiazzamento per l'accesso
* ai parametri di scambio e alle variabili locali
OLD_FP          EQU     0
RET_ADDR        EQU     4
A               EQU     8               spiazzamento base
B               EQU     10              spiazzamento esponente
C               EQU     12              spiazzamento risultato
POWR            LINK    A6,#0           usa A6 come frame pointer
                MOVEM.L D0-D2,-(SP)     salva registri su stack
                MOVE    A(A6),D0        copia A in D0
                MOVE    B(A6),D1        copia B in D1
                MOVE.L  #1,D2           usa D2 come accumulatore
LOOP            SUBQ    #1,D1           decrementa B
                BMI.S   EXIT            if D1-1<0 then EXIT
                MULS    D0,D2           moltiplica D2 per A
                BRA     LOOP            e ripeti se necessario
EXIT            MOVE    D2,C(A6)        C:=(D2)
                MOVEM.L (SP)+,D0-D2     ripristina registri
                UNLK    A6
                RTS
                END     MAIN
```

Esercizio 16.1

- Eseguire il debug degli esempi precedenti con Asim e ricavare l'utilizzo dello stack, riportando graficamente le locazioni di memoria e le variabili in esse contenute.

Esercizio 16.2

- Scrivere un programma che esegua il fattoriale di un numero utilizzando una subroutine.
- Riportare graficamente lo stack e le sue eventuali variazioni nel corso del programma (prima e dopo le chiamate a subroutine,ecc.).

Esercizio 16.3

- **Calcolare il prodotto degli elementi dispari di un vettore di interi attraverso un sottoprogramma PRODDISP, che riceve l'indirizzo del vettore VETT (passaggio per indirizzo) ed il valore del suo riempimento come parametri sullo stack e memorizza il prodotto nella variabile PROD allocata nel programma principale.**
- **Si provi il programma con il vettore { -1, 5, 6, -3, -7, 4, 2 }.**

Esercizio 16.4

- Scrivere un programma per la ricerca di un elemento in un vettore di interi. Si utilizzi una subroutine per l'elaborazione. Alla subroutine si passi l'indirizzo di partenza del vettore, il numero di elementi ed il valore che si intende ricercare all'interno di esso.
- La subroutine restituisca l'indirizzo dell'elemento se esistente, 0 altrimenti.
- Il passaggio dei parametri avvenga attraverso lo stack.

Esercizio 16.5

- Scrivere un programma per la ricerca di un elemento in una matrice di interi M(i,j). Si utilizzi una subroutine per l'elaborazione. Alla subroutine si passi l'indirizzo di partenza della matrice, gli indici, ed il valore che si intende ricercare all'interno di esso.
- La subroutine restituisca gli indici (x,y) dell'elemento se esistente, 0 altrimenti.
- Il passaggio dei parametri avvenga attraverso lo stack.