Corso di Laurea in Informatica, A.A. 2021/2022

Architetture degli Elaboratori e Sistemi Operativi (AESO corso A e B)

Secondo Appello – 23 Giugno 2022

Scrivere in modo comprensibile e chiaro rispondendo alle domande/esercizi riportati in questo foglio. Sviluppare le soluzioni dei primi due esercizi (prima parte) in un foglio separato rispetto alla soluzione degli ultimi due esercizi (seconda parte), in modo da facilitare la correzione da parte dei docenti. Riportare su tutti i fogli consegnati nome, cognome, numero di matricola e corso (A o B).

Parte 1

Esercizio 1

La procedura **EL * aggiungi(char c, EL * lista)** lavora su una lista di elementi EL che sono sequenze di tre campi:

- un carattere **c**, rappresentato come intero da 32 bit. Il codice ASCII è contenuto nel byte meno significativo della parola;
- un intero **occ**, di 32 bit, che rappresenta un numero di occorrenze del carattere c;
- un puntatore **next**, anch'esso di 32 bit, che rappresenta il puntatore al prossimo elemento della lista.

La procedura riceve in ingresso un carattere ASCII e un puntatore alla lista:

- se il puntatore è NULL (ovvero 0), crea un elemento EL con il campo **c** inizializzato con il carattere ricevuto come primo parametro, il campo **occ** uguale a 1, il campo **next** uguale a NULL, e restituisce l'indirizzo dell'elemento appena creato;
- se il puntatore non è NULL, scorre la lista fino a che non trova un elemento con il campo **c** uguale al carattere passato come primo argomento, oppure finché non si raggiunge la fine della lista senza trovare il carattere cercato. Nel primo caso, aggiorna il campo **occ** (**occ = occ + 1**) e restituisce il secondo parametro (puntatore alla lista). Nel secondo caso, aggiunge (in testa alla lista) un nuovo elemento **EL** con il campo **c** uguale al primo parametro, il campo **occ** a 1 e il campo **next** settato al puntatore alla lista, e ne restituisce l'indirizzo.

Si richiede di fornire il codice Assembler della funzione **aggiungi** che rispetti tutte le convenzioni per l'implementazione delle funzioni/procedure ARMv7.

Esercizio 2

Si consideri il seguente codice assembler:

```
 1  f:     ldr r0, [r0]
 2         ldr r2, [r1],#4
 3  loop: cmp r2, #0
 4         beq fine
 5         ldr r3, [r0]
 6         str r3, [r0],#4
 7         sub r2, r2, #1
 8         b loop
 9  fine: mov r0, r3
10         mov pc, lr
```

Di questo codice, eseguito sul processore pipeline completo visto a lezione, indicare:

- se ci sono dipendenze logiche e di controllo ed eventualmente quali sono;
- supponendo che il ciclo venga eseguito 10 volte, quale è il numero delle istruzioni eseguite;
- quale è il tempo di complemento del codice, inteso come numero di cicli di clock trascorsi fra il fetch della prima istruzione ed il write-back dell'ultima;
- quale è il CPI relativo a questo programma.

Parte 2

Esercizio 3

3-a) Descrivere che cosa si intende per gerarchia di memoria. Fornire un esempio di gerarchia di memoria con almeno 3 livelli diversi indicando un ordine di grandezza per i tempi di accesso per ogni livello (massimo 6 righe).

3-b) Si consideri un'architettura con ciclo di clock di 2 GHz ed una cache di capacità 2784 parole set associativa a 2 vie. La parola di memoria è di 24 bit, ed il numero di parole per ogni blocco di cache è 4. Il tempo per un cache hit è di 2 cicli di clock, mentre un cache miss costa in media 30 cicli di clock. Supponendo che vengano effettuate, in sequenza ordinata, delle operazioni di load dei seguenti indirizzi esadecimali e che in caso di conflitti sul blocco di cache l'algoritmo di sostituzione usato sia LRU (Least Recently Used):

0x0A2B1C, 0x0F4B18, 0x001F00, 0x001F0C, 0x011F0C, 0x0A2B14, 0x000B10, 0x000B14, 0x0A2B18

a) Determinare la lunghezza in bit del TAG, del Set (che determina il numero di linee di cache) e dell'offset;
b) Mostrare il contenuto delle linee di cache interessate al termine della sequenza di load supponendo che all'inizio delle operazioni la cache contenga solamente il blocco a cui appartiene l'indirizzo 0x0A2B1C.
c) Calcolare il miss rate generato dalla sequenza di operazioni.
d) Calcolare il tempo medio di accesso in memoria (AMAT) della sequenza di operazioni.

Esercizio 4

4-a) Descrivere in massimo 6 righe il concetto di stato sicuro e dire quali sono le condizioni che determinano lo stallo.

4-b) In un sistema che utilizza la tecnica di prevenzione dinamica dello stallo tramite l'algoritmo del banchiere sono presenti i processi A, B, C, D, E che utilizzano risorse dei tipi R1, R2, R3, R4. Al tempo *T* il sistema ha raggiunto lo stato sicuro mostrato nelle tabelle seguenti.

Assegnazione attuale				
	R1	R2	R3	R4
A	1	1	-	1
B	2	-	-	1
C	-	1	1	1
D	1	1	1	-
E	-	1	2	-

Esigenza Attuale				
	R1	R2	R3	R4
A	0	1	3	1
B	0	2	2	2
C	0	0	2	1
D	0	0	1	2
E	1	2	0	3

Molteplicità			
R1	R2	R3	R4
4	5	6	5

Disponibilità			
0	1	2	2

Si considerino i seguenti casi **in modo alternativo**:

a) Al tempo *T* il processo C richiede 2 risorse di tipo R3. Dire, applicando l'algoritmo del banchiere, se lo stato risultante è sicuro e quale sarà lo stato del processo C (esecuzione o attesa) in seguito a tale richiesta.
b) Al tempo *T* il processo A richiede 2 risorse di tipo R3. Dire, aplicando l'algoritmo del banchiere, se lo stato risultante è sicuro e quale sarà lo stato del processo A (esecuzione o attesa) in seguito a tale richiesta.

Soluzione Esercizio 1

```
        .text
        .global aggiungi
        .type aggiungi,%function
        @ r0 = c , r1 = lista

aggiungi: mov r3, r1              @ lista (salvato per restiuirlo)
loop:     cmp r1,#0               @ confronta puntatore alla lista
          beq fine                @ o è vuota o non ho trovato il carattere
          ldr r2, [r1]            @ altrimenti carica carttere c
          cmp r2, r0              @ controlla se è quello cercato
          beq trovato             @ semmai incrementa e ritorna
          ldr r1, [r1,#8]         @ continua col next
          b loop
trovato:  ldr r2, [r1,#4]         @ carica occ
          add r2, r2, #1          @ incrementalo
          str r2, [r1, #4]        @ salvalo
          mov r0, r3              @ restiusci vecchio puntatore
          mov pc, lr              @ ritorno
fine:     push {r0,r3,lr}         @ salva c lista e ritorno
          mov r0, #12
          bl malloc
          pop {r2,r3,lr}
          str r2, [r0]            @ c
          mov r2, #1
          str r2, [r0, #4]        @ occ = 1
          str r3, [r0, #8]        @ next = lista (originale)
          mov pc, lr              @ return (r0 punta già al nuovo elemento)
```

Soluzione Esercizio 2

Dipendenze logiche

- 2->3, il risultato della LDR è letto dalla CMP (che di fatto fa una SUB).
- 5->6, il risultato della LDR è letto nella STR

Dipendenze sul controllo

- B LOOP, salto sempre preso
- BEQ FINE, salto preso una volta sola alla fine del ciclo for, negli altri casi è un salto non preso

Numero di istruzioni eseguite:

- 2 (prima del loop) + 10 *6 (corpo del loop) + 2 (cmp e beq fine loop) + 2 (dopo il loop) = 66
  la beq conta come istruzione priva di effetto 10 volte e con effetto l'undicesima

Tempo di completamento

- In teoria: riempimento del pipeline (#stadi – 1) cicli + una istruzione a ciclo di clock per lo steady state. Farebbero: 4 + 66 cicli
- Vanno considerati i ritardi introdotti dalle dip sui dati (bolla da 1 per la dip 2->3, una sola volta, bolla da 1 per la dip 5->6 in tutte le iterazioni del ciclo) e sul controllo (bolla da 2 per il salto preso B LOOP, 11 volte, e bolla da 2 per la BEQ FINE, una volta sola). Dunque vanno aggiunti (1+2*11+1*10+2) cicli al totale precedente
- In totale: 70 cicli + 35 cicli = 105 cicli per eseguire 56 istruzioni

CPI

E' dato del numero di cicli impiegati / numero delle istruzioni, quindi

- CPI = 105/66 = 1,59

Soluzione Esercizio 3:

3-b)

a) La cache ha $\frac{2784}{4*2} = 348$ linee. Per indicizzarle servono $\lceil \log_2 348 \rceil = 9$ bit, mentre l'offset di blocco è dato da $\lceil \log_2 (4 * 3) \rceil = 4$ bits (2 bit per indicizzare le 4 parole del blocco e 2 bit per il byte offset). Di conseguenza il numero di bit per il TAG è $24 - 9 - 4 = 11$.
Quindi ogni indirizzo da 24 bit viene visto come una tripla di bit < TAG-11bit, Set-9bit, Off-4bit >.

b)
Determiniamo la tripla di bit per alcuni degli indirizzi forniti:
0x0A2B1C= 0000 1010 0010 1011 0001 1100 → <00001010001, 010110001, 1100>
0x0F4B18 = 0000 1111 0100 1011 0001 1000 → <00001111010, 010110001, 1000> stesso set del precedente
0x001F00 = 0000 0000 0001 1111 0000 0000 → <00000000000, 111110000, 0000>
0x011F0C = 0000 0001 0001 1111 0000 1100 → <00000001000, 111110000, 1100> stesso set del precedente
0x000B10 = 0000 0000 0000 1011 0001 0000 → <00000000000, 010110001, 0000> conflitto con 0x0A2B1C

Situazione iniziale della cache:

	Way 1					Way 0				
Set	TAG	W3	W2	W1	W0	TAG	W3	W2	W1	W0
010110001						00001010001	0x0A2B1C	0x0A2B18	0x0A2B14	0x0A2B10
…										

Situazione appena dopo la sequenza 0x0A2B1C, 0x0F4B18, 0x001F00, 0x001F0C, 0x011F0C, 0x0A2B14

	Way 1					Way 0				
Set	TAG	W3	W2	W1	W0	TAG	W3	W2	W1	W0
010110001	00001111010	0x0F4B1C	**0x0F4B18**	0x0F4B14	0x0F4B10	00001010001	**0x0A2B1C**	0x0A2B18	**0x0A2B14**	0x0A2B10
…										
111110000	00000001000	**0x011F0C**	0x011F08	0x011F04	0x011F00	00000000000	**0x001F0C**	0x001F08	0x001F04	**0x001F00**

Situazione al termine della sequenza 0x0A2B1C, 0x0F4B18, 0x001F00, 0x001F0C, 0x011F0C, 0x0A2B14, 0x000B10, 0x000B14, 0x0A2B18

	Way 1					Way 0				
Set	TAG	W3	W2	W1	W0	TAG	W3	W2	W1	W0
010110001	00001010001	0x0A2B1C	**0x0A2B18**	0x0A2B14	0x0A2B10	00000000000	0x000B1C	0x000B18	**0x000B14**	**0x000B10**
…										
111110000	00000001000	**0x011F0C**	0x011F08	0x011F04	0x011F00	00000000000	0x001F0C	0x001F08	0x001F04	**0x001F00**

b) I miss avvengono per le operazioni di load degli indirizzi 0x0F4B18, 0x001F00, 0x011F0C, 0x000B10, 0x0A2B18. Di questi gli ultimi 2 provocano l'esecuzione dell'algoritmo LRU con il rimpiazzamento di due blocchi. Il blocco contenente l'indirizzo 0x0A2B18 viene espulso e poi ricaricato (capacity miss). Il miss rate è quindi 5/9 = 55.55%.

c) L'AMAT è dato da $HitTime + MissRate * MissPenalty = \left(2 * 0.5 + \frac{5}{9} * 30 * 0.5\right) = 9.33\ ns$

Soluzione Esercizio 4:

4-b)

a) Stato raggiunto dopo l'assegnazione di 2 istanze di R3 al processo C:

Assegnazione attuale				
	R1	R2	R3	R4
A	1	1	-	1
B	2	-	-	1
C	-	1	3	1
D	1	1	1	-
E	0	1	2	0

Esigenza Attuale				
	R1	R2	R3	R4
A	0	1	3	1
B	0	2	2	2
C	0	0	0	1
D	0	0	1	2
E	1	2	0	3

Molteplicità			
R1	R2	R3	R4
4	5	6	5

Disponibilità			
0	1	0	2

Il processo C può terminare, il vettore disponibilità diventa: 0,2,3,3
Il processo D può terminare, il vettore disponibilità diventa: 1,3,4,3
Il processo B può terminare, il vettore disponibilità diventa: 3,3,4,4
Il processo E può terminare, il vettore disponibilità diventa: 3,4,6,4
Il processo A può terminare, il vettore disponibilità diventa: 4,5,6,5

Quindi lo stato è sicuro e la richiesta di assegnazione di risorse è accolta. Il processo C resta in stato di esecuzione.

b) Stato raggiunto dopo l'assegnazione di 2 istanze di R3 al processo A:

Assegnazione attuale				
	R1	R2	R3	R4
A	1	1	2	1
B	2			1
C		1	1	1
D	1	1	1	
E		1	2	

Esigenza Attuale				
	R1	R2	R3	R4
A	0	1	1	1
B	0	2	2	2
C	0	0	2	1
D	0	0	1	2
E	1	2	0	3

Molteplicità			
R1	R2	R3	R4
4	5	6	5

Disponibilità			
0	1	0	2

Nessun processo ha esigenza attuale minore o uguale alla disponibilità

Quindi lo stato risultante non è sicuro e la richiesta di assegnazione di risorse non può essere accolta. Il processo A viene messo in stato di attesa.