Attivazione di subroutine

Argomento:

- L'istruzione di chiamata a subroutine
- Il passaggio dei parametri

Materiale didattico:

- Capitolo 12 [S16], sezione "Procedure Call Instructions (da pag. 459)

Chiamata a procedura

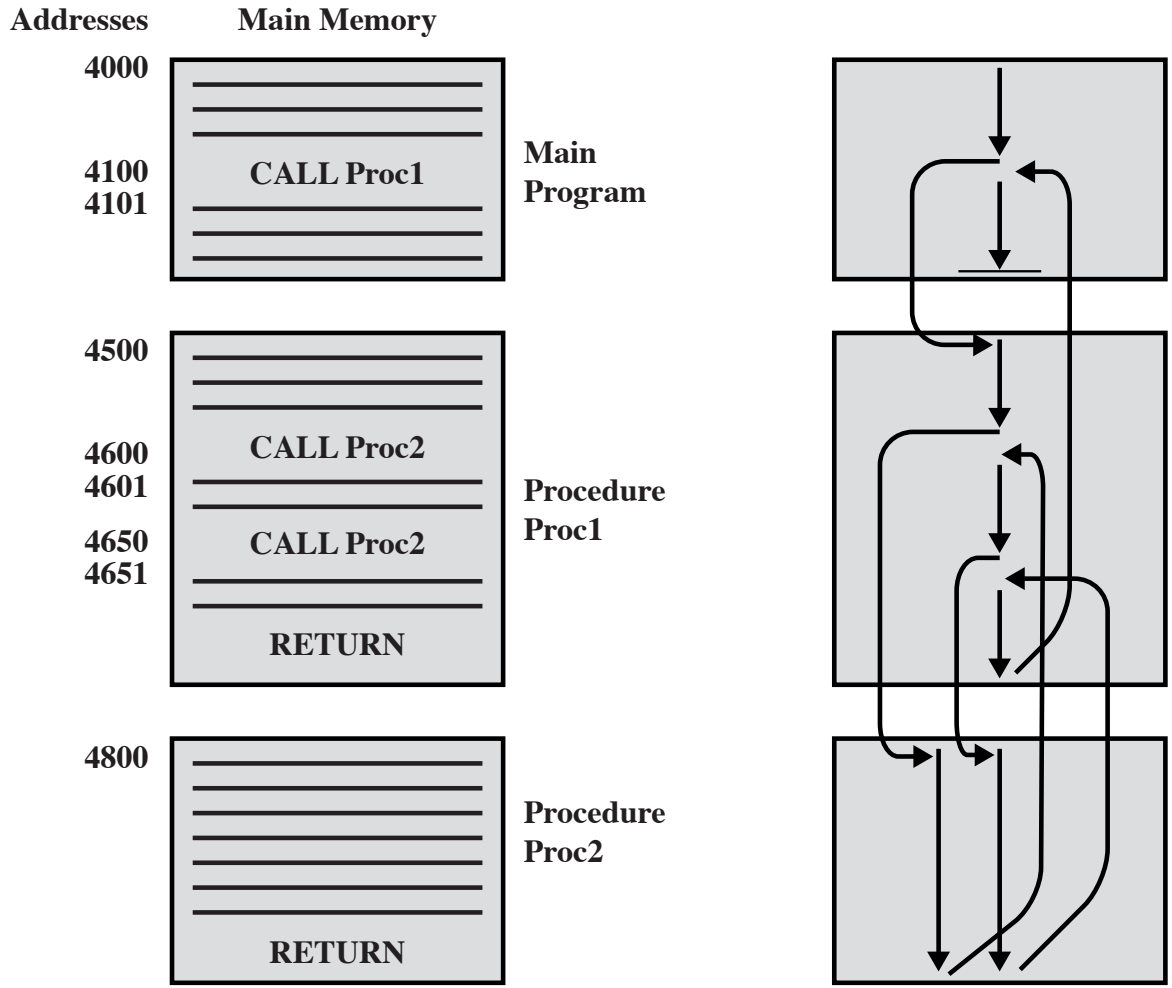
- Una **procedura/subroutine** è un sequenza di istruzioni che può essere invocata da un altro programma
  - Può essere invocata/chiamata in più punti del programma
  - Il processore esegue il codice della procedura e poi ritorna al punto da cui la chiamata è partita
- Motivi principali per l'utilizzo di procedure:
  - Permette di riutilizzare codice
  - Permette una struttura modulare

Chiamata a procedura (2)

- Il meccanismo di chiamata a procedura richiede:
  - Un'istruzione per chiamare la procedura
  - Un'istruzione per ritornare al punto iniziale

- Il punto di ritorno al programma è ogni volta diverso: è **l'istruzione successiva a quella di chiamata**

Chiamata a procedura (3)



(a) Calls and returns

(b) Execution sequence

Procedure in linguaggi ad alto livello

Passaggio parametro per riferimento

Passaggio parametro per valore

```
int vect_sum (int* vect, int n){
        int tmp = 0;
        for(int i=0; i<n; i++) tmp = tmp+i;
        return tmp;
}
```

Firma di una funzione

Definizione della funzione

Termine funzione e ritorno valore di output

```
int main(int argc, char **argv){
        int n = 10;
        int* v = malloc(n*sizeof(int));
        // ... assegna valori al vettore
        int result = vect_sum(v,n);
        printf("%d", result);
}
```

Chiamata della funzione

Problematiche con subroutine in assembly

1. Come definire una funzione?

2. Come ritornare al punto di partenza?
   - E' necessario che la subroutine disponga dell'**indirizzo di ritorno**

2. Come passare eventuali parametri?
   - i dati su cui la subroutine deve operare sono i **parametri di input**;
   - I risultati restituiti dalla subroutine sono i **parametri di output.**

Definire una funzione

Definire una funzione

- Una funzione viene indicata in modo univoco dall'indirizzo in memoria contenente la prima istruzione
- In linguaggio assembly, si utilizza una label per definire l'indirizzo di inizio della funzione (perché l'indirizzo non è ancora noto)

```
FUN:  MOV R0, #0 @ prima istruzione della funzione FUN
      CMP R1, R0 @ seconda istruzione della funzione FUN
      ...
```

Definire una funzione (2)

- Una funzione viene chiamata con opportune istruzioni

- Ad esempio in ARM con B o BL
  - B FUN → inserisce in PC l'indirizzo indicato dalla label FUN
  - BL è simile a B, ma effettua altre operazioni

Ritorno da funzione

Ritorno da una funzione

- Al termine della funzione invocata è necessario ritornare indietro
- L'indirizzo di ritorno è l'indirizzo dell'istruzione successiva a quella di chiamata
- Al termine della funzione è necessario inserire in PC l'indirizzo di ritorno.

- Dove si trova l'indirizzo di ritorno?

Indirizzo di ritorno: 1ª soluzione

- L'indirizzo di ritorno potrebbe venir salvato in una locazione della memoria adibita a tale scopo, ad esempio quella situata all'indirizzo 0.

- Limitazione: se la subroutine ne chiamasse un'altra, verrebbe perso l'indirizzo di ritorno

1ª soluzione: chiamata e ritorno

- Per chiamare una procedura PROC:

```
    MOV R0, #0      @ Indirizzo dove salvare PC
    LDR R1, =RIT    @ Indirizzo di ritorno
    STR R1, [R0]    @ Salva RIT in M[0]
    B PROC          @ Salta alla procedura (PROC→PC)
RIT: ...            @ Prima istruzione da eseguire
                    @ dopo il ritorno da funzione
```

- Per ritornare al punto di partenza alla fine di PROC:

```
PROC: ...
      MOV R0, #0
      LDR R0, [R0]
      MOV PC, R0
```

Ripristina PC all'indirizzo di ritorno

Indirizzo di ritorno: 2ª soluzione

- L'indirizzo di ritorno potrebbe venir salvato nella **prima locazione di ciascuna subroutine**, con la convenzione che le istruzioni eseguibili della subroutine siano collocate a partire dalla locazione successiva

- Limitazione: questa soluzione consente alla subroutine PROC di chiamarne un'altra, ma se la subroutine PROC chiamasse se stessa (direttamente o indirettamente), verrebbe perso l'indirizzo di ritorno.

2ª soluzione: chiamata e ritorno

- Per chiamare una procedura PROC:

```
    LDR R0, =PROC @ Indirizzo dove salvare RIT
    LDR R1, =RIT  @ Indirizzo di ritorno
    STR R1, [R0]  @ Salva il RIT in M[PROC]
    B PROC+4      @ PROC+4 → PC
RIT:
```

- Struttura della subroutine PROC:

```
PROC: .space 4  @ Word per l'indirizzo di ritorono
      ...       @ Codice della funzione
       LDR R0, =PROC
       LDR R0, [R0]
       MOV PC, R0
```

Indirizzo di ritorno: 3ª soluzione

- L'indirizzo di ritorno viene salvato in un registro di CPU anziché in una locazione di memoria:

- Limitazione: Questa soluzione consente alla subroutine PROC di chiamarne un'altra, purché venga usato un registro diverso

3ª soluzione: chiamata e ritorno

- Per chiamare una procedura PROC:

```
    LDR R14, =RIT       @ Indirizzo di ritorno
    B PROC              @ PROC → PC
RIT:
```

- Per ritornare al punto di partenza alla fine di PROC:

```
PROC: ... @ Codice della funzione
      ...
      MOV PC, R14
```

Soluzioni 1-3

- Tutte le soluzioni hanno limitazioni
  - Soluzioni 1,3 non permettono di chiamare altre funzioni
  - Soluzione 2 non permette di chiamare la stessa funzione (ricorsione)
- Le limitazioni permettono di non perdere l'indirizzo di ritorno
- Le limitazioni si possono evitare, se l'indirizzo di ritorno esistente viene temporaneamente salvato in una posizione sicura (e.g., stack) per poi essere ripristinato.

Indirizzo di ritorno: 4ª soluzione

- L'indirizzo di ritorno viene memorizzato in uno stack con un'operazione di push
  - Ogni volta è in un posto diverso!

- Questa soluzione consente alla subroutine PROC di chiamarne un'altra, o anche se stessa (direttamente o indirettamente).
- Le subroutine realizzate con questa soluzione possono essere rientranti: di esse possono essere in esecuzione contemporaneamente più istanze
  - Per poter chiamare ricorsivamente se stessa una subroutine deve essere rientrante.

4ª soluzione: chiamata e ritorno

- Per chiamare una procedura PROC:

```
LDR R0, =RIT
PUSH {R0} @ L'indirizzo RIT viene
          @ salvato nello stack
B PROC
RIT: ...
```

- Per ritornare al punto di partenza alla fine di PROC:

```
PROC: ...
      ...
      POP {PC} @ Carica in PC il valore
               @ nello stack (ovvero RIT)
```

Soluzioni a confronto

Permettono chiamate ad altre funzioni all'interno di PROC?

Sol. 1  Sol. 2  Sol. 3  Sol. 4 

Sol. 2,3 possono gestire chiamate a funzioni in certe condizioni

- E' efficiente?

Sol. 1  Sol. 2  Sol. 3  Sol. 4 

Sol. 1,2,4 richiede accesso alla memoria;
Sol. 3 accede solo ai registri

Chiamate a procedura in ARM

- Compilatore GCC per ARM usa un mix di soluzione 3 e 4
  - Per invocare una procedura PROC si salva il punto di ritorno nel registro R14 chiamato anche LR (Link Register)
  - Se è poi necessario invocare altre procedure all'interno di PROC, si salva prima il valore di LR nello stack
- Permette di sfruttare l'efficienza della soluzione 3, e permette di invocare altre procedure ad un costo leggermente superiore (e comunque inevitabile)

Chiamata e ritorno in ARM

- Una procedura PROC viene chiamata con

  BL PROC

- L'istruzione BL (Branch-and-Link)
  - salva l'indirizzo dell'istruzione successiva a BL in R14 (indirizzo di ritorno),
  - carica nel PC l'indirizzo della subroutine (effettua il salto).
- R14 si chiama **Link Register** (si può usare il simbolo **LR**)
- Non esiste un'istruzione apposita per il ritorno dalla subroutine; si ottiene con l'istruzione:

  MOV PC, LR

Passaggio dei parametri

Il passaggio dei parametri

- Si distinguono due tipi di parametri:
  - **parametri di ingresso**: dati passati alla subroutine;
  - **parametri di uscita**: risultati restituiti dalla subroutine
- Il veicolo più naturale e rapido per effettuare il passaggio dei parametri è costituito dai registri di CPU.

Esempio: passaggio per valore

- Calcolare il modulo della differenza tra due numeri interi in complemento a due.

- Supponiamo di utilizzare i registri R0, R1 e R2 come segue per passare **i valori dei parametri**:
  - parametri di ingresso: R1 e R2
  - parametro di uscita: R0

Esempio: funzione ABS

```
ABS: SUBS R0, R1, R2   @ Calcola R1-R2
     RSBMI R0, R1, R2  @ Se negativo,
                       @ calcola R2-R1
     MOV PC, LR        @ Ritorna
```

Esempio: chiamata di ABS

```
.data
OP1: .word 13       @ Primo operando
OP2: .word 9        @ Secondo operando

.bss
RIS: .skip 4        @ Risultato
```

Esempio: chiamata di ABS (2)

```
.text
LDR R3, =OP1        @ Carica indirizzo OP1
LDR R1, [R3]        @ Carica OP1
LDR R3, =OP2        @ Carica indirizzo OP2
LDR R2, [R3]        @ Carica OP2
BL ABS              @ Chiama la procedura ABS
LDR R3, =RIS        @ Carica indirizzo RIS
STR R0, [R3]        @ Salva output
```

Esempio: passaggio per indirizzo

- Anziché i valori dei parametri, è possibile passare **gli indirizzi delle locazioni** di memoria ove quei valori sono contenuti:
  - Indirizzi dei parametri di ingresso: R1 e R2
  - Indirizzo del parametro di uscita: R0

```
ABS: LDR R1, [R1]        @ Carica OP1
     LDR R2, [R2]        @ Carica OP2
     SUBS R3, R1, R2     @ Calcola R1-R2
     RSBMI R3, R1, R2    @ Se negativo,
                         @ calcola R2-R1
     STR R3, [R0]        @ Salva output
     MOV PC, LR          @ Ritorna
```

Esempio: passaggio per indirizzo (2)

```
.data
OP1: .word 13        @ Primo operando
OP2: .word 9         @ Secondo operando
.bss
RIS: .skip 4         @ Risultato

.text
LDR R1, =OP1         @ Carica indirizzo OP1
LDR R2, =OP2         @ Carica indirizzo OP2
LDR R0, =RIS         @ Carica indirizzo RIS
BL ABS               @ Chiama la procedura ABS
```

Parametri per valore o per indirizzo?

- **parametri passati per indirizzo**:
  - avendo a disposizione l'indirizzo, la subroutine può accedere al valore originario del parametro e modificarlo.

- **parametri passati per valore**:
  - se alla subroutine viene passato il valore (cioè copia del valore) di un parametro, eventuali modifiche apportate a questo valore dalla subroutine non coinvolgono il valore originario.

Disciplina di programmazione

- Conviene che, dopo l'esecuzione di una subroutine, risulti modificato solo ciò è esplicitamente previsto che la subroutine modifichi, cioè i parametri di uscita (poiché questi restituiscono i risultati);
- se la subroutine modificasse i valori originari dei parametri di ingresso, questo sarebbe un effetto collaterale indesiderato della sua esecuzione.

Esempio

Programma chiamante:

```
. . .
RIS = abs(OP1, OP2);
. . .
```

parametri di ingresso

parametro di uscita

Procedura chiamata:

```
int  abs(int x, int y);
     x = x - y;
     if(x < 0){x := -x;}
     return x;
end
```

La procedura abs(…) modifica x :
se OP1 fosse passato per indirizzo, risulterebbe modificato anche OP1 (effetto collaterale indesiderato);
se OP1 è passato per valore, le modifiche di x non hanno effetto su OP1.

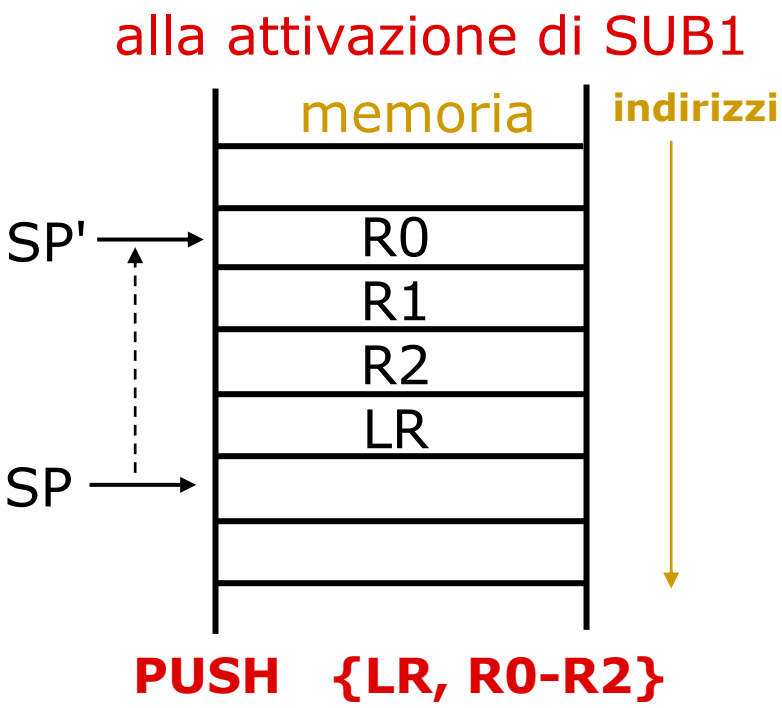
Utilizzo dello stack

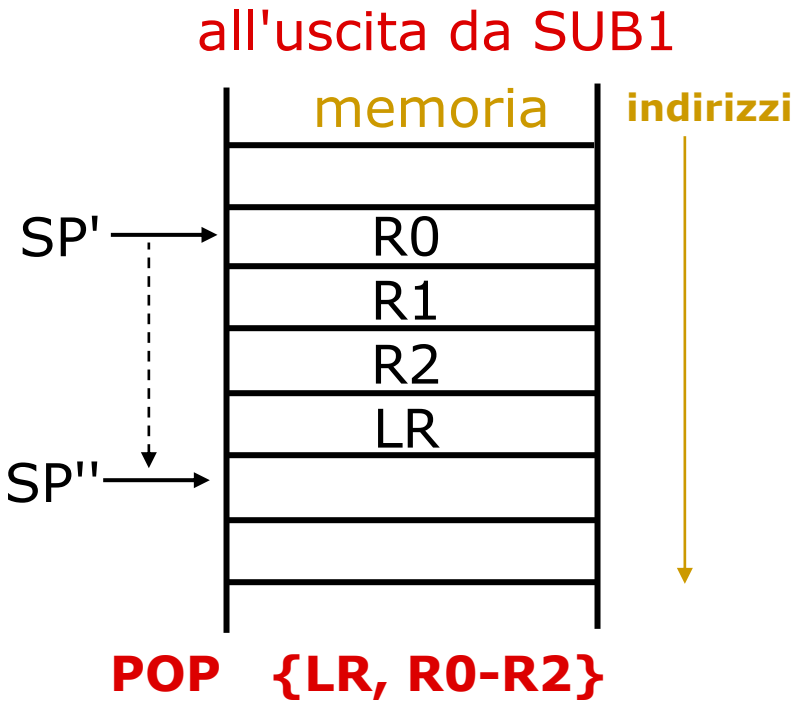
- Una subroutine può avere la necessità di utilizzare dei registri per collocarvi i suoi risultati intermedi
  - Questo utilizzo può violare la disciplina di programmazione (non modificare ciò che non è esplicitamente previsto).
- In questo caso, conviene usare lo stack per:
  - **salvare il contenuto** dei registri usati dalla subroutine (che non siano usati per i parametri di uscita), prima di modificarli;
  - **ripristinare il contenuto** di questi registri, prima di ritornare al programma chiamante.
  - Nel caso di ARM, vale in particolare per il registro LR/R14

Salvataggi nello stack

```
        BL SUB1
        . . .
SUB1: PUSH {LR, R0-R2}  @ push LR e i registri usati
        . . .
        LDR R2, [R1, #4]! @ modifica dei registri
        ADD R0, R0, R2    @ modifica altri registri
        BL SUB2           @ chiamata a una subroutine
                          @ annidata
        . . .
        POP {LR, R0-R2}   @ pop LR e i registri usati
        MOV PC, LR        @ ritorno
```

Salvataggi nello stack





Con le istruzioni PUSH e POP, i registri sono collocati in memoria, a partire dall'indirizzo puntato da SP', nell'ordine (ad indirizzi crescenti con l'indice dei registri), indipendentemente dall'ordine scritto nell'istruzione.

Annidamento delle subroutine

SUB1 chiama un'altra subroutine SUB2.

Se anche SUB2 salva nello stack LR e i registri usati, la situazione dello stack è: