Architettura degli elaboratori – 13 gennaio 2021

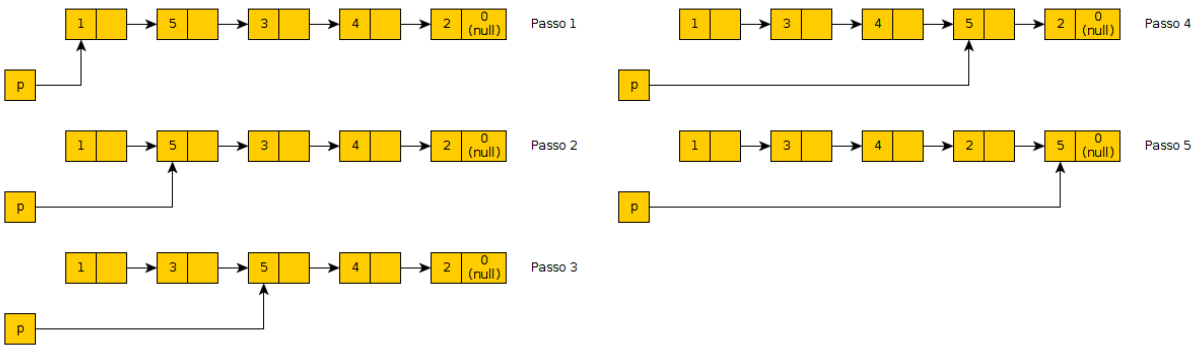
pre-prova per l'orale senza prove di verifica intermedia valide

Consideriamo una lista formata da elementi che contengono un'informazione di tipo intero. In C il tipo dell'elemento della lista potrebbe essere definito come segue:

```
typedef struct __node { int info; struct __node * next; } node_t;
```

Si richiede di scrivere due funzioni in codice assembler ARMv7:

- una funzione con signature **`void subst(node_t * p)`** che scorre la lista e per ognuno degli elementi trovati, qualora ci sia un elemento successivo e le informazioni dei due elementi della lista risultino disordinati (informazione del primo elemento maggiore di quella dell'elemento successivo) scambiano i valori dei due campi informazione. La Figura seguente illustra il funzionamento della **`subst`**.
- Una funzione di signature **`void l_stampa_a(node_t * p)`** che stampi uno dopo l'altro, nell'ordine, tutti i campi informazione che compaiono nella lista puntata da **`p`**. Ogni elemento dovrà essere stampato con una chiamata alla funzione di libreria tipo **`printf("%d -> ",info)`**.



La lista su cui operare è generata da un pezzetto di programma C ed il **`main`** è fornito (file **`main.s`**) insieme alla funzione che alloca una lista da utilizzare come esempio (file **`alloca.c`**) in modo che si possano testare i risultati delle proprie funzioni. Se nel codice del **`main`** si sostituiscono le **`bl l_stampa_a`** (chiamate alla funzione stampa definita in assembler) con **`bl l_stampa`** (chiamate alla funzione stampa definita in C nel file **`alloca.c`**) si può utilizzare per la stampa delle liste la funzione già scritta in C, così che si possa controllare incrementalmente che tutto quanto funzioni. Per compilare il tutto, immaginando che abbiate un file **`subst.s`** e uno **`stampa.s`** utilizzate un comando **`arm-linux-gnueabihf-gcc main.s alloca.c subst.s stampa.s`** e poi un comando **`qemu-arm a.out`** per eseguire il programma (se utilizzate una macchina con un processore arm, **`gcc main.s alloca.c subst.s stampa.s`** per compilare e **`./a.out`** per eseguire).

ATTENZIONE: *devono essere rispettate tutte le convenzioni che abbiamo visto nelle lezioni e si valuteranno positivamente la minimizzazione del numero di istruzioni e delle operazioni che fanno accesso alla memoria.*

File main.s

```
.text
            .global main

main:  push {r4,lr}         @ salviamo r4, che si utilizza per il puntatore alla lista
                            @ e salviamo anche l'indirizzo di ritorno
@ preparazione lista in memoria, chiamata a funzione C
mov r0, #16           @ 16 valori nella lista
mov r1, #8            @ fra 0 e 8
bl l_alloca           @ restituisce in r0 la lista

mov r4, r0            @ ci salviamo l'indirizzo della lista
bl l_stampa_a         @ stampa la lista iniziale
mov r0, r4            @ puntatore alla lista
bl subst              @ r0 vero/falso
mov r0,r4             @ puntatore alla lista
bl l_stampa_a         @ stampa la lista finale
pop {r4,pc}           @ e restituisce il controllo al chiamante
```

File alloca.c

```
#include <stdio.h>
#include <stdlib.h>

/* tipo per l'elemento della lista: due parole contigue con informazione (intera)
   e      puntatore      al      prossimo      elemento      della      lista      */

typedef struct __node { int info; struct __node * next; } node_t;

/* alloca una lista con un numero di elementi dato (informazione: numero pseudo
   casuale in [0,max) e restituisce il puntatore al primo elemento */

void * l_alloca(int n, int max) {
  node_t * start = NULL;
  srand(123);
  for(int i=0; i<n; i++) {
    node_t * ptr = (node_t *) malloc(sizeof(node_t));
    ptr->info = rand() % max;
    ptr->next = start;
    start = ptr;
  }

  return ((void *) start);
}

/*    stampa    una    lista    di    elementi    node_t    (solo    campo    info)    */

void l_stampa(node_t * p) {
  printf("Lista generata:\n");
  while(p != NULL) {
    printf("%d -> ",p->info);
    p = p->next;
  }
  printf("Fine lista\n");
  return;
}
```

Bozza di soluzione

File subst.s

```
.text
.global subst

@ r0 indirizzo di partenza della lista
subst:
cmp r0, #0
moveq pc, lr @ ritorna se non c'e' piu' nulla

ldr r1, [r0,#4]
cmp r1, #0
moveq pc, lr @ ritorna anche se non c'è un next

ldr r2, [r1] @ curr.next.info
ldr r3, [r0] @ curr.info
cmp r3, r2   @ se curr.info e' piu' grande di next.info
strgt r2, [r0]      @ scambiali
strgt r3, [r1]      @ sono gia' nei registri quindi basta questo

mov r0, r1   @ next
b subst
```

File stampa.s

```
.data
fmt:   .string "%d -> "
nl:    .string "\n"

.text
.global l_stampa_a

l_stampa_a:
cmp r0, #0          @ controlliamo che il puntatore non sia NULL
moveq pc, lr        @ in questo caso si ritorna direttamente, la lista e' vuota

push {r4, lr}       @ salvo lr perche' si fanno bl e r4 per mantenerci il puntatore
corrente

loop:  cmp r0, #0          @ controllo se fine lista
beq fine            @ fine lavori
mov r4, r0          @ salva indirizzo elemento corrente
ldr r0, =fmt        @ prepara argomenti della printf: stringa formato
ldr r1, [r4]        @ info del nodo della lista
bl printf           @ stampiamo il contenuto dell'elemento corrente
ldr r0, [r4, #4]    @ recupera il next dell'elemento corrente
b loop              @ e processa il prossimo elemento della lista

fine:  ldr r0, =nl         @ aggiungi un newline in fondo alle stampe
bl printf

pop {r4, pc}        @ e ritorna
```