Architettura degli elaboratori – Appello 7 settembre 2022 – Corsi A&B

Scrivere Nome, Cognome, Matricola e Corso su ognuno dei fogli consegnati. I risultati e il calendario degli orali saranno pubblicati sul Google classroom del corso AE 20-21 (ultima edizione prima del nuovo ordinamento).

Si richiede di scrivere in assembler ARMv7 due funzioni:

- la prima ha signature **int somma(int x, int * a, int n)** e calcola il valore del polinomio con coefficienti nel vettore a, di **n** posizioni, sulla variabile **x**, ovvero il valore:
  sum = a[0] * x^0 + a[1] * x^1 + … + a[n-1] * x^{n-1}
- la seconda ha signature **int * sommaT(int x, int * mat, int n, int m)** e calcola, per ogni riga della matrice **mat** di **n** righe e **m** colonne il valore del polinomio i cui coefficienti sono rappresentati nella riga utilizzando chiamate alla funzione somma (tutti gli interi sono interi a 32 bit).

Ad esempio, assumendo che i parametri valgano **x**=2, **n**=4, **m**=3, e che la matrice **mat** sia quella rappresentata a sinistra nella figura seguente e considerando che la colonna più a sinistra rappresenta i coefficienti di x^0 e che la colonna più a destra rappresenta i coefficienti da moltiplicare per x^2, il risultato della **sommaT** dovrebbe essere il vettore a destra nella stessa figura.

1	2	3	17
3	2	1	11
1	2	1	9
2	1	1	8

Per controllare che il codice calcoli quanto richiesto, si può utilizzare il codice C seguente:

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

#define N 4
#define M 3

extern int somma(int, int*, int);
extern int* sommat(int, int*, int, int);

int main(int argc, char **argv) {

  int m[N][M];
  int * rv;
  int i,j;
```

```
int x = atoi(argv[1]);

for(i=0; i<N*M; i++)
     m[i/M][i%M] = atoi(argv[2+i]);

for(i=0;i<N;i++) {
    for(j=0; j<M; j++)
      printf("%d ", m[i][j]);
    printf("\n");
  }

  rv = sommat(x,&m[0][0],N,M);
  for(i=0;i<N; i++)
    printf("%d\n",rv[i]);

  return(0);
}
```

Bozza di soluzione

Prima funzione: somma

```
.global somma
.type somma, %function        @ param: x, v, n (r0, r1, r2)

somma:  push {r4-r7}          @ salvataggio registri utilizzati
mov r4, r0              @ x
mov r5, r1              @ base v
mov r6, r2              @ n
mov r0, #0              @ sum (init 0)
mov r7, #1              @ xallan (inizialmente x alla 0 = 1)
mov r1, #0              @ i (init 0)
loop:   cmp r1, r6            @ controlla fine ciclo (i=n)
beq esci                @ in caso esci
ldr r2, [r5,r1, lsl #2]       @ carica v[i]
mul r3, r2, r7          @ v[i] * xallan
add r0, r0, r3          @ sum += v[i] * xallan
mul r7, r7, r4          @ xallan *= n
add r1, r1, #1          @ i++
b loop                  @ prossima iterazione
esci:   pop {r4-r7}           @ ripristina registri salvati
mov pc, lr              @ ritorna al chiamante, r0 contiene già il giusto valore di ret
```

Seconda funzione: sommaT

```
.global sommat
.type sommat, %function       @ param: x, base matrice, n, m (r0 - r3)

sommat: push {r4-r9,lr}       @ serve salvare anche lr perche' chiamo somma e malloc
mov r4, r0             @ x
mov r5, r1             @ base matrice
mov r6, r2             @ n righe
mov r7, r3             @ m colonne
mov r0, r6, lsl #2     @ n * sizeof(int)
bl malloc              @ alloca la memoria per il vettore risultato
mov r8, r0             @ indirizzo vettore
mov r9, #0             @ i, variabile di iterazione
loopt:  cmp r9, r6             @ itero se i<n
beq escit              @ altrimenti esco
mov r0, r4             @ preparo param per chiamare somma: x
mov r1, r5             @ indirizzo del vettore
mov r2, r7             @ numero degli elementi (elem per colonna: m)
bl somma               @ chiamo somma
str r0, [r8, r9, lsl #2]      @ memorizzo il risultato alla posizione i del vettore
add r5, r7, lsl #2     @ faccio puntare alla prossima riga (somma m*sizeof(int) byte)
add r9, r9, #1         @ i++
b loopt                @ loop
escit:  mov r0, r8             @ restituisco l'indirizzo del nuovo vettore risultato
pop {r4-r9,pc}         @ ripristino i registri, passando LR -> PC (return)
```