Esempi di subroutine in ARM

Argomenti:

- Esempi di chiamate a subroutine in ARM con o senza stack frame

Tre esempi

- Es 1 (lunghezza stringa): creazione dello stack, funzioni senza stack frame
- Es 2 (fattoriale): utilizzo stack di sistema, funzioni senza stack frame; due versioni (iterativa e ricorsiva)
- Es 3 ("codice fiscale"): utilizzo stack di sistema, funzioni con stack frame

Es 1: lunghezza di una stringa

- Si vuole scrivere una subroutine che calcoli la **lunghezza di una stringa.**
- La stringa in memoria sia terminata dal codice di controllo **EOS** (End Of String): **byte nullo** che indica la line della stringa.

- Interfaccia della subroutine **strlen:**
  - **Parametro di ingresso**: **R0** puntatore alla stringa;
  - **Parametro di uscita**: **R1** lunghezza della stringa.

Es 1: dati non inizializzati (.bss)

- **La subroutine fa uso di uno stack** (per salvare i registri usati):
  - Quando si utilizza un compilatore, lo stack è già disponibile.
  - Assumiamo però di doverlo crearlo noi!
  - Dimensione stack: 2048 byte

Es 1: dati non inizializzati (.bss)

- E' prevista un'area di memoria destinata allo stack, nella sezione contenente i dati non inizializzati (.bss):
- lo stack è di tipo full descending: lo stack cresce verso gli indirizzi più bassi e lo stack pointer viene fatto avanzare (decrementato) prima dei push;
- Il valore iniziale dello stack pointer **SP** (a stack vuoto) è l'indirizzo `miostack`.

```
.bss

        .skip 2048  @ 2048 byte di stack "privato"
miostack:
```

Es 1: dati inizializzati (.data)

La stringa (di cui si vuol calcolare la lunghezza) può essere collocata nella sezione dei dati inizializzati (.data), tramite l'apposita direttiva (.ascii):

- in questo modo la stringa viene caricata in memoria insieme con il programma.

```
.data
str1:  .ascii "Questa stringa e' lunga 36 caratteri"
       .byte 0         @ EOS (terminatore di stringa)
       .align   4      @ allineamento a indirizzo di word
```

Es 1: inizializzazione

- È necessario impostare inizialmente:
  - il valore dello stack pointer (l'indirizzo **miostack**),
  - il valore da inserire in **R0** (param. di ingresso);
  - per calcolare la lunghezza della stringa, se ne collocherà in **R0** l'indirizzo (**str1**) e si chiamerà la subroutine **strlen**: la lunghezza verrà restituita in **R1**.
- l'ambiente di programmazione usato prevede che la prima istruzione da eseguire sia all'indirizzo **main**
- le istruzioni sono collocate nella sezione **.text**:

Es 1: inizializzazione

```
.text

main:
        LDR SP, =miostack    @ inizializza SP
        PUSH {R0, LR}
        LDR R0, =str1        @ puntatore alla stringa str1
        BL strlen            @ salta alla subroutine
        POP {R0, LR}
        MOV PC, LR
```

Es 1: subroutine `strlen`

La subroutine conteggia in R1 i caratteri fino a EOS escluso:

- usa il registro R2 (ci mette il carattere da esaminare);
- il contenuto di R2 viene prima salvato e, alla fine, ripristinato

Es 1: subroutine `strlen`

```
strlen:  PUSH {R2}          @ salva R2 sullo stack
                            @ R0: ind. inizio della stringa
         MOV R1, #0 @ R1: lunghezza stringa
loop:    LDRB R2, [R0, R1]@ carica in R2 il byte
                            @ di indirizzo R0+R1
         CMP R2, #0 @ è fine stringa (EOS)?
         ADDNE R1, R1, #1  @ no: conteggia il carattere
         BNE loop           @ esamina quello successivo
fine:    POP {R2}           @ ripristina i registri usati
         MOV PC, LR         @ fine subroutine
```

Es 2: fattoriale

- Si vuole scrivere una subroutine che calcoli il fattoriale di un numero

$$n! = \begin{cases} n \cdot (n-1)! & \text{per } n > 0 \\ 1 & \text{per } n = 0 \end{cases}$$

Interfaccia:

- **Parametro di ingresso**: `R0` contiene n
- **Parametro di uscita**: `R0` contiene $n!$ oppure 0 in caso di overflow

Es 2: dati e inizializzazione

- Per lo stack assumiamo di avere lo stack già a disposizione creato prima dell'invocazione del main.

Es 2: dati e inizializzazione

E’ necessario impostare inizialmente:

- il valore di n da inserire in **R0** (param. di ingresso):

```
.text
main:
        PUSH {R0,LR} @ Salviamo indirizzo di ritorno
        MOV R0, #6   @ calcoleremo 6!=0x02D0
        BL fact      @ salta alla subroutine
        POP {R0,LR}
        MOV PC, LR   @ termina il programma ritornano
                     @   il controllo al sistema
```

Es 2: subroutine `fact`

```
fact:     CMP R0, #0       @ R0=0?
          MOVEQ R0, #1     @ sì: 0!=1
          MOVEQ PC, LR     @ fine subroutine
          CMP R0, #12      @ 13! non ci sta in 32 bit
          BLS do_it        @ non c'e' overflow: procedi
          MOV R0, #0       @ c'è overflow: poni R0=0
          MOV PC, LR       @ fine subroutine
do_it:    PUSH {R1}        @ salva R1 sullo stack
          MOV R1, R0       @ N -> R1
ciclo:    SUBS R1, R1, #1  @ R1=N-1
          BEQ fine         @ se R1=0 il ciclo e' finito
          MUL R0, R1, R0   @ R0=R1*R0 = N*(N-1)
          B ciclo          @ vai all'iterazione successiva
fine: POP {R1}             @ ripristina R1 dallo stack
      MOV PC, LR           @ fine subroutine
```

Es 2: versione alternativa: ricorsiva

rfact: versione ricorsiva della subroutine

- prevede un solo parametro (di ingresso e di uscita) in un registro (R0);
- utilizza un solo dato locale collocato in un registro (R1), che viene salvato nello stack prima di ciascuna chiamata ricorsiva;
- la sua semplicità consente di garantire la rientranza senza l'uso di stack frame.

Es 2: versione ricorsiva

```
rfact:      CMP R0, #12      @ 13! provoca overflow
            BLS do_it_r      @ no overflow: procedi
            MOV R0, #0       @ c'e' overflow: R0=0
            MOV PC, LR       @ fine subroutine
do_it_r:    CMP R0, #2       @ R0 e' = a 0, 1 o 2?
            BHI fa           @ no: procede
            CMP R0, #0       @ R0=0?
            MOVEQ R0, #1     @ sì: 0!=1
            MOV PC, LR       @ altrimenti 1!=1 e 2!=2
fa:         PUSH {R1, LR}    @ salva R1 e LR
            MOV R1, R0       @ N in R1
            SUB R0, R0, #1   @ N-1
            BL do_it_r       @ calcola (N-1)! in R0
            MUL R0, R1, R0   @ N!=N*(N-1)!
            POP {R1, LR}     @ fine subroutine
            MOV PC, LR       @ fine subroutine
```

Es 3: Codice Fiscale

Si vuole scrivere un programma che elimini le vocali da una stringa, creando una sorta di "**codice fiscale**" (le vocali eliminate vengano messe alla fine, come indicato nell'esempio)

Prima: **Quarantaquattro**

Dopo: **Qrntqttr**uaaaauao

Es 3: convenzioni

- **Parametri di ingresso**
  - **R0**: indirizzo primo carattere della stringa
  - **R1**: lunghezza della stringa in byte

- **Parametri di uscita**
  - **R0**: puntatore alla stringa del codice fiscale
  - **R1**: puntatore alla stringa delle vocali eliminate

- **Stack frame:** secondo le convenzioni di **gcc**
- Assumiamo di poter utilizzare lo stack di sistema

Es 3: sezioni .data e .bss

Nella sezione dei dati inizializzati (.data) viene posta la stringa da elaborare (str1) e la tabella (codvoc) contenente i codici ASCII delle vocali:

```
.data
codvoc:   @ Codici ASCII vocali:
.byte 0x41, 0x45, 0x49, 0x4F, 0x55 @ maiuscole
.byte 0x61, 0x65, 0x69, 0x6F, 0x75 @ minuscole
.align 4
str1: .ascii "Quarantaquattro gatti in fila per tre con il resto di 2“
.align 4
```

Es 3: inizializzazione

- Prima di chiamare la subroutine codfisc, viene chiamata la subroutine **strlen** (vedi esempio 1), che restituisce in **R1** la lunghezza della stringa;
- Il puntatore a **str1** viene passato, in **R0**, sia alla **strlen**, sia alla **codfisc**.

```
.text
.global main
main:
     PUSH {R0,LR}        @ Salva registri
     LDR R0, =str1       @ lavoreremo su str1
     BL strlen           @ in R1 la lunghezza
     BL codfisc
     POP {R0,LR}         @ Ripristina registri
     MOV PC, LR          @ Termina il main
```

Es 3: subroutine `vocale`

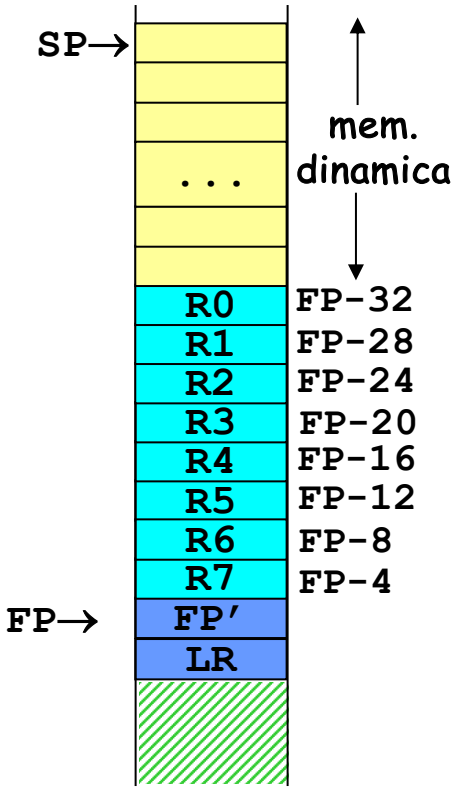
- È utile una subroutine **(vocale)** che stabilisce se un carattere è una vocale confrontandolo con la tabella dei codici ASCII delle vocali, :
  - **input**: **R0** contiene il carattere da esaminare
  - **output**: **R0**=0 se input è una vocale, ≠0 altrimenti

Es 3: subroutine `vocale`

```
vocale:   PUSH {R1-R3}   @ salva R1-R3 sullo stack
          LDR R1, =codvoc     @ R1 puntatore tab. delle vocali
          MOV R2, #0          @ R2 offset nella tabella
          MOV R3, #0          @ R3: registro di lavoro
ciclovoc: LDRB R3, [R1, R2]   @ carica una vocale
                              @ della tabella
          CMP R0, R3          @ R0 contiene una vocale?
          BEQ voc             @ si: salta a voc
          ADD R2, R2, #1      @ no: esamina elem. succ. in
tab.
          CMP R2, #10         @ la tabella e' finita?
          BLO ciclovoc        @ no: iterazione successiva
          B finevoc           @ si: fine subroutine (R0≠0)
voc:      MOV R0, #0          @ è una vocale: restituisce R0=0
finevoc:  POP  {R1-R3}        @ ripristina i registri usati
          MOV PC, LR          @ fine subroutine
```

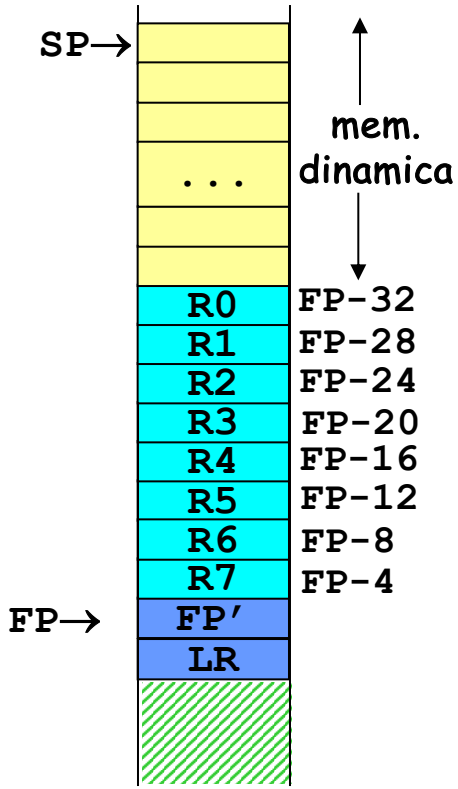
Es 3: subroutine `codfisc` - 1

Fase iniziale: la subroutine ricopia la stringa nella memoria dinamica in cima allo stack (area gialla in figura): il numero di word di questa mem. dinamica deve quindi essere sufficiente a contenere la stringa.



Es 3: subroutine `codfisc` - 1

```
codfisc:
PUSH {FP, LR}
MOV FP, SP      @ nuovo FP
PUSH {R0-R7}    @ salvataggio reg. usati
                @ R1=num. caratteri della str.
BIC R1, R1, #3 @ azzera i bit 0 e 1
ADD R1, R1, #4 @ aggiunge 4 (così in R1 c'è
                @ il multiplo di 4 superiore)
SUB SP, SP, R1 @ spazio mem.dinamica
                @ sufficiente per str.
```



Es 3: subroutine `codfisc` - 2

Scansione 1: copia la stringa nello spazio di memoria dinamica del frame e ne conta il numero di non-vocali (lunghezza del cod. fiscale).

Es 3: subroutine `codfisc` - 2

```
LDR R1, [FP, #-28] @ ripristina R1=lunghezza stringa
MOV R2, #0         @ R2=lunghezza codice fisc. (non-vocali)
MOV R3, #0         @ R3=offset del car.nella str.
LDR R4, [FP, #-32] @ R4=puntatore alla str.(R0 nel frame)
MOV R5, SP         @ R5=puntatore alla str.in mem.dinamica

scan:
     LDRB R0, [R4, R3]  @ carica un carattere della str.
     STRB R0, [R5, R3]  @ copia il car. nella mem. dinamica
     BL vocale          @ controlla se vocale
     CMP R0, #0         @ il carattere è una vocale?
     ADDNE R2, R2, #1   @ no: allora appartiene
                        @     al cod.fiscale
     ADD R3, R3, #1     @ incrementa l'offset prossimo car.
     CMP R1, R3         @ la str. è finita?
     BHI scan           @ no: esamina il carattere
                        @     successivo
```

Es 3: subroutine `codfisc` - 3

Scansione 2: costruisce le stringhe codice fiscale e vocali.

Es 3: subroutine `codfisc` - 3

```
LDR R1, [FP, #-32]     @ R1=puntatore alla stringa codice fiscale
ADD R2, R1, R2         @ R2=puntatore alla stringa delle vocali
MOV R3, SP             @ R3=puntatore alla stringa in mem. dinam.
MOV R4, #0             @ R4=offset nella stringa codice fiscale
MOV R5, #0             @ R5=offset nella stringa delle vocali
MOV R6, #0             @ R6=offset nella stringa in mem. dinam.
LDR R7, [FP, #-28]     @ R7=lungh. stringa (R1 salvato nel frame)
build: LDRB R0, [R3, R6]    @ car. dalla stringa nella
                            @ mem.dinamica
      BL vocale             @ controlla se vocale
      CMP R0, #0            @ il carattere è una vocale?
      LDRB R0, [R3, R6]     @ (R0 modificato da vocale: va ricaricato)
      STRNEB R0, [R1, R4]   @ no: ricopialo nel codice fiscale
      ADDNE R4, R4, #1      @ e incrementa il relativo offset
      STREQB R0, [R2, R5]   @ si: ricopialo nella stringa delle vocali
      ADDEQ R5, R5, #1      @ e incrementa il relativo offset
      ADD R6, R6, #1        @ carattere successivo
      CMP R7, R6            @ la stringa è finita?
      BHI build             @ no: esamina il carattere successivo
```

Es 3: subroutine `codfisc` - 4

Fase finale: sistemazione dei parametri d'uscita e rimozione dello stack frame

Es 3: subroutine `codfisc` - 4

```
fine:
MOV R0, R1    @ R0=puntatore alla stringa cod. fiscale
MOV R1, R2    @ R1=puntatore alla stringa delle vocali
              @ R7=numero di caratteri della stringa
BIC R7, R7, #3    @ azzera i bit 0 e 1
ADD R7, R7, #4    @ aggiunge 4 (multiplo di 4
superiore)
ADD SP, SP, R7    @ disalloca memoria dinamica
ADD SP, SP, #8    @ disalloca spazio R0-R1 (non vanno
                  @ ripristinati: parametri di uscita!
POP  {R2-R7}      @ ripristina gli altri registri
                  @  usati
POP  {FP, LR}     @ riattiva il frame precedente e
MOV PC, LR        @ ritorna al programma chiamante
```