

**SOLUZIONI DELLA PROVA SCRITTA DEL CORSO DI**  
**CALCOLATORI ELETTRONICI**  
**NUOVO ORDINAMENTO DIDATTICO**

28 Giugno 2002

**MOTIVARE IN MANIERA CHIARA LE SOLUZIONI PROPOSTE A CIASCUNO DEGLI ESERCIZI SVOLTI**

**ESERCIZIO 1** (7 punti)

Progettare una ALU che realizzi le seguenti funzionalità (l'apice indica la negazione):

		Funzione	
S1	S0	Cin=0	Cin=1
0	0	A+B	A-B'
0	1	A'	-A
1	0	B	B+1
1	1	A	A+1

A e B sono i due operandi a N bit.

- Realizzare la ALU utilizzando un parallel adder e un'opportuna rete logica, illustrando graficamente lo schema progettuale (4 punti).
- Realizzare la ALU utilizzando un MUX 4-1 in luogo delle reti logiche del punto precedente (3 punti).

**Soluzione.**

- L'esercizio si risolve semplicemente implementando le reti logiche che elaborano ogni bit degli operandi prima di passarli al parallel adder. Tali reti logiche si desumono per ispezione visiva della tabella di verità nel testo dell'esercizio:

		S <sub>1</sub> S <sub>0</sub>			
A <sub>i</sub>		00	01	11	10
	0		1		
	1	1		1	

$$A_i^{new} = A_i \overline{S_1} \cdot \overline{S_0} + \overline{A_i} \cdot \overline{S_1} S_0 + A_i S_1 S_0$$

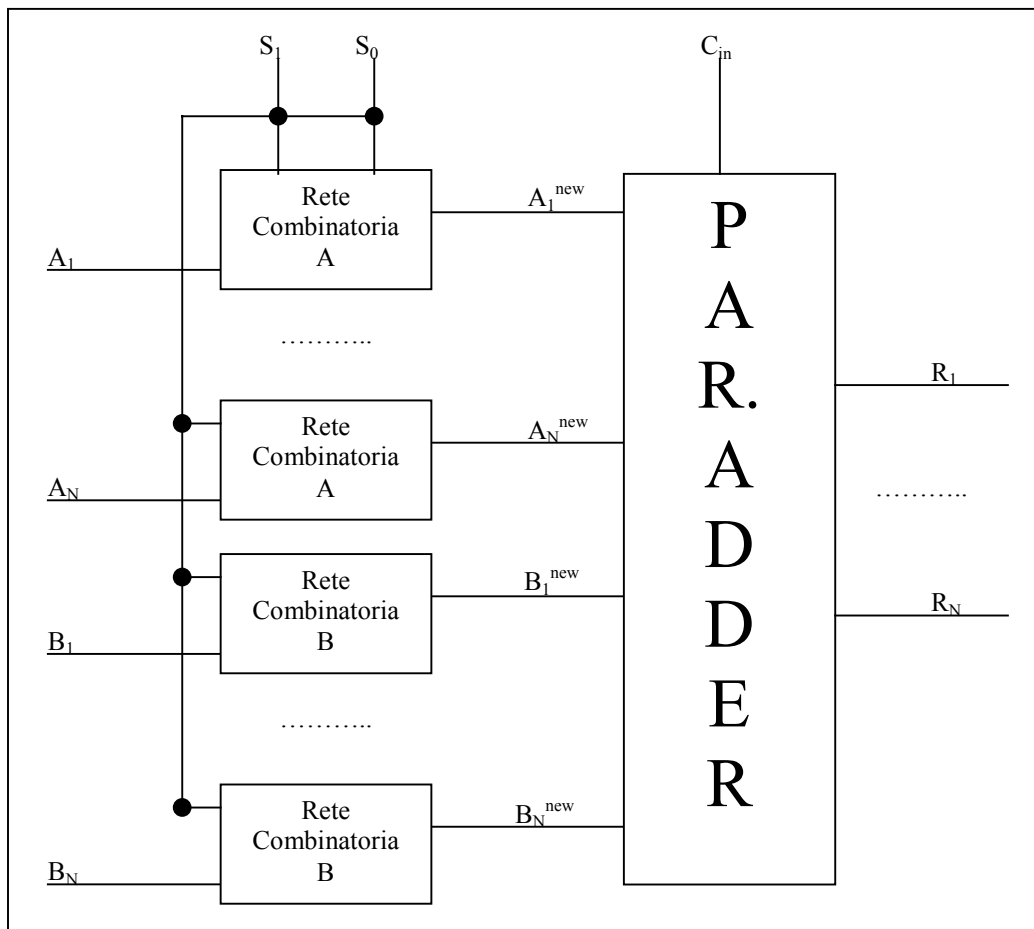
(Rete combinatoria A)

		S <sub>1</sub> S <sub>0</sub>			
A <sub>i</sub>		00	01	11	10
	0				
	1	1			1

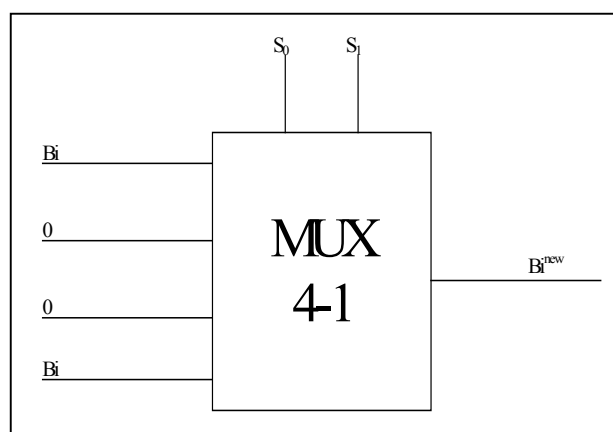
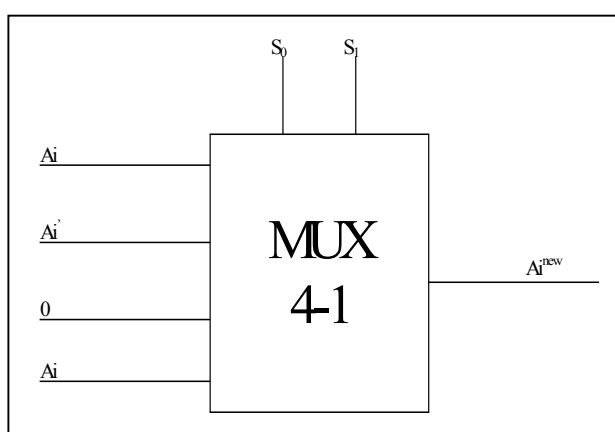
$$B_i^{new} = B_i \overline{S_0}$$

(Rete combinatoria B)

Lo schema progettuale è il seguente:



b) I due MUX 4-1 devono essere così pilotati:



**ESERCIZIO 2** (8 punti)

Si consideri il seguente formato per la rappresentazione binaria dei numeri in virgola mobile: 24 bit, con esponente a 8 bit in eccesso **125**, mantissa frazionaria e normalizzata in segno e valore (1.M).

1. Si calcoli il minimo e il massimo numero **positivo** rappresentabili, escluso lo zero. (2 punti)
2. Si rappresentino nel formato dato i numeri 130.25 e 120.75. (3 punti)
3. Si sommino i due numeri seguendo i passi usati nell'algoritmo dei calcolatori. (3 punti)

**Soluzione.**

1. Minimo numero:  $2^{-125}$ .  
Massimo numero:  $(2-2^{-15}) * 2^{130}$ .

2.

$$130.25 = 10000010.01 = 1.000001001 * 2^7.$$

$$120.75 = 1111000.11 = 1.11100011 * 2^6.$$

$$\text{In eccesso 125: } 7 \rightarrow 111 + 1111101 = 10000100$$

$$6 \rightarrow 110 + 1111101 = 10000011$$

	S	Esponente	Mantissa
130.25	0	10000100	0000010010000000
120.75	0	10000011	1111000110000000

3. Seguendo l'algoritmo dei calcolatori, la somma si realizza coi seguenti passi:

- a) **confronto degli esponenti e allineamento.** L'esponente del primo numero è superiore a quello del secondo, per cui il secondo numero viene portato allo stesso esponente del primo. Per far ciò la mantissa scorre verso destra di una posizione, venendo così denormalizzata:  $1.11100011 \rightarrow 0.111100011$
- b) **somma delle mantisse.** Ora è possibile sommare le mantisse:

$$\begin{array}{r} 1.000001001 + \\ 0.111100011 = \\ \hline 1.111101100 \end{array}$$

- c) **eventuale normalizzazione del risultato.** Non v'è alcun bisogno di normalizzare il risultato, essendo il riporto dopo il primo bit della parte intera nullo, per cui il risultato si rappresenta con lo stesso esponente e con la parte frazionaria della mantissa:

	S	Esponente	Mantissa
251.00	0	10000100	1111011000000000

**ESERCIZIO 3** (8 punti)

Si consideri una gerarchia di memoria a tre livelli: cache, primaria, disco. Dopo alcune prove, vengono effettuate le seguenti misurazioni:

- numero totale di accessi alla memoria: 50
- numero di hit in cache: 45
- numero di hit in primaria: 49
- tempo medio di accesso in cache: 4 ns
- tempo medio di accesso in primaria: 100 ns
- tempo medio di accesso a disco: 50 ms

- a) Qual è lo scopo di un'architettura gerarchica di memorie? Come funziona? (2 punti)
- b) Scrivere e spiegare la formula per il calcolo del tempo medio di accesso alla gerarchia. (4 punti)
- c) Calcolare il tempo medio di accesso alla gerarchia. (2 punti)

**Soluzione.**

- a) Lo scopo di un'architettura gerarchica di memorie è quello di far intendere all'utilizzatore di disporre di una memoria di grandi dimensioni e veloce. Tale concetto è noto come "virtualizzazione della memoria". Per questo motivo il livello più basso della gerarchia è costituito dalla memoria più piccola e veloce (la cache), e a salire di livello troviamo memorie di dimensioni superiori ma più lente (il disco). Naturalmente, per massimizzare le prestazioni di una simile architettura è molto importante massimizzare il numero di successi al primo livello della gerarchia.

Il funzionamento segue il concetto che si vuole realizzare: quando viene richiesto un accesso alla memoria, il dato viene cercato nella memoria del livello più basso della gerarchia. Ciò richiede un tempo pari al tempo medio di accesso a tale memoria, nel nostro caso la cache. Se non viene trovato nessun dato in cache, si passa al livello successivo, che richiede un ulteriore tempo pari al tempo di accesso alla primaria. Infine, se il dato non viene trovato nemmeno in primaria, si passa alla memoria disco che richiede un tempo aggiuntivo pari appunto al tempo medio di accesso al disco.

- b) In accordo con quanto espresso al punto precedente, la formula per il calcolo del tempo medio di accesso alla gerarchia è la seguente:

$$\bar{T} = H_C T_C + (H_P - H_C)(T_P + T_C) + (H_D - H_P)(T_D + T_P + T_C)$$

$T_C, T_P, T_D$ , sono i tempi medi di accesso a cache, primaria e disco.

$H_C, H_P, H_D$ , sono gli *hit ratio* di cache, primaria e disco. Di solito  $H_D = 1$ , in quanto si assume che tutti i dati siano contenuti nella memoria del livello più alto della gerarchia.

$H_i$  ( $i=C,P,D$ ) è la probabilità di trovare un dato nella memoria di livello  $i$  della gerarchia.

La differenza  $(H_P - H_C)$  è la probabilità di trovare un dato in primaria e di non trovarlo in cache. Stesso significato, associato al livello superiore della gerarchia, ha la differenza  $(H_D - H_P)$ .

- c) Gli hit ratio di primaria e di cache si calcolano immediatamente dai dati forniti:  $H_C = 45/50$ ;  $H_P = 49/50$ .

Sostituendo questi e gli altri valori nella formula indicata al punto precedente, si ottiene:

$$\bar{T} = \frac{45}{50} 4 + \frac{4}{50} 104 + \frac{1}{50} 50000104 = 1000014 \text{ ns}.$$

**ESERCIZIO 4** (10 punti)

Si consideri il seguente insieme di processi.

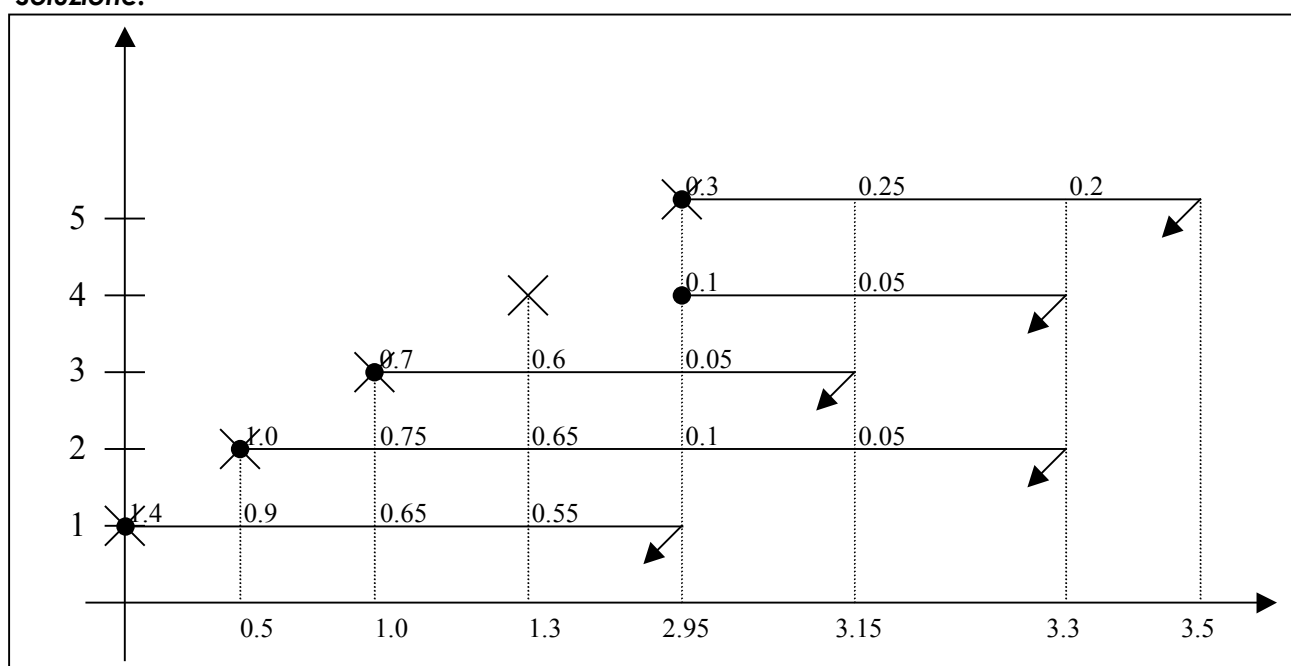
Processo	Tempo di arrivo	Tempo di CPU richiesto	Memoria richiesta
1	0.00	1.40	50K
2	0.50	1.00	30K
3	1.00	0.70	20K
4	1.30	0.10	10K
5	2.95	0.30	20K

La memoria disponibile, di 100K, è gestita dal sistema operativo mediante partizionamento dinamico. All'istante iniziale, si consideri la memoria vuota.

I processi sono gestiti con strategia FIFO multiprogrammata "round robin".

- Mostrare col metodo grafico lo sviluppo di ciascun processo e lo stato della memoria (7 punti).
- Definire e calcolare il tempo di turnaround medio e di turnaround pesato medio (3 punti).

**Soluzione.**

**Stato della memoria.**

Istante 0.0: Crea partizione da 50K per il processo 1. Avvio del processo 1.

Istante 0.5: Crea partizione da 30K per il processo 2. Avvio del processo 2.

Istante 1.0: Crea partizione da 20K per il processo 3. Avvio del processo 3.

Istante 1.3: Non c'è spazio in memoria per allocare il processo 4, che viene posto in stato di attesa.

Istante 2.95: Il processo 1 ha termine, si alloca una partizione da 10K per il processo 4 che viene avviato. Arriva il processo 5 per cui si alloca una partizione da 20K. Avvio del processo 5.

Da questo istante i processi vengono gestiti regolarmente con FIFO multiprogrammata "round robin" fino al termine di tutti i processi.

Processo	Arrivo	Avvio	Termine	Turnaround	W.Tourn.
1	0.00	0.00	2.95	2.95	2.11
2	0.50	0.50	3.30	2.80	2.80
3	1.00	1.00	3.15	2.15	3.07
4	1.30	2.95	3.30	2.00	20.00
5	2.95	2.95	3.50	0.55	1.83
<b>Media</b>				2.09	5.96