

**PROVA SCRITTA DEL CORSO DI**  
**CALCOLATORI ELETTRONICI**  
**NUOVO ORDINAMENTO DIDATTICO**  
10 Febbraio 2009

NOME:

COGNOME:

MATRICOLA:

**ESERCIZIO 1 (10 punti)**

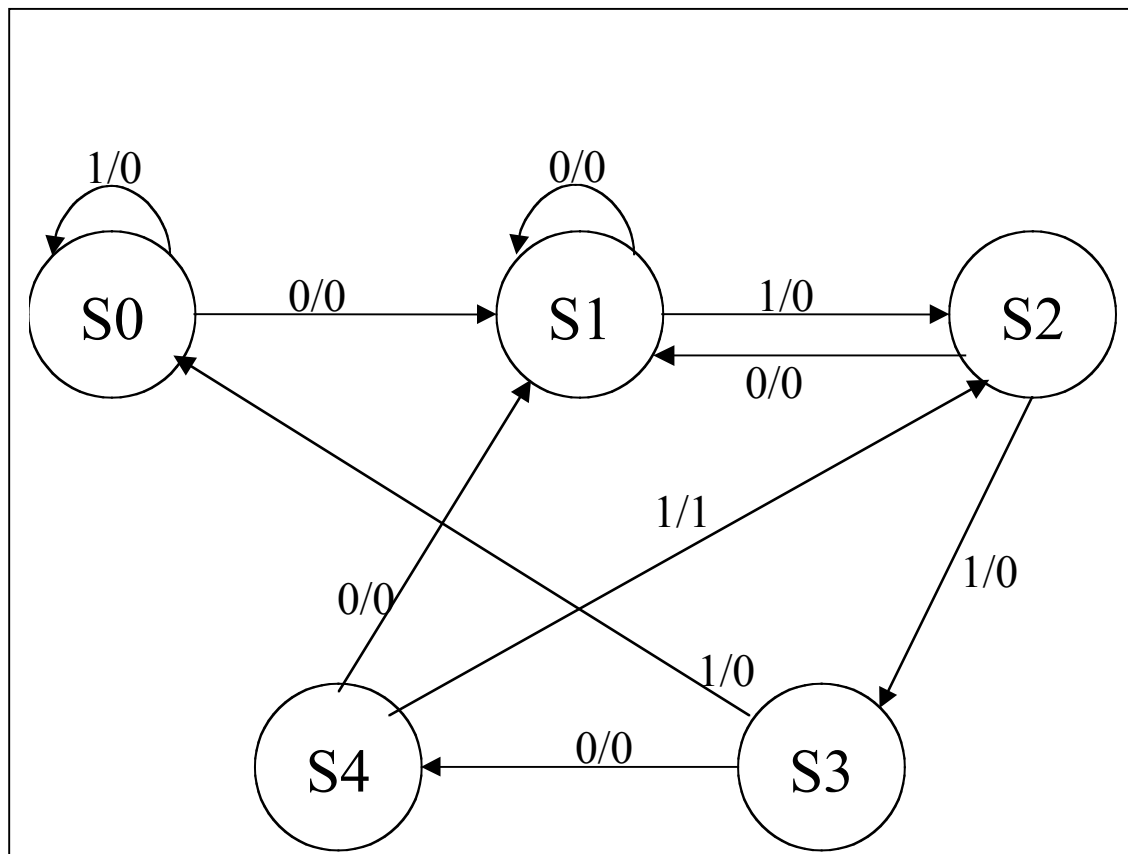
Progettare una rete sequenziale che presenti un ingresso X e un'uscita Z posta a 1 ogni volta che viene riconosciuta la sequenza 01101.

Si richiede:

- a) (6 punti) il diagramma degli stati, la tabella di flusso e la tabella delle transizioni;
- a) (4 punti) il calcolo delle forme minime delle variabili di eccitazione dei flip flop con le mappe di Karnaugh. Si usino flip flop JK. Calcolare anche la rete combinatoria per l'uscita Z.

**Soluzione.**

Il diagramma degli stati è il seguente:



La tabella di flusso è data da:

Stato presente	Stato successivo/Uscita	
	X=0	X=1
S0	S1/0	S0/0
S1	S1/0	S2/0
S2	S1/0	S3/0
S3	S4/0	S0/0
S4	S1/0	S2/1

Per codificare 5 stati occorrono tre flip flop. La codifica è la seguente:

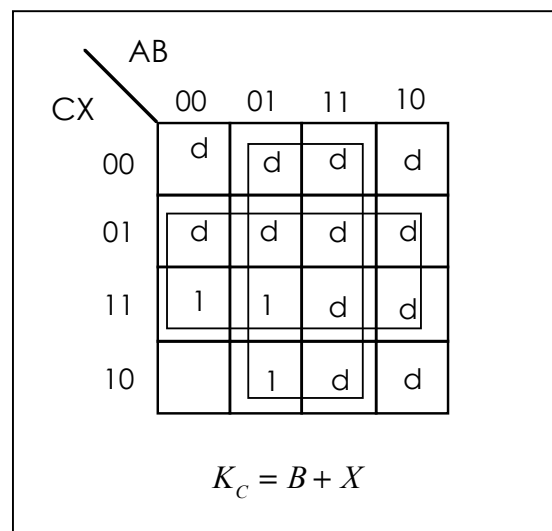
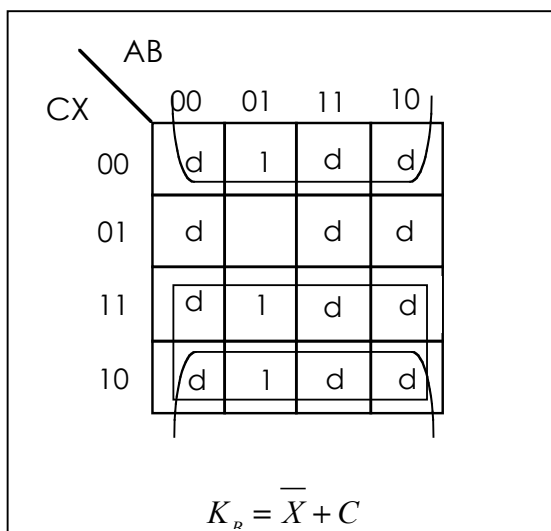
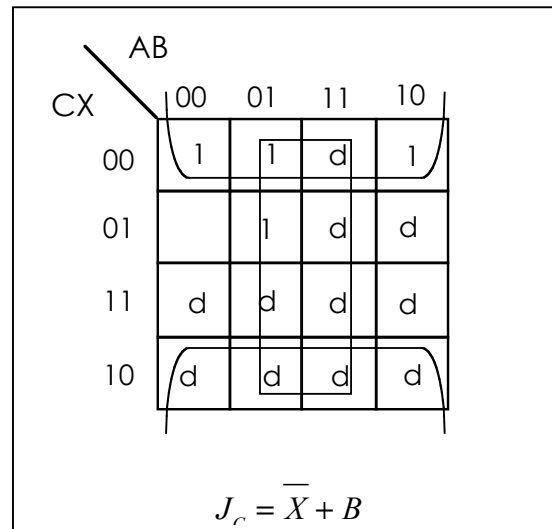
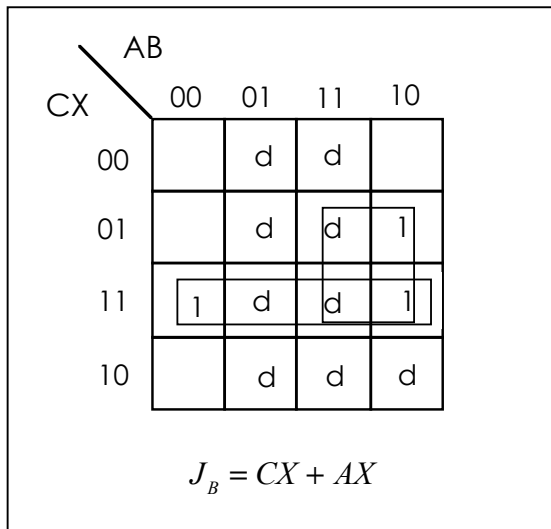
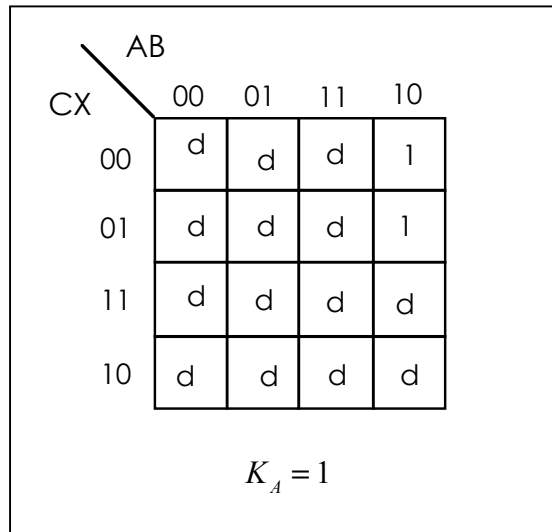
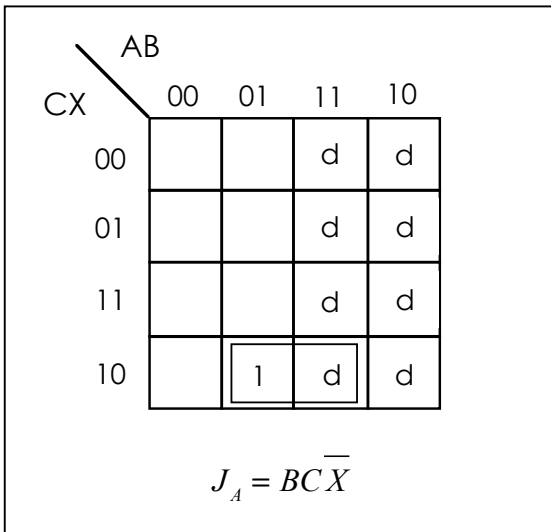
S0 → 0 0 0; ...; S4 → 1 0 0. Nel seguito indicheremo ciascun bit della codifica con le lettere A, B, C. L'apice indicherà il bit nell'istante successivo a quello considerato.

A partire dalla tabella di eccitazione del flip flop JK:

Q	Q'	J	K
0	0	0	D
0	1	1	D
1	0	D	1
1	1	D	0

A	B	C	X	A'	Ja	Ka	B'	Jb	Kb	C'	Jc	Kc	Z
0	0	0	0	0	0	D	0	0	D	1	1	D	0
0	0	0	1	0	0	D	0	0	D	0	0	D	0
0	0	1	0	0	0	D	0	0	D	1	D	0	0
0	0	1	1	0	0	D	1	1	D	0	D	1	0
0	1	0	0	0	0	D	0	D	1	1	1	D	0
0	1	0	1	0	0	D	1	D	0	1	1	D	0
0	1	1	0	1	1	D	0	D	1	0	D	1	0
0	1	1	1	0	0	D	0	D	1	0	D	1	0
1	0	0	0	0	D	1	0	0	D	1	1	D	0
1	0	0	1	0	D	1	1	1	D	0	0	D	1
1	0	1	0	D	D	D	D	D	D	D	D	D	D
1	0	1	1	D	D	D	D	D	D	D	D	D	D
1	1	0	0	D	D	D	D	D	D	D	D	D	D
1	1	0	1	D	D	D	D	D	D	D	D	D	D
1	1	1	0	D	D	D	D	D	D	D	D	D	D
1	1	1	1	D	D	D	D	D	D	D	D	D	D

Ora possiamo disegnare le mappe di Karnaugh



Infine, per quanto riguarda l'uscita Z:

		AB			
		00	01	11	10
CX	00			d	
	01			d	1
	11			d	d
	10			d	d

$$Z = AX$$

## ESERCIZIO 2 (7 punti)

I trasferimenti di parole a/dalla memoria di un calcolatore sono codificate utilizzando il codice di Hamming. Si consideri la stringa di 8 bit 11010111 (il bit meno significativo è a sinistra). Spiegando bene ogni passo del ragionamento:

1. (1 punto) calcolare il minimo numero di bit di controllo necessari per la codifica della parola;
2. (3 punti) codificare la stringa data;
3. (3 punti) imporre un errore nel terzo bit della stringa informazione e spiegare come l'errore viene rivelato e corretto per mezzo della codifica di Hamming.

### Soluzione.

- 1) Deve essere rispettata la condizione:

$$2^K \geq N + K + 1 \quad (1),$$

dove K è il numero di bit di controllo inseriti. Essendo N=8, il numero minimo di bit di controllo richiesto è 4.

- 2) Nella codifica di Hamming, la sequenza in ingresso presenta la seguente struttura:

c <sub>0</sub>	c <sub>1</sub>	b <sub>0</sub>	c <sub>2</sub>	b <sub>1</sub>	b <sub>2</sub>	b <sub>3</sub>	c <sub>3</sub>	b <sub>4</sub>	b <sub>5</sub>	b <sub>6</sub>	b <sub>7</sub>
		1		1	0	1		0	1	1	1

Dove c<sub>0</sub>...c<sub>3</sub> sono i quattro bit costituenti il vettore di controllo, e b<sub>0</sub>...b<sub>7</sub> gli otto bit trasmessi. Tali bit si ottengono con le seguenti operazioni

$$c_0 = b_0 \oplus b_1 \oplus b_3 \oplus b_4 \oplus b_6 = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0$$

$$c_1 = b_0 \oplus b_2 \oplus b_3 \oplus b_5 \oplus b_6 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 0$$

$$c_2 = b_1 \oplus b_2 \oplus b_3 \oplus b_7 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$$

$$c_3 = b_4 \oplus b_5 \oplus b_6 \oplus b_7 = 0 \oplus 1 \oplus 1 \oplus 1 = 1$$

La stringa codificata è 001110100111.

- 3) Nell'ipotesi di un errore sul terzo bit della stringa iniziale, la stringa ricevuta risulta: 001111100111. Per rivelare questo errore, bisogna ricalcolare i bit di controllo:

$$c'_0 = b_0 \oplus b_1 \oplus b_3 \oplus b_4 \oplus b_6 = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0$$

$$c'_1 = b_0 \oplus b_2 \oplus b_3 \oplus b_5 \oplus b_6 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1$$

$$c'_2 = b_1 \oplus b_2 \oplus b_3 \oplus b_7 = 1 \oplus 1 \oplus 1 \oplus 1 = 0$$

$$c'_3 = b_4 \oplus b_5 \oplus b_6 \oplus b_7 = 0 \oplus 1 \oplus 1 \oplus 1 = 1$$

Il passo successivo è calcolare il vettore di errore dato dalla differenza dei vettori di controllo c e c' (ricordiamo che somma e differenza tra bit producono lo stesso risultato):

$$e_0 = c_0 \oplus c'_0 = 0$$

$$e_1 = c_1 \oplus c'_1 = 1$$

$$e_2 = c_2 \oplus c'_2 = 1$$

$$e_3 = c_3 \oplus c'_3 = 0$$

Poiché il vettore risultante 0110 non è nullo, vi è un errore nella stringa di 12 bit e precisamente nella posizione indicata dal vettore di errore tradotto in notazione decimale. Il bit sbagliato nella stringa codificata è quindi il sesto (b<sub>2</sub>), che può venire dunque corretto.

### ESERCIZIO 3 (9 punti)

Si scriva il codice Assembly MIPS di una funzione che, ricevendo in ingresso l'indirizzo iniziale di un vettore di interi  $v$ , la sua dimensione  $N$ , un secondo vettore di interi  $w$ , la sua dimensione  $M$ , scriva nel vettore  $w$  il numero di occorrenze degli interi in  $v$ , da 0 a  $M-1$ . In altri termini,  $w[j]$  conterrà il numero di occorrenze del valore  $j$  nel vettore  $v$ . Si assuma per semplicità che il contenuto di  $w$  sia inizializzato a 0.

Allocazione dei parametri di ingresso:  $\&v[0] \rightarrow \$4$ ,  $N \rightarrow \$5$ ,  $\&w[0] \rightarrow \$6$ ,  $M \rightarrow \$7$ .

Il codice MIPS potrebbe implementare il seguente codice C:

```
void occorrenze(int v[], int N, int w[], int M)
{
    int i,j;

    for(i=0; i<N; i++)
    {
        j=v[i];
        if((j<M) && (j>=0))
            w[j]++;
    }
}
```

### Soluzione.

```
$8 ← i; $9 ← v[i]; $10 ← w[j]; $11 ← (v[i]<M); $12 ← (v[i]<0);
$13 ← &w[j]
```

```
occorrenze:  addi $29, $29, -24      #salvataggio contesto
              sw $8, 0($29)
              sw $9, 4($29)
              sw $10, 8($29)
              sw $11, 12($29)
              sw $12, 16($29)
              sw $13, 20($29)
              move $8, $0           #i=0
for:         beq $8, $5, exit
              lw $9, 0($4)          #j=v[i]
              addi $4, $4, 4        #aggiorno indici
              addi $8, $8, 1
              slt $11, $9, $7       #$11 ← (v[i]<M)
              beq $11, $0, for      #if !($11) then for
              slt $12, $9, $0       #$12 ← (v[i]<0)
              bne $12, $0, for      #if ($12) then for
              muli $13, $12, 4
              add $13, $13, $6      #calcolo indirizzo di w[j]
              lw $10, 0($13)        #
              addi $10, $10, 1      #w[j]++
              sw $10, 0($13)        #
              j for
exit:        lw $8, 0($29)          #ripristino contesto
              lw $9, 4($29)
              lw $10, 8($29)
              lw $11, 12($29)
              lw $12, 16($29)
              lw $13, 20($29)
              addi $29, $29, 24
              jr $31               #ritorno al chiamante
```

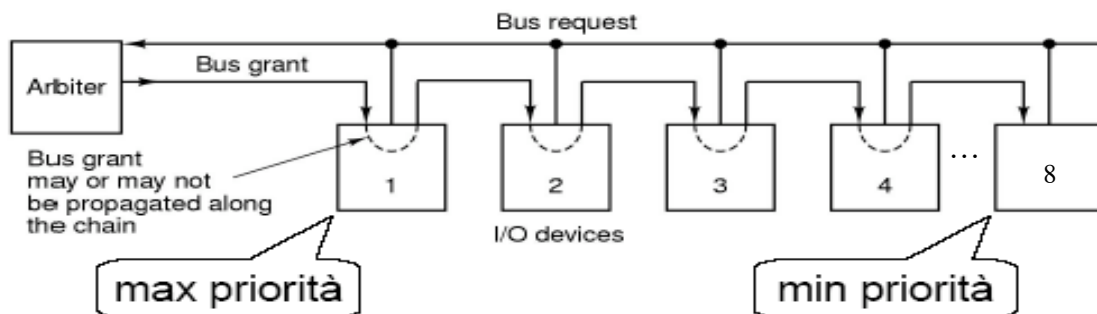
#### ESERCIZIO 4 (7 punti)

Si consideri il caso di un Bus di I/O sul quale devono essere collegate 8 periferiche esterne.

1. (4 punti) Si ipotizzi che sulla parte di controllo del Bus si abbiano solo 2 segnali di controllo liberi da poter utilizzare per gestire l'arbitraggio delle 8 periferiche da collegare. Descrivere la tecnica di arbitraggio utilizzabile in una tale situazione, disegnando un possibile schema di collegamento delle periferiche e spiegando chiaramente le implicazioni del suo utilizzo sulla gestione delle richieste di I/O delle periferiche.
2. (3 punti) Nelle ipotesi di cui al punto precedente, sarebbe utilizzabile la tecnica di arbitraggio centralizzato con richieste indipendenti per ciascuna periferica? In caso di risposta negativa, cosa sarebbe necessario modificare per renderla utilizzabile?

#### Soluzione.

1) Con soli 2 segnali liberi è possibile solo utilizzare la tecnica "daisy chain" che ha solo 2 segnali (request e grant). Ovviamente questa tecnica implica una gestione rigida delle priorità, che sono definite una volta per tutte.



2) Non è ovviamente possibile. Bisognerebbe avere almeno  $8 \times 2 = 16$  segnali di controllo liberi.