Corso di Calcolatori Elettronici I
A.A. 2010-2011

Subroutine in linguaggio macchina: collegamento e passaggio dei parametri

Lezione 26 seconda parte

Prof. Roberto Canonico

Università degli Studi di Napoli Federico II
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica (allievi A-DE+Q-Z)
Corso di Laurea in Ingegneria dell'Automazione

Sottoprogrammi: richiami

- Sottoprogramma o subroutine: è un segmento di codice che svolge un'elaborazione su dati forniti in input e restituisce eventualmente risultati in output
- Realizza un'astrazione procedurale, fornendo un servizio attraverso un'interfaccia
- Consente un'organizzazione modulare del codice

Sottoprogrammi: parametri

- Un sottoprogramma scambia con il programma chiamante dei dati sia in input che in output
- Le informazioni scambiate sono dette *parametri*
- Nel testo del sottoprogramma i parametri sono riferiti mediante dei nomi simbolici: *parametri formali*
- L'attivazione (o invocazione) di un sottoprogramma richiede che ai parametri formali siano assegnate degli opportuni valori: *parametri effettivi*

Sottoprogrammi: problematiche

1. **Collegamento** (o **linkage**) - Modo in cui un calcolatore rende possibili le operazioni di **chiamata** e di **ritorno** delle procedure
2. **Passaggio dei parametri** - Modo in cui il programma chiamante rende disponibili le **informazioni di ingresso** alla subroutine e la subroutine rende disponibili al programma chiamante le **informazioni di uscita**

Sottoprogrammi: procedure e funzioni

- In alcuni linguaggi di alto livello (Pascal) si distingue esplicitamente tra
  - "funzioni": sottoprogrammi al cui nome è associato un valore "di ritorno" assegnabile ad una variabile, es. a := max(b, c);
  - "procedure", che invece compiono solo una elaborazione sui parametri di scambio ma non hanno un valore di ritorno, es. stampa(vettore, n);
- In C tutti i sottoprogrammi sono detti "funzioni", e le procedure hanno un dato di ritorno di tipo *void*

Gestione dei sottoprogrammi in I/m

- Per consentire l'organizzazione del codice assembly in sottoprogrammi, occorrono:
  - meccanismi per il collegamento tra programma chiamante e sottoprogramma
  - opportune convenzioni sulle modalità di scambio dei parametri, che definiscano:
    - quali informazioni si scambiano il programma chiamante ed il sottoprogramma ("cosa")
    - attraverso quali meccanismi avviene lo scambio (memoria, registri, …) ("come")

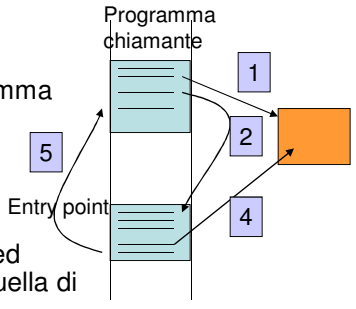
Sottoprogrammi: scambio dei parametri

- Nei linguaggi procedurali di alto livello
  - "cosa" va passato, ovvero l'elenco ed il tipo dei parametri di scambio viene definito nella dichiarazione della procedura
    - Es: `void prod (int a, int b, int *p)`
  - "come" avviene lo scambio è convenuto dal linguaggio: la convenzione è implementata dal compilatore
- Se si programma in assembler, è il programmatore che stabilisce ed implementa le convenzioni
  - a meno che non si debba scrivere moduli assembler che interagiscono con moduli C o di altro linguaggio di alto livello: in tal caso occorre che il programmatore rispetti le convenzioni del compilatore

Sottoprogrammi: esecuzione

L'esecuzione di un sottoprogramma avviene mediante i seguenti passi:

1. Scrittura dei parametri effettivi secondo la convenzione stabilita per lo scambio
2. Trasferimento del controllo al sottoprogramma ("chiamata")
3. Esecuzione del sottoprogramma
4. Scrittura dei parametri di output
5. Trasferimento del controllo al programma principale ("ritorno dal sottoprogramma") ed esecuzione dell'istruzione successiva a quella di chiamata



Chiamata di un sottoprogramma

- La chiamata di un sottoprogramma interrompe l'esecuzione sequenziale delle istruzioni del programma chiamante e pertanto si presenta come un caso particolare di salto
  - L'indirizzo della prima istruzione della subroutine (*entry point*) deve essere caricato nel *Program Counter*
- A differenza di un normale salto, occorre prevedere un meccanismo per il ritorno al chiamante
- Esempi di istruzioni per la chiamata di subroutine:
  - Motorola 68000: `jsr label` – `bsr label`
  - Intel 8086: `call label`
  - PowerPC e MIPS: `jal label`

Collegamento

- Il problema di consentire il ritorno dal sottoprogramma al programma chiamante, o meglio all'istruzione del programma chiamante successiva alla chiamata, è detto problema di *collegamento del sottoprogramma* (*subroutine linkage*)
- Tre soluzioni sono possibili:
  - L'indirizzo di ritorno è salvato in un registro di macchina
  - L'indirizzo di ritorno è salvato in una locazione di memoria particolare (ad esempio, nelle locazioni immediatamente precedenti l'entry point della subroutine)
  - L'indirizzo di ritorno è salvato sullo stack

Collegamento: link register

- In alcune architetture (PowerPC, MIPS) un'istruzione di *jump&link* carica l'indirizzo di ritorno in un registro del processore (*Link Register*) e successivamente effettua il salto:
  - `jal label` → $ra := [PC]; PC := label
- Un'altra istruzione effettua l'operazione reciproca, il caricamento nel PC del valore salvato nel registro di link:
  - `jr $ra` → PC := [$ra]
- Il nesting delle subroutine e la ricorsione richiedono il salvataggio del link register su uno stack

Collegamento: salvataggio in area di memoria

- Si può stabilire una convenzione, per la quale l'indirizzo di ritorno è salvato nelle locazioni precedenti l'entry point della subroutine
- Es.:

```
MAIN   MOVE.L #RET1,SUBR
       JMP           SUBR+4
RET1   ...
…
SUBR   DS.L          1
       … codice subroutine …
       MOVEA.L       SUBR,A0
       JMP           (A0)
```

- Inconvenienti: non può essere usata se il codice della subroutine è in ROM; non consente il nesting, la ricorsione, …

Collegamento mediante stack

- In alcuni processori (es. il Motorola 68000 e nella famiglia Intel x86) l'istruzione di salto a subroutine automaticamente salva l'indirizzo di ritorno sulla cima dello stack di sistema
  - `jsr subr` → pushm(SP, PC); PC:= subr
- Reciprocamente, un'istruzione di ritorno da subroutine consente il ripristino dell'indirizzo di ritorno dalla cima dello stack
  - `rts` → popm(SP, PC)

Convenzioni per il passaggio dei parametri

- I parametri effettivi su cui un sottoprogramma deve operare possono essere scambiati mediante:
  - Registri del processore
  - Un'area di memoria associata al chiamante
  - Un'area di memoria associata al sottoprogramma
  - L'uso di uno stack

Passaggio dei parametri mediante registri

- E' la tecnica più veloce: non richiede accessi in memoria
- Occorre che chiamante e sottoprogramma si accordino sull'uso dei registri
- Il numero di registri generali del processore rappresenta un limite

Passaggio dei parametri mediante un'area di memoria

- Sono possibili due varianti:

1. Il codice della subroutine definisce un'area di memoria usata per lo scambio dei parametri
2. Il programma chiamante alloca un'area di memoria per lo scambio dei parametri e ne passa l'indirizzo iniziale al sottoprogramma, il quale accede ai parametri effettivi mediante un opportuno spiazzamento rispetto all'indirizzo base (ad es., usando un modo di indirizzamento indiretto con displacement)

Esempio

```
A       EQU             0
B       EQU             4
C       EQU             6
        ORG             $6000
AREA    DS.L            1
        DS.W            1
        DS.W            1
        ORG             $8000
MAIN    …
        MOVEA.L         #AREA,A0
        … scrive i parametri effettivi in area …
        JSR             SUBR
        …
SUBR    MOVE.L A(A0),D0
        MOVE.W B(A0),D1
        MOVE.W C(A0),D2
```

Passaggio parametri mediante stack

- La tecnica di scambio dei parametri che consente in modo naturale il nesting e la ricorsione utilizza un'area di memoria allocata dinamicamente (record di attivazione) per la memorizzazione dei parametri effettivi e delle variabili locali del sottoprogramma
- Tipicamente il record di attivazione è allocato dinamicamente sulla cima dello stack, in quanto la tecnica LIFO di accesso ben si adatta al ciclo di vita delle procedure innestate o ricorsive (l'ultima invocazione è la prima a ritornare)

Uso dello stack per la memorizzazione del record di attivazione

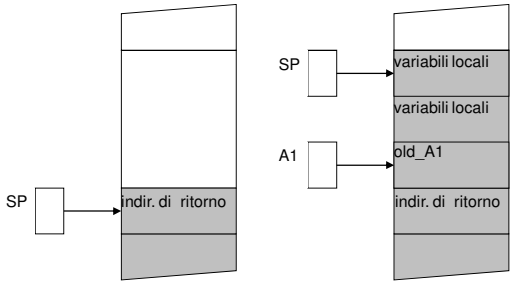
- Per fissare un riferimento fisso al record di attivazione, da usare nel sottoprogramma per accedere ai parametri di scambio, si crea un *Frame Pointer (FP)*, ovvero si assegna ad un registro del processore l'indirizzo della cima dello stack all'entrata della subroutine
- Successivamente, la subroutine può fare uso dello stack per variabili locali e quindi lo Stack Pointer può ulteriormente spostarsi, mentre gli spiazzamenti dei campi del record di attivazione rispetto al Frame Pointer rimangono costanti

Istruzione LINK del 68000

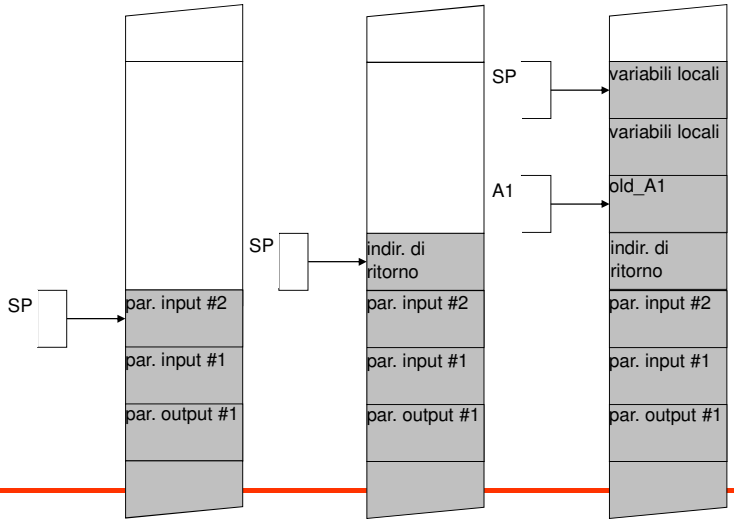
- Per facilitare la gestione del record di attivazione sullo stack, il processore Motorola 68000 dispone di un'istruzione: `LINK A,#im`
  - pushm(SP,A); A := [SP]; SP := [SP] + im;
- salva il contenuto del registro A sullo stack, carica in A il valore aggiornato dello Stack Pointer, ed infine incrementa 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.
- Dualmente: `ULINK A` effettua SP := [A]; popm(SP,A);

LINK e ULINK

- Stato dello stack prima e dopo LINK
- L'istruzione ULINK A ripristina in A il valore puntato da A (old_A) e



Uso dello stack per il passaggio dei parametri



Esempio

```
* Calcola Z = X^Y mediante sottoprogramma POWR
*********************************************************
* Programma principale
          ORG       $1000
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
          JMP       $8008   ritorna al sist.operativo
* Allocazione variabili X, Y e Z
X         DC.W      3
Y         DC.W      4
Z         DS.W      1
```

Esempio (cont.)

```
•  Subroutine POWR: calcola A^B
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
```