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).

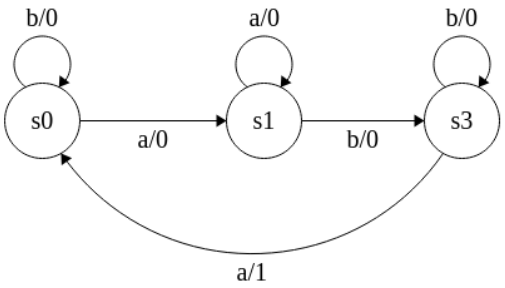
Parte 1

Esercizio 1

Si consideri l'automa in figura. L'automa riconosce le stringhe **a(b$^+$)a**[1] sull'alfabeto **{a,b}**. Di tale automa si chiede di fornire la descrizione completa della rete sequenziale che lo implementa:

a. Funzione che calcola l'uscita
b. Funzione che calcola il prossimo stato interno
c. Numero di bit del registro di stato
d. Schema dei collegamenti fra i componenti della rete sequenziale
e. Ritardo delle due funzioni (e di conseguenza lunghezza del ciclo di clock)

Le funzioni dovranno essere definite in termini di porte AND ed OR con massimo 8 ingressi.



Esercizio 2

Si fornisca il codice Assembler di una funzione che, dato un puntatore ad una stringa di caratteri terminata da un NULL (un carattere di codice 0) controlla se la parte iniziale della stringa è una delle stringhe riconosciute dall'automa dell'esercizio 1, ovvero se ha la forma **a(b$^+$)a**. La funzione restituisce 0 se la parte iniziale della stringa non è nella forma **a(b$^+$)a**, oppure *k* se lo è e la lunghezza del segmento **a(b$^+$)a** è di *k* caratteri. Si ricorda che il codice ASCII della 'a' è 97.

[1] il simbolo + indica uno o più occorrenze del carattere che lo precede. (b+) dunque indica tutte le stringhe di una o più occorrenze della lettera b.

Parte 2

Esercizio 3

3-a) Descrivere in massimo 6 righe l'algoritmo di sostituzione Second Chance.

3-b) In un sistema POSIX, un processo esegue un calcolo utilizzando *n+1* thread *T0, T1,…,Tn*. I thread *T1,…,Tn* (denominati Worker) eseguono un calcolo in modo indipendente, il thread *T0* raccoglie i risultati calcolati dai Worker e li aggrega per produrre il risultato finale. Il calcolo consiste nello svolgere un certo numero di iterazioni tutte uguali denominate round. Ad ogni round i thread *T1,…,Tn* depositano, al termine del calcolo relativo al round, il risultato calcolato in un buffer *B* di *n* posizioni e poi attendono di iniziare un nuovo round. Quando tutti i thread *T1,…,Tn* hanno completato il round, il thread *T0* svuota completamente il buffer B raccogliendo gli *n* risultati e solo dopo fa partire un nuovo round.
Il buffer *B* è implementato come un array ad *n* posizioni, ed il generico thread *Ti* ($1 \leq i \leq n$) deposita il risultato nell'elemento del buffer di indice *i-1*. I thread si sinconizzano attraverso variabili di condizione. Per operare sulle variabili di condizione e di mutua esclusione i thread usano esclusivamente le seguenti chiamate POSIX pthread_cond_wait, pthread_cond_signal, pthread_mutex_lock e pthread_mutex_unlock, inoltre è disponibile la funzione *getMyId()* che ritorna l'id del thread che la invoca ($0 \leq$ id $\leq n$). Si supponga che le variabili di mutex e di condizione siano state inizializzate correttamente e che esista una funzione *aggregaDati(B,n)* chiamata dal thread *T0* che restituisce il risultato aggregato utilizzando i risultati parziali contenuti nel buffer B.

Si chiede di implementare le funzioni *putResult(risultato)* invocata dai thread *T1,…Tn* e *getResults(&risultato)* invocata dal thread *T0*.

Esercizio 4

4-a) Descrivere in massimo 6 righe che cos'è e per cosa viene utilizzato il TLB (Translation Lookaside Buffer).

4-b) In un file system UNIX i blocchi del disco hanno ampiezza di 2K e gli i-node contengono 10 indirizzi diretti e 3 indirizzi indiretti (singolo, doppio e triplo). Tutti gli indirizzi hanno una lunghezza di 32 bit.
Si chiede di calcolare:

1) il numero di puntatori che possono essere contenuti in ogni blocco indice;
2) il numero di blocchi indirizzabili con indirizzamento indiretto semplice, doppio e triplo;
3) dati i file A di 85.000 byte, B di 1.445.000 byte e C di 4.500.000 byte, calcolare per ognuno di questi file il numero di blocchi indice di primo, secondo e terzo livello necessari per rappresentarlo.

Soluzione Esercizio 1

Codifichiamo le due lettere dell'alfabeto dato come a=0 e b=1. Inoltra scegliamo di codificare gli stati come s0=00, s1=01 ed s2=11.

Con queste convenzioni, la tabella di verità di stato successivo e uscita è la seguente:

S0	S1	x		S0'	S1'	Z
0	0	0		0	1	0
0	0	1		0	0	0
0	1	0		0	1	0
0	1	1		1	1	0
1	1	0		0	0	1
1	1	1		1	1	0

Di conseguenza possiamo derivare la forma in somma di prodotti per la funzione delle uscite come

Z = S0 and S1 and not(X)

e le due funzioni che definiscono i due bit dello stato interno come:

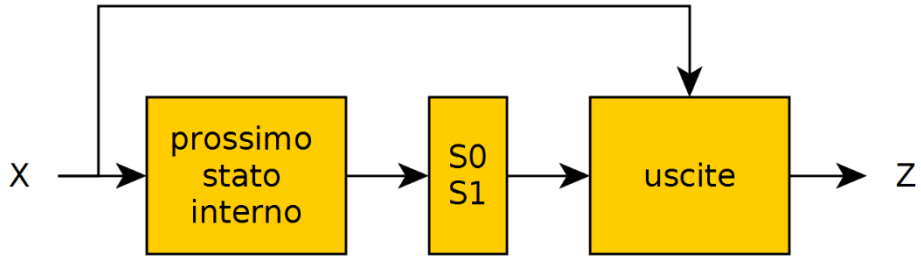
S0' = not(S0) and S1 and X or S0 and S1 and X = (not(S0) or So) and (S1 and X) = S1 and X

S1' = not(S0) and not(S1) and not(X) or not(S0) and S1 and not(X) or not(S0) and S1 and X or S0 and S1 and X = not(S0) and not(X) or S1 and X

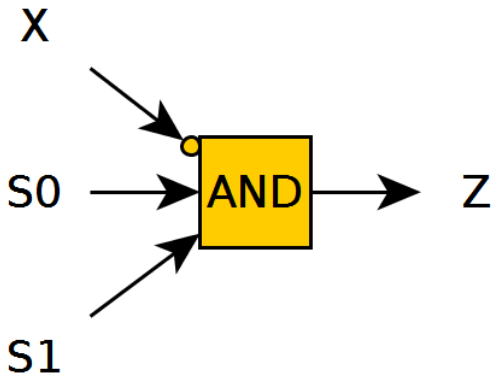
Da questo si conclude che la rete che calcola le uscite ha un unico livello di porte (solo una porta and) e la rete che calcola il prossimo stato interno ha due livelli di porte (due porte and e una porta or). Quindi il ritardo massimo e quello di due livelli di porte, cui va aggiunto il ritardo per la scrittura nel registro.

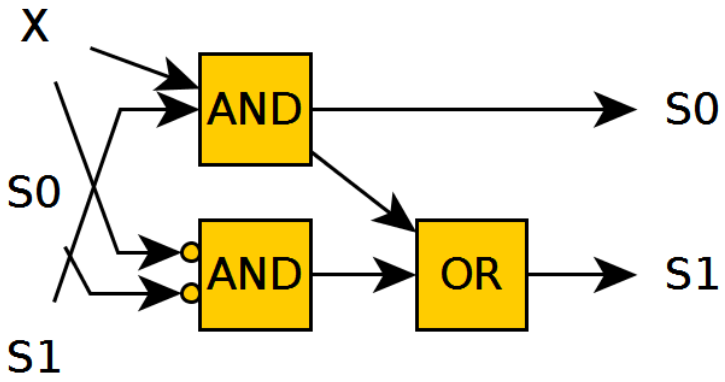
Il ciclo di clock dovrà pertanto essere pari alla somma di questi ritardi.

Lo schema della rete sarà dunque:



Le due reti (uscite e prossimo stato interno) saranno rispettivamente:





Soluzione Esercizio 2

Il codice della funzione Assembler richiesto è descritto dal codice C:

```
int ismatch(char * s) {
  int count = 0;
  if(*s != 'a')
    return(count);
  s++;
  count++;
  if(*s != 'b')
    return(0);
  count++;
  while(*s == 'b') { s++; count++; }
  if(*s == 'a')
    return(count);
  else
    return(0);
}
```

Il corrispondente codice Assembler può essere dunque scritto come segue:

```
      .text
.global ismatch
.type ismatch, %function

@ r0 -> indirizzo della stringa
```

```
ismatch:mov r1, #0    @ contatore caratteri
ldrb r3, [r0]         @ primo carattere in r3
add r1, r1, #1  @ count ++
cmp r3, #97     @ controlla se fosse una a
bne nontrovato  @ se non lo fosse, restituisci 0
     add r0, r0, #1  @ prossimo catattere
     ldrb r3, [r0]
add r1, r1, #1  @ +1 carattere
cmp r3, #98     @ se non e' una b
     bne nontrovato
add r0, r0, #1  @ prossimo carattere
bbb: ldrb r3, [r0]   @ in r3
add r1, r1, #1  @ count ++
cmp r3, #98     @ e' una 'b' ?
addeq r0, r0, #1@ in caso continua al prossimo carattere
beq bbb
cmp r3, #97     @ controlla ultima a
bne nontrovato
mov r0, r1 @ restituisci count
mov pc, lr
nontrovato:
movne r0, #0
movne pc, lr
```

Soluzione Esercizio 3:

3-b)

```
pthread_mutex_t mux;
pthread_cond_t attesaRisultati;
pthread_cond_t nextRound;

int datiDisponibili=0;
int nSospesi=0;

void putResult(risultato) { // invocate da T1,…, Tn

    pthread_mutex_lock(&mux);
    B[getMyId()-1] = risultato;
    datiDisponibili++;
    if (datiDisponibili == n) pthread_cond_signal(&attesaRisultati);
    nSospesi ++;
    while (nSospesi>0) pthread_cond_wait(&nextRound, &mux);
    pthread_mutex_unlock(&mux);

}

void function getResult(&risultato){  // invocate da T0

    pthread_mutex_lock(&mux);
    while (datiDisponibili < n) pthread_cond_wait(&attesaRisultati,&mux);

    risultato = aggregaDati(B, n);
    datiDisponibili=0;
    nSospesi=0;
    for (int i= 0; i< n; ++i) pthread_cond_signal(&nextRound);

    pthread_mutex_unlock(&mux);
}
```

Soluzione Esercizio 4:

4-b)

1. Dato che ogni indirizzo occupa 4 byte, il numero di puntatori contenuti in un blocco indice è 2^11/2^2 = 512 blocchi
2. Il numero di blocchi indirizzabili con indirizzamento indiretto semplice è 512, mentre il numero di blocchi indirizzabili con indirizzamento indiretto doppio è pari a 2^9*2^9=2^18= 256K blocchi. Il numero di blocchi indirizzabili con indirizzamento indiretto triplo è pari a 2^27= 128M blocchi.

3. Data la dimensione dei file A, B e C, in nessun caso si deve ricorrere all'indirizzamento indiretto triplo.
   3.a) Il file A occupa $\lceil 85.000 \textbf{ div } 2048 \rceil = 42$ blocchi dati, dei quali:

   - 10 sono indirizzati dai puntatori diretti (indici 0 .. 9 nello i-node),

   - 32 sono indirizzati attraverso un unico blocco indice di primo livello, a sua volta indirizzato dal puntatore indiretto semplice (indice 10 nello i-node).

   Pertanto per la rappresentazione del file è necessario un solo blocco indice.

   3.b) Il file B occupa $\lceil 1.445.000 \textbf{ div } 2048 \rceil = 706$ blocchi logici, dei quali:

   - 10 sono indirizzati dai puntatori diretti (indici 0 .. 9 nello i-node),

- 696 sono indirizzati attraverso $\lceil 696 \ \mathbf{div} \ 512 \rceil = 2$ blocchi indice di primo livello, dei quali:

  - il primo, che indirizza 512 blocchi dati, è a sua volta indirizzato dal puntatore indiretto semplice (indice 10 nello i-node).

  - i rimanenti 184 blocchi dati sono indirizzati dai puntatori di indice 0,..,183 di un unico blocco indice di primo livello puntato dal puntatore 0 di un blocco indice di secondo livello, a sua volta indirizzato dal puntatore indiretto doppio (indice 11 nello i-node).

Pertanto per la rappresentazione del file sono necessari in totale 3 blocchi indice (2 di primo livello e 1 di secondo livello)

3.c) Il file C occupa $\lceil 4.500.000 \ \mathbf{div} \ 2048 \rceil = 2198$ blocchi logici, dei quali:

- 10 sono indirizzati dai puntatori diretti (indici 0 .. 9 nello i-node),

- 2188 sono indirizzati attraverso $\lceil 2188 \ \mathbf{div} \ 512 \rceil = 5$ blocchi indice di primo livello, dei quali:

  - il primo, che indirizza 512 blocchi dati, è a sua volta indirizzato dal puntatore indiretto semplice (indice 10 nello i-node).

  - i rimanenti 4 blocchi indice di primo livello sono indirizzati dai puntatori di indice 0 … 3 di un unico blocco indice di secondo livello, a sua volta indirizzato dal puntatore indiretto doppio (indice 11 nello i-node).

Pertanto per la rappresentazione del file sono necessari in totale 6 blocchi indice (5 di primo livello ed 1 di secondo livello).