

SOLUZIONI DELLA SECONDA PROVA INTERMEDIA DEL CORSO DI CALCOLATORI ELETTRONICI NUOVO ORDINAMENTO DIDATTICO

8 Giugno 2005

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

ESERCIZIO 1 (7 punti)

Progettare una ALU che esegua le seguenti operazioni su due operandi A e B ad n bit (l'apice indica la negazione):

s1	s0	F
0	0	A+B
0	1	A-B
1	0	B-A
1	1	A'+B'

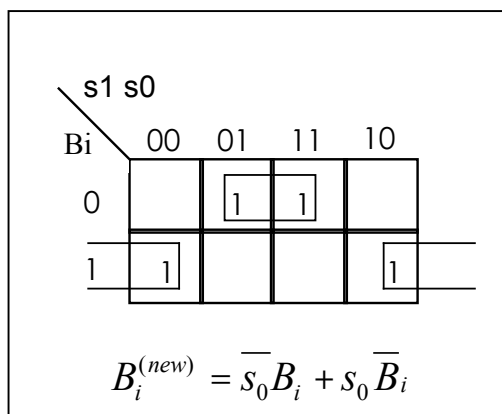
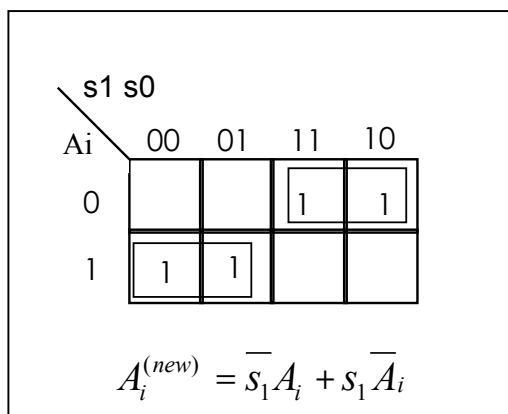
dove **s1** ed **s0** sono due variabili di controllo.

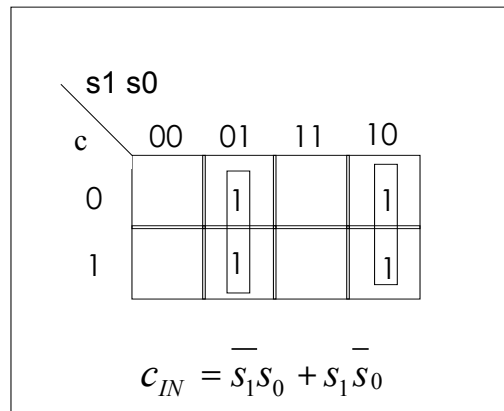
- (4 punti) Si utilizzi un parallel adder ad n bit e una opportuna rete logica. Si tenga conto anche della rete relativa al bit di riporto in ingresso al parallel adder.
- (3 punti) Si sostituiscano le reti logiche del passo precedente con opportuni multiplexer.

Soluzione.

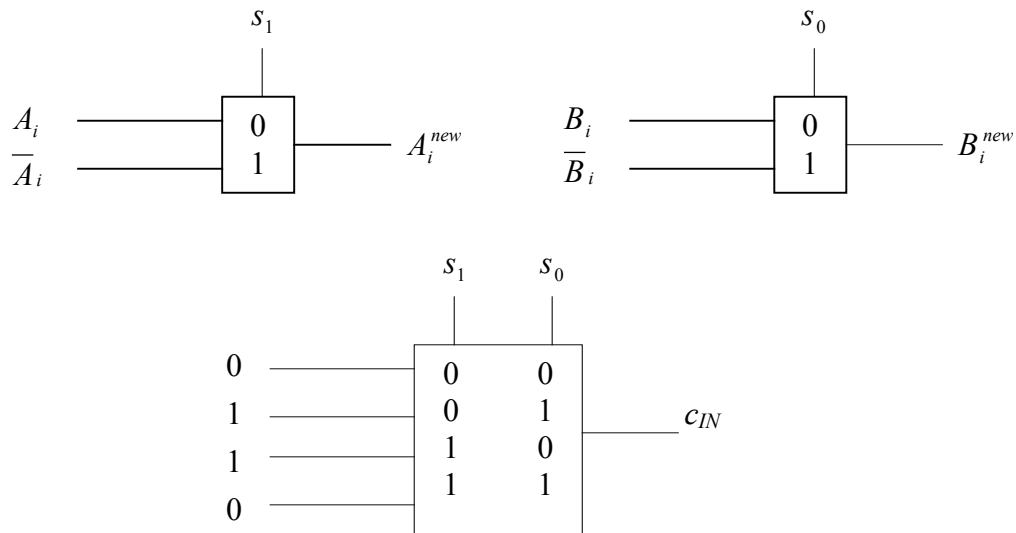
Punto a).

Una semplice ispezione visiva della tabella ci conduce immediatamente alle mappe di Karnaugh relative alle reti logiche a monte del parallel adder. Indicando con A_i e B_i l' i -esimo bit del primo e del secondo addendo, si calcolano le forme minime per l'espressioni degli addendi in ingresso al parallel adder.





Punto b). In questo caso è sufficiente connettere gli addendi nella forma voluta (diretta o negata, oppure in forma costante), agli ingressi di ciascun multiplexer, lasciando il compito di selezionare quelli opportuni alle due variabili di controllo. Nel caso dei bit degli operandi, sono sufficienti MUX 2-1, mentre per il bit di riporto è necessario un MUX 4-1.



ESERCIZIO 2 (9 punti)

Considerato un campo di 28 bit, sia dato il seguente formato: rappresentazione in virgola mobile con mantissa frazionaria e normalizzata in segno e valore (1.M) ed esponente a 8 bit in eccesso 128 (bit di segno a zero per i numeri positivi).

- (1 punto) Quanti sono i valori rappresentabili?
- (3 punti) Calcolare il minimo e il massimo valore rappresentabile in valore assoluto, escluso lo zero.
- (3 punti) Sommare i numeri $(211.5)_{10}$ e $(120.75)_{10}$, esprimendoli in virgola mobile secondo la rappresentazione data, con l'algoritmo dei calcolatori, ed indicando i vari passi dell'algoritmo.
- (2 punti) Proporre un modo per ampliare l'intervallo dei valori rappresentabili a parità di campo disponibile, e discutere vantaggi e svantaggi della soluzione indicata.

Soluzione

- Naturalmente sono 2^{28} .
- Minimo: 2^{-128} Max: $2^{127}(2-2^{-19}) \rightarrow$ va escluso il bit di segno dal conteggio!
- $(211.5)_{10} = 11010011.1 = 1.10100111 * 2^7$
 $(120.75)_{10} = 1111000.11 = 1.11100011 * 2^6$

I due numeri si possono rappresentare nel seguente modo:

Segno	Esponente	Mantissa
0	10000111	1010011100000000000
0	10000110	1110001100000000000

Poiché il primo ha esponente maggiore del secondo ($7 > 6$) di quest'ultimo si fa scorrere la mantissa a destra di una posizione.

I due numeri da sommare sono:

$$\begin{array}{r} 1.101001110 + \\ 0.111100011 = \\ \hline 10.100110001 \quad (*2^7) \end{array}$$

E' necessario normalizzare il risultato

Segno	Esponente	Mantissa
0	10001000	0100110001000000000

- Si può incrementare i bit dell'esponente a scapito della mantissa. In questo modo siamo in grado di ampliare l'intervallo di valori rappresentabile, ma riduciamo la precisione, con legge esponenziale all'allontanarsi dallo zero, con la quale rappresentiamo i numeri.

ESERCIZIO 3 (8 punti)

Scrivere una funzione Assembler MIPS che, ricevendo in ingresso gli indirizzi iniziali di tre vettori v , w , z (allocati rispettivamente nei registri $\$4$, $\$5$, $\$6$) di dimensione N (allocata nel registro $\$7$), scriva nella posizione i -esima di z la media aritmetica dei corrispondenti componenti di v e w . In altri termini $z_i = (v_i + w_i)/2$.

Si assuma, nello scrivere il codice della funzione, di disporre di una funzione `dividi(x,y)` che, ricevendo in ingresso due numeri x e y , allocati rispettivamente in $\$4$ e $\$5$, scriva in $\$6$ il risultato della divisione x/y .

Ad esempio, il codice MIPS potrebbe essere la traduzione del seguente codice C:

```
void media (int v[], int w[], int z[], int N)
{
    int i;
    for (i=0; i<N; i++)
    {
        z[i]=v[i]+w[i];
        z[i]=dividi(z[i],2);
    }
}
```

Vincolo: i contenuti dei registri $\$4$, $\$5$, $\$6$, ovvero gli indirizzi iniziali dei tre vettori, NON devono andare perduti durante l'esecuzione del codice e devono mantenersi tali all'uscita della funzione.

Soluzione.

$\$8 \leftarrow i; \$9 \leftarrow i*4$

$\$10, \$11, \$12 \leftarrow$ copie degli indirizzi iniziali di v, w, z

$\$13 \leftarrow v[i]$

$\$14 \leftarrow w[i]$

$\$15, \$16, \$17 \leftarrow$ indirizzi di $v[i], w[i], z[i]$

<pre>media: addi \$29, \$29, -44 sw \$8, 0(\$29) sw \$9, 4(\$29) sw \$10, 8(\$29) sw \$11, 12(\$29) sw \$12, 16(\$29) sw \$13, 20(\$29) sw \$14, 24(\$29) sw \$15, 28(\$29) sw \$16, 32(\$29) sw \$17, 36(\$29) sw \$31, 40(\$29) move \$10, \$4 move \$11, \$5 move \$12, \$6 addi \$5, \$0, 2 move \$8, \$0 for: beq \$8, \$7, exit muli \$9, \$8, 4 add \$15, \$10, \$9 add \$16, \$11, \$9 add \$17, \$12, \$9 lw \$13, 0(\$15) lw \$14, 0(\$16) add \$4, \$13, \$14 jal dividi sw \$6, 0(\$17) addi \$8, \$8, 1</pre>	<pre>exit: j for move \$4, \$10 move \$5, \$11 move \$6, \$12 lw \$8, 0(\$29) lw \$9, 4(\$29) lw \$10, 8(\$29) lw \$11, 12(\$29) lw \$12, 16(\$29) lw \$13, 20(\$29) lw \$14, 24(\$29) lw \$15, 28(\$29) lw \$16, 32(\$29) lw \$17, 36(\$29) lw \$31, 40(\$29) addi \$29, \$29, 44 jr \$31</pre>
--	--

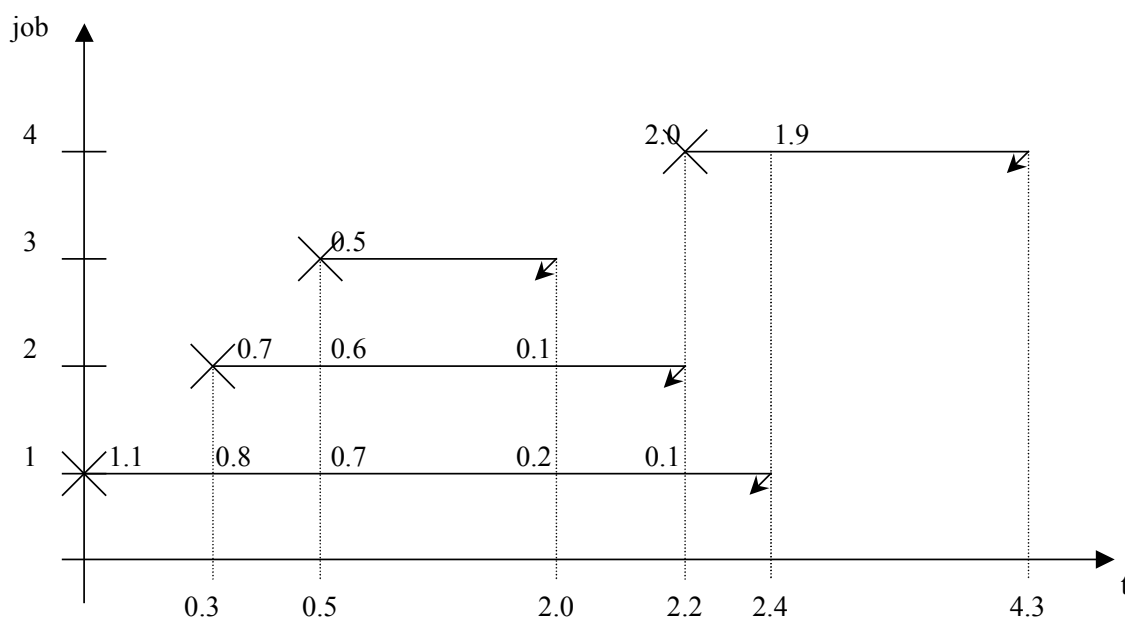
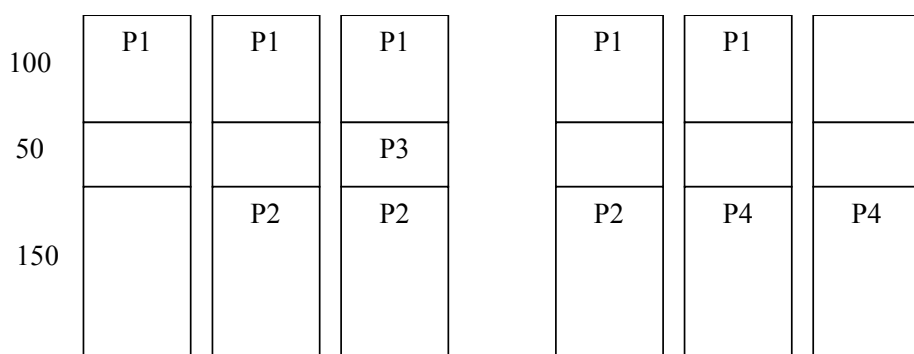
ESERCIZIO 4 (9 punti)

Sia data la seguente lista di processi (si supponga che l'istante iniziale sia 0):

Job	Tempo di Arrivo	Tempo di CPU richiesto	Memoria
1	0.0	1.1	100KB
2	0.3	0.7	70KB
3	0.5	0.5	40KB
4	2.0	2.0	110KB

La memoria disponibile è suddivisa in tre partizioni statiche da 100KB, 50KB, 150KB in questo ordine, con strategia di allocazione Best Fit.

- (6 punti) Mostrare la sequenza di esecuzione dei job usando un grafico (tempo,job), precisando nei vari istanti di tempo il contenuto intermedio della memoria.
- (3 punti) Calcolare il tempo di *turnaround* medio e il tempo di *turnaround pesato* medio, qualora si impieghi la politica di scheduling FIFO multiprogrammata.



Job	Arrivo	Start	Finish	CPU time	Turnaround	W.turnaround
1	0.0	0.0	2.4	1.1	2.4	2.2
2	0.3	0.3	2.2	0.7	1.9	2.7
3	0.5	0.5	2.0	0.5	1.5	3.0
4	2.0	2.2	4.3	2.0	2.3	1.2
				Average	2.0	2.3