Esempi di programmi assembly

Copia di un vettore

Lunghezza di un vettore

Fattoriale

Copia di un vettore (avanzato)

EsA1: copia di un vettore

Nel progettare un programma che esegua la **copia di un vettore** vanno definiti:

- **i dati** su cui il programma opera:
  - sia quelli "inizializzati", che hanno già un valore iniziale prima che inizi l'esecuzione del programma (nell'esempio il vettore da copiare può essere di questo tipo),
  - sia quelli "non inizializzati", nei quali verranno scritti dei valori durante l'esecuzione del programma (nell'esempio il vettore che riceve la copia può essere di questo tipo);
- **le istruzioni** che, operando su quei dati, realizzano quanto richiesto.

EsA1: dati inizializzati (.data)

- Il vettore da copiare, contenente ad es. una stringa di 480 caratteri (480 byte), può esser collocato nella sezione **.data**, tramite la direttiva **.ascii**
- In questo modo, il vettore viene caricato in memoria (all'indirizzo vett1), prima dell'inizio del programma:

```
        .data
vett1:
  .ascii "Registers R0 to R7 are unbanked registers. This me"
  .ascii "ans that each of them refers to the same 32-bit ph"
  .ascii "ysical register in all processor modes. They are c"
  .ascii "ompletely general-purpose registers, with no speci"
  .ascii "al uses implied by the architecture, and can be us"
  .ascii "ed wherever an instruction allows a general-purpos"
  .ascii "e register to be specified. Registers R8 to R14 ar"
  .ascii "e banked registers. The physical register referred"
  .ascii "to by each of them depends on the current processo"
  .ascii "r mode. Where a particular phy"
```

EsA1: dati non inizializzati(.bss)

- Nella sezione contenente i dati non inizializzati (**.bss**), si può collocare il vettore destinato a ricevere la copia (vett2):

```
        .bss
vett2:  .skip 480               @ spazio di 480 byte
```

EsA1: uso dei registri

- Conviene mantenere nei registri le informazioni che servono per realizzare l'algoritmo:
  - **R0**: lunghezza del vettore in byte,
  - **R1**: indirizzo primo elemento di vett1 (vettore sorgente),
  - **R2**: indirizzo primo elemento di vett2 (vett. destinazione).

```
          .global main
          .text

main:     LDR R1, =vett1      @ indirizzo del v.re sorgente
          LDR R2, =vett2      @ indirizzo del v.re dest.
          MOV R0, #480        @ lunghezza vettore in BYTE
```

EsA1: cuore del programma

```
      ADD R0, R1, R0  @ R0: indirizzo di stop (indirizzo
                      @ successivo all'ultimo elemento)
loop: LDR R3, [R1], #4 @ carica un word dal 1.mo vettore
      STR R3, [R2], #4 @ salva il word nel 2.do vettore
      CMP R1, R0       @ raggiunta la fine?
      BLO loop         @ no: riesegue il loop
```

- Modi di indirizzamento usati:
  - di registro,
  - di registro indiretto,
  - post-incremento.
- Viene usato **R3** (come registro di transito dei dati).

EsA1: il programma completo

```
        .global main
        .text

main:   LDR R1, =vett1      @ indirizzo del v.re sorgente
        LDR R2, =vett2      @ indirizzo del v.re dest.
        MOV R0, #480        @ lunghezza vettore in BYTE
        ADD R0, R1, R0      @ indirizzo di stop in R0
loop:   LDR R3, [R1], #4    @ carica un word dal 1.mo v.re
        STR R3, [R2], #4    @ salva il word nel 2.do v.re
        CMP R1, R0          @ raggiunta la fine?
        BLO loop            @ no: riesegue il loop

trap:   B trap              @ fine programma
```

Prima delle istruzioni viste, vanno inserite le istruzioni che inseriscono nei registri i valori di inizializzazione (sezioni .data, .bss)

EsA2: lunghezza di una stringa

- Si vuole scrivere un programma che calcoli la **lunghezza di una stringa**.
- La stringa in memoria sia terminata dal codice di controllo **EOS** (End Of String): byte con tutti i bit uguali a zero, che indica la fine della stringa.

EsA2: sezioni .data e .bss

Il programma non fa uso di dati non inizializzati:

- la sezione .bss non c'è.

```
/* Dati inizializzati (stringa) */

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

EsA2: uso dei registri

- **R0**: contenga, all'inizio, l'indirizzo del primo carattere della stringa;
- **R1**: contenga, alla fine, la lunghezza della stringa, cioè il numero di caratteri di cui è costituita (EOS escluso).

EsA2: le istruzioni (.text)

- esempio di istruzioni eseguite sotto condizione;
- esempio di indirizzamento con offset da registro.

```
          .global main
          .text

main:     LDR R0, =str1        @ puntatore alla stringa str1
strlen:   MOV R1, #0           @ R1: lunghezza stringa
loop:     LDRB R2, [R0, R1]    @ carica byte all'ind. R0+R1
          CMP R2, #0           @ e' fine stringa (EOS)?
          ADDNE R1, R1, #1     @ no: conteggia il carattere
          BNE loop             @ esamina quello successivo
trap:     B trap               @ fine programma
```

EsA3: fattoriale

Si vuole scrivere un programma che calcoli il **fattoriale di un numero**

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

Il valore n è contenuto in R0

EsA3 : sezioni .data e .bss

- Il programma non fa uso di dati inizializzati:
  - la sezione .data non c'è;
- Il programma non fa uso di dati non inizializzati:
  - la sezione .bss non c'è.

EsA3 : uso dei registri

- **R0** contenga:
  - all'inizio, il numero n di cui calcolare il fattoriale;
  - alla fine:
    - n! (il fattoriale di n),
    - 0 in caso di overflow.

EsA3: parte principale

```
fact:   CMP R0, #12       @ 13! non ci sta in 32 bit
        BLS do_it         @ non c'e' overflow: procedi
        MOV R0, #0        @ c'e' overflow: poni R0=0
        B fine            @ fine programma
do_it:  MOV R1, R0        @ R1 iteratore
        MOV R0, #1        @ R0 prodotto parziale
ciclo:  CMP R1, #0        @ Controlla se R1=0
        BEQ fine          @ Se si, fine
        MUL R0, R1, R0   @ Moltiplica
        SUB R1, R1, #1    @ Decrementa R1
        B ciclo           @ Ripeti
fine:
```

EsA3: main

```
        .global main
        .text
main:   MOV R0, #6          @ da calcolare 6!=720=0x02D0

fact:   CMP R0, #12         @ 13! non ci sta in 32 bit
        BLS do_it           @ non c'e' overflow: procedi
        MOV R0, #0          @ c'e' overflow: poni R0=0
        B fine              @ fine programma
do_it:  MOV R1, R0          @ R1 iteratore
        MOV R0, #1          @ R0 prodotto parziale
ciclo:  CMP R1, #0          @ Controlla se R1=0
        BEQ fine            @ Se si, fine
        MUL R0, R1, R0    @ Moltiplica
        SUB R1, R1, #1    @ Decrementa R1
        B ciclo             @ Ripeti
fine:
trap:   B trap              @ fine programma
```

EsA4: copia di vettori (avanzato)

Spostare il contenuto di una serie di locazioni contigue di memoria da una posizione ad un'altra (programma che copia il contenuto di un array)

EsA4: inizio

```
.text
copy:   LDR R1, =TABLE1 @ R1 punta a TABLE1
        LDR R2, =TABLE2 @ R2 punta a TABLE2
        LDR R0, [R1] @ carica il primo valore
        STR R0, [R2] @ copia il primo valore
             ...

.data
TABLE1: ...             @ array sorgente

.bss
TABLE2: ...             @ array destinazione
```

EsA4: somma

- Bisogna ora continuare a copiare il secondo valore (… poi dovremo preoccuparci anche degli elementi successivi):

```
copy:   LDR R1, =TABLE1 @ R1 punta a TABLE1
        LDR R2, =TABLE2 @ R2 punta a TABLE2
        LDT R0, [R1] @ carica il primo valore
        STR R0, [R2] @ copia il primo valore

        ADD R1, R1, #4  @ R1 punta al successivo
        ADD R2, R2, #4  @ R2 punta al successivo
        LDR R0, [R1]    @ carica il secondo valore
        STR R0, [R2]    @ … e copialo
```

- Si sono incrementati i due puntatori alle tabelle (i registri R1 e R2) della dimensione degli elementi (word da 4 byte)

EsA4: offset

- Si può inserire l'operazione di incremento dei puntatori nelle istruzioni di LDR e STR, con l'indirizzamento a due componenti (registro+offset immediato):

```
copy:   LDR R1, =TABLE1       @ R1 punta a TABLE1
        LDR R2, =TABLE2       @ R2 punta a TABLE2
        LDR R0, [R1]      @ carica il primo val.
        STR R0, [R2]      @ salva il primo valore
        LDR R0, [R1, #4]      @ carica il secondo val.
        STR R0, [R2, #4]      @ ... e copialo
```

- Usati i puntatori contenuti in R1 e R2, si è aggiunta ad essi la dimensione dell'elemento (4) in modo che puntino al successivo (con un'unica istruzione):

LDR R0, [R1, #4] @ R0 ← M[R1+4]

EsA4: pre-incremento

- I registri R1 e R2 puntano all'inizio delle rispettive tabelle; per scorrerle tutte bisogna aumentare, ogni volta, gli offset (8, 12, 16, ...), oppure (meglio) si incrementano i puntatori R1 e R2, usando l'indirizzamento con pre-incremento:

```
copy:   LDR R1, =TABLE1       @ R1 punta a TABLE1
        LDR R2, =TABLE2       @ R2 punta a TABLE2
        LDR R0, [R1]      @ carica il primo val.
        STR R0, [R2]      @ salva il primo valore
        LDR R0, [R1, #4]!     @ carica il secondo val.
        STR R0, [R2, #4]!     @ ... e copialo
```

- Il punto esclamativo comporta la pre-modifica del registro:

LDR R0, [R1, #4]! @ R1 ← R1+4, R0 ← M32[R1]

EsA4: post-incremento

- È possibile uniformare anche la prima operazione, usando l'indirizzamento con post-incremento:

```
copy:   ldr R1, =TABLE1          @ R1 punta a TABLE1
        ldr R2, =TABLE2          @ R2 punta a TABLE2
loop:   ldr R0, [R1], #4 @ carica dal 1° vettore
        str R0, [R2], #4 @ salva nel 2° vettore
```

- Rispetto al caso precedente, R1 e R2 sono prima usati per accedere agli operandi, e poi vengono incrementati di 4:

```
LDR R0, [R1], #4    @  R0←M32[R1], R1←R1+4
```

- Servono altre istruzioni per eseguire il loop il numero di volte desiderato (istruzioni di confronto e di salto)

EsA4: Completamento dell'esempio

- Supponiamo che l'array da copiare contenga 16 elementi: le istruzioni scritte prima vanno eseguite ripetutamente (loop) 16 volte; allo scopo si può utilizzare un registro (ad es. R3) come contatore:
  - vi si inserisce il valore iniziale 16,
  - lo si decrementa di una unità ad ogni iterazione,
  - con una instruzione di salto condizionato si riesegue il loop se il contatore è diverso zero, altrimenti no:

EsA4: Completamento dell'esempio (2)

```
        MOV R3, #0x10      @ valore iniziale
                           @ contatore
copy:   LDR R1, =TABLE1    @ R1 punta a TABLE1
        LDR R2, =TABLE2    @ R2 punta a TABLE2
loop:   LDR R0, [R1], #4   @ carica dal primo
                           @vetore
        STR R0, [R2], #4   @ salva nel secondo
                           @ vettore
        SUBS R3, R3, #1    @ decrementa il
                           @ contatore
        BNE loop           @ salta se non è zero
```