Corso di Laurea in Informatica, A.A. 2022/2023

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

Primo Appello – 17 Maggio 2023

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).

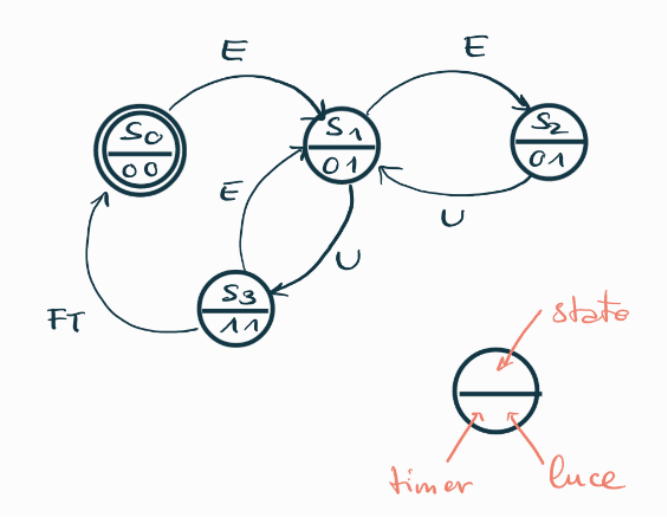
Parte 1

Esercizio 1

Si fornisca il codice assembler ARMv7 di una funzione **int hash(char * s)** che, data una sequenza di caratteri ASCII null terminated, restituisce il numero ottenuto dalla giustapposizione degli ultimi due bit di ciascun carattere, nell'ordine, modulo 16. Esempio: stringa "abc". Il codice ASCII di 'a' è 97 (1100001 in binario, 64 + 32 +1). I due bit meno significativi sono 01. Analogamente il codice di 'b' è 98 e quello di 'c' è 99 quindi gli ultimi due bit sono rispettivamente 10 e 11. La giustapposizione dei 6 bit così ottenuti risulta quindi 011011_2, che modulo 16_{10}, ovvero prendendo gli ultimi 4 bit, risulta essere $1011_2 = 11_{10}$).

Esercizio 2

L'automa di Moore rappresentato in figura accetta come ingressi gli eventi E (entrata), U (Uscita) e FT (fine timer) e ha come uscite una T (richiesta timeout) e L (luce accesa/spenta). L'automa governa in modo autonomo l'accensione e lo spegnimento di una luce in un ambiente che ha una capacità massima di due persone. Gli stati sono etichettati con il valore delle uscite T e L, nell'ordine. Lo stato etichettato con S_0 è lo stato iniziale.



Si descriva nel modo più dettagliato possibile la rete sequenziale che implementa l'automa.

Parte 2

Esercizio 3

3-a) Descrivere in massimo 6 righe la differenza tra la semantica MESA ed Hoare delle variabili di condizione.

3-b) Un sistema dispone di N risorse non prerilasciabili di molteplicità uno. Nel sistema sono presenti due tipologie di thread che utilizzano le risorse: il primo tipo (T1) acquisisce una sola risorsa alla volta e la detiene per più tempo, il secondo tipo (T2) acquisisce due risorse contemporaneamente e le detiene per poco tempo. I thread di tipo T2 hanno priorità maggiore dei thread di tipo T1 nell'acquisizione delle risorse. Al rilascio di una o più risorse da parte di un thread, le risorse disponibili vengono assegnate secondo la seguente politica:

a) Se non ci sono thread di tipo T2 in attesa, le risorse disponibili vengono assegnate agli eventuali thread T1 in attesa.
b) Se ci sono thread di tipo T2 in attesa, e ci sono almeno 2 risorse disponibili, vengono assegnate ai thread T2 in attesa finché è possibile assegnarne 2 a ciascuno di essi.
c) Se ci sono thread di tipo T2 in attesa ma non ci sono almeno 2 risorse disponibili, le risorse non vengono assegnate a nessun thread.

I thread T1 eseguono il seguente pseudo-codice:

Pseudo-codice del thread T1:

```
while(1) {
  acquisisciRisorse(1, R1,NULL);
  // usa R1
  rilasciaRisorse(1, R1, NULL);
}
```

Pseudo-codice del thread T2:

```
while(1) {
  acquisisciRisorse(2, R1, R2);
  // usa R1 ed R2
  rilasciaRisorse(2, R1, R2);
}
```

Si chiede di implementare (in pseudo-codice C) le funzioni
*acquisisciRisorse(int n, res_t *R1, res_t *R2)* e
*rilasciaRisorse(int n, res_t *R1, res_t *R2)*

utilizzando i semafori (invocando le funzioni P() e V()). Si supponga esista una funzione *assegna()* che ritorna una risorsa disponibile da assegnare (ad esempio, per assegnare R1 all'interno della funzione acquisisciRisorse() è possibile scrivere R1 = assegna()). Analogamente per il rilascio (ad esempio, rilascia(R1)). Indicare per ogni semaforo utilizzato il valore di inizializzazione.

Esercizio 4

4-a) Descrivere in massimo 6 righe la traduzione degli indirizzi in un sistema che implementa la segmentazione con paginazione multilivello.

4-b)) Un sistema gestisce la memoria con paginazione. Gli indirizzi sono a 16 bit, le pagine logiche e fisiche di 8 KB. Nella figura seguente sono riportate la tabella delle pagine di due processi P1 e P2 in esecuzione nel sistema al tempo T. Il simbolo "X" in tabella indica che la pagina non è caricata in memoria.

Tabella del processo P1

0	X
1	3
2	9
3	4
4	10
5	X
6	6
7	X

Tabella del processo P2

0	X
1	X
2	5
3	8
4	7
5	X
6	X
7	11

1. Si chiede di effettuare la traduzione degli indirizzi logici: 50.892 e 16.000 per il processo P1 e 7.211 e 23.184 per il processo P2. Indicare, il numero di pagina fisica, l'offset all'interno della pagina fisica e l'indirizzo fisico tradotto.
2. Supponendo che la memoria fisica totale del sistema sia di 96 KB e che il SO utilizzi le pagine fisiche 0,1,2 a lui riservate, riportare tutte le entrate dell'Inverted Page Table (Tabella delle Pagine Inversa) utilizzata dal SO al tempo T.

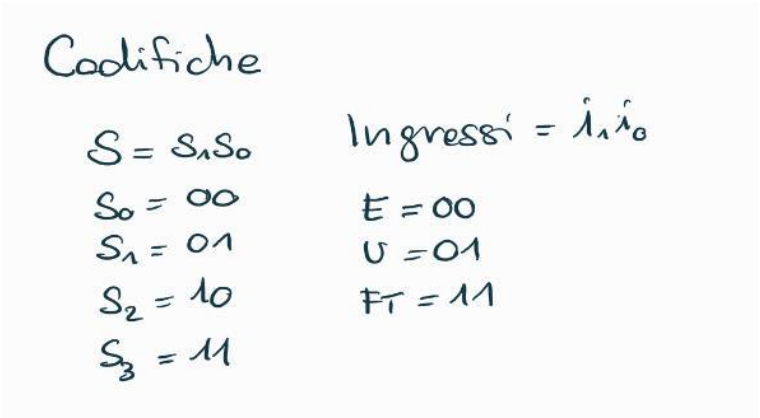
SOLUZIONE:

Esercizio 1

```
.text
.global hash
.type hash,%function @ r0 -> indirizzo stringa, null terminated
hash: mov r3, #0 @ valore da restituire
loop: ldrb r1, [r0] @ primo carattere in r1
cmp r1, #0 @ controllo se sono alla fine
beq fine
lsl r3, r3, #2 @ shifta il ris di due bit a sn
and r1, r1, #3 @ prendi i 2 bit meno significativi
add r3, r3, r1 @ mettili nel risultato
add r0, r0, #1 @ indirizzo prossimo carattere
b loop
fine: and r0, r3, #15 @ prendo gli ultimi 4 bit
mov pc, lr
```

Esercizio 2

Dobbiamo realizzare una rete di Moore, costituita da due reti combinatorie (calcolo del prossimo stato interno a partire dallo stato interno corrente e dagli ingressi, e calolo delle uscite a partire dal solo stato interno) e da un registro di stato di due bit, nel nostro caso, visto che dobbiamo enumerare 4 stati diversi. Scegliamo come codifica di stati ed ingressi:

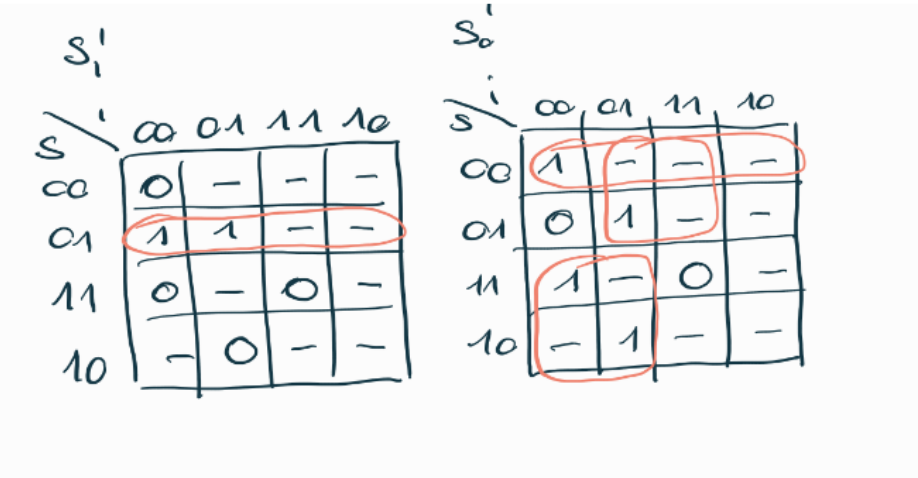


La codifica porta alle seguenti tabelle di verità per la funzione prossimo stato interno (a sinistra) e per la funzione delle uscite (a destra):

$s_1 s_0$ stato	ingresso	$s_1' s_0'$ stato'
00	00	01
01	00	10
01	01	11
10	01	01
11	00	01
11	11	00

stato	uscite T L
00	00
01	01
10	01
11	11

Utilizziamo le mappe di Karnaugh per la funzione che calcola il prossimo stato interno:



Da cui si evince che:

$$s_1' = \bar{s}_1 s_0$$

$$s_0' = \bar{s}_1 \bar{s}_0 + s_1 \bar{i}_1 + \bar{s}_1 i_0$$

Per la funzione delle uscite, osserviamo che la richiesta timer T è vera solo nello stato rappresentato come 11 mentre il segnale che accende le luci è sempre 1 tranne che nello stato rappresentato come 0 (quindi codifichiamo lo zero e neghiamo l'uscita):

$$T = s_1 s_0$$

$$L = \overline{\bar{s}_1 \bar{s}_0}$$

La funzione che calcola il prossimo stato interno ha due livelli di logica, quella che calcola le uscite uno solo, ma va preso il massimo quindi la rete lavora con un ciclo di clock pari almeno al tempo di stabilizzazione di una rete a due livelli di logica più il tempo necessario alla scrittura del registro di stato (che è di 2 bit).

Esercizio 3-b

// variabili globali utilizzate dalle procedure *acquisiciRisorse e rilasciaRisorse()*.
sem_t mutex; // semaforo di mutua esclusione inizializzato ad 1
struct coda_t { sem_t *selfSem ; res_t *R1, res_t *R2 } ;
coda_t QT1; // coda FIFO utilizzata per memorizzare le info del thread T1 da sospendere sul semaforo selfSem
coda_t QT2; // analoga a QT1 ma utilizzata per i thread T2
int risorseDisponibili = N; // contatore
int T1inattesa = 0; // contatore
int T2inattesa = 0; // contatore

// si suppone che siano implementate le funzioni:
- creaSemaforo() che crea ed inizializza un semaforo a 0 restituendone il puntatore
- distruggiSemaforo(sem_t *) che distrugge il semaforo passato come argomento

```
void acquisisciRisorse(int n, res_t *R1, res_t *R2) {

 int devoattendere=0;
 sem_t *selfSem;

 mutex.P();
 if (n==1) { // thread T1
  if (risorseDiponibili==0 || T2inattesa>0) {
    T1inattesa++;
    devoattendere=1;
    selfSem = creaSemaforo() ; //selfSem e' iniz. a 0
    QT1.push(selfSem, R1, NULL) ;
  } else { risorseDisponibili--;  R1 = assegna(); }
 } else {  // thread T2
  if (risorseDiponibili<2) {
    T2inattesa++;
    devoattendere=1;
    selfSem = creaSemaforo() ; //selfSem e' iniz. a 0
    QT2.push(selfSem, R1, R2) ;
  } else {
     risorseDisponibili -=2;
     R1 = assegnaR(); R2 = assegnaR();}
 }
 mutex.V();
 if (devoattendere) {
   selfSem->P();
   distruggiSemaforo(selfSem);
 }
```

```
void rilasciaRisorse(int n, res_t *R1, res_t *R2) {

 mutex.P();
 risorseDisponibili += n;
 rilascia(R1);
 if (R2) rilascia(R2);

 while (T2inattesa>0 && risorseDisponibili>1) {
    T2inattesa--;
    risorseDisponibili -= 2;
    coda_t *e = QT2.pop();
    e->R1 = assegna();
    e->R2 = assegna();
    e->selfSem->V();
  }
  if (T2inattesa == 0) {
    while( T1inattesa>0 && risorseDisponibili>0) {
      T1inattesa--;
      risorseDisponibili --;
      coda_t *e = QT1.pop();
      e->R1 = assegna();
      e->selfSem->V();
    }
  }
  mutex.V();
}
```

Esercizio 4-b

1. Consideriamo l'indirizzo logico 16.000 del processo P1. Dato che le pagine sono di 8KB, abbiamo che l'entry della tabella delle pagine da considerare è (16.000 div 8192) = 1 cioè il frame fisico 3 mentre l'offset è dato da 16.000 % 8192 = 7808. Quindi l'indirizzo fisico è dato da: (3 x 8192) + 7808 = 32.384.

Processo	Indirizzo logico	Frame fisico	Offset	Indirizzo fisico
P1	50.892	6	1.740	50.892
P1	16.000	3	7.808	32.384
P2	7.211	Genera un fault di pagina.		
P2	23.184	5	6.800	47.760

2. Ogni elemento della Inverted Page Table (IPT) ha due campi: l'id del processo che detiene il corrispondente frame di pagina fisica, e l'id di pagina logica contenuta nel frame. Considerando che il sistema ha in tutto 12 frame di memoria fisica di cui i primi 3 sono occupati dal SO in modo permanente, dalle tabelle delle pagine dei processi P1 e P2 date si deduce la seguente IPT:

	ID del processo	Indice di pagina logica
0	SO	-
1	SO	-
2	SO	-
3	1	1
4	1	3
5	2	2
6	1	6
7	2	4
8	2	3
9	1	2
10	1	4
11	2	7