Soluzioni degli esercizi del Capitolo 6

Questo documento contiene le soluzioni ad un numero selezionato di esercizi del Capitolo 6 del libro "Calcolatori Elettronici - Architettura e organizzazione", Mc-Graw Hill 2017.
Coloro che avessero sviluppato soluzioni alternative a quelle qui proposte, o soluzioni a esercizi non compresi tra quelli qui trattati, sono invitati a trasmetterle all'indirizzo sotto riportato. Serviranno a migliorare e tenere aggiornati i contenuti di questo sito.
L'autore sarà grato nei confronti di coloro che segnaleranno errori di qualunque genere, sia nella parte che segue sia nel libro menzionato.
giacomo.bucci@unifi.it

Aggiornato il 19 aprile 2017

6.6 L'esercizio può essere svolto in modo seguente.

Si indichi con t_c il tempo di scrittura (accesso) alla cache, con t_w il tempo di scrittura (di una parola) in memoria centrale e con t_l il tempo richiesto per scrivere una linea dalla cache nella memoria centrale (quando la linea modificata deve essere sostituita).
Sia W il numero di parole per linea. Sia b $(0 < b < 1)$ un coefficiente che esprime il guadagno nello scrivere una linea (in modo *burst*) rispetto a W scritture distinte in memoria, tale che si possa scrivere $t_l = Wt_wb$.
Si indichi con N il numero di accessi alla cache nell'intervallo di tempo che va dal caricamento di una linea alla sua sostituzione e con $s \leq 1$ il rapporto tra numero di scritture e numero di accessi nel medesimo intervallo; indicando con N_s il numero di scritture, risulta $N_s = sN$.

Con la politica write-through il tempo complessivo speso per gli N accessi è:

$$(1 - s)Nt_c + sNt_w$$

mentre con la politica write-back il tempo complessivo speso per gli N accessi è:

$$Nt_c + t_l$$

Conseguentemente la politica write-through è preferibile alla politica write-back quando:

$$(1 - s)Nt_c + sNt_w < Nt_c + t_l$$

ovvero

$$-sNt_c + sNt_w < t_l$$

$$sN(t_w - t_c) < t_l$$

che esprime il fatto ovvio secondo cui il "tempo in più" dovuto alla scrittura in memoria di $N_s = sN$ parole nel caso del write-through deve essere inferiore al tempo di scrittura dell'intera linea in write-back. Se ora si sostituisce t_l, si ha:

$$sN(t_w - t_c) < Wt_wb$$

da cui:

$$sN(1 - \frac{t_c}{t_w}) < Wb$$

Essa ci dice che per un dato t_c, t_w e N, il write-through trova vantaggio, com'è ovvio, da un basso s.

Vale la pena di esaminare l'influenza della dimensione di linea (W). Apparentemente il termine di sinistra è indipendente da W. Si tratta evidentemente di un controsenso: la dimensione della linea ha influenza sul tasso di miss, come pure sul numero di riferimenti che una linea riceve in cache. In altre parole il prodotto sN è pure funzione di W. Si faccia l'ipotesi che il prodotto sN cresca linearmente con W. Se ora si considera il termine di destra, si deve notare che il coefficiente b non è costante, in quanto il vantaggio del trasferimento burst è tanto maggiore quanto maggiore è W. In altre parole il prodotto Wb cresce meno che linearmente. Dunque, con la precedente ipotesi su sN, le linee grandi tendono a favorire la politica write-back (il termine a destra risulta relativamente più piccolo al crescere di W, quindi la condizione risulta più difficile da soddisfare).

6.7 Sono dati:
$M = 2^{26}$ byte (dimensione massima della memoria); $v = 8 = 2^3$ vie; $C = 256\,\text{KB} = 2^{18}$ byte (dimensione complessiva della cache). Ogni linea è di $64 = 2^6$ byte; le parole sono di 4 byte, per cui si hanno 16 parole a linea. La capacità di una via è $C_v = C/v = 2^{18}/2^3 = 2^{15}$ byte. Da cui deriva che ogni via contiene $L = 2^{15}/2^6 = 2^9$ linee. Usando i simboli del testo si ha dunque: $w = 4$; $l = 9$. Nella determinazione del campo del dell'indirizzo di blocco si deve tener conto che la memoria è limitata a 2^{26} byte e che quindi le linee di indirizzo di interesse vanno da A_{25} a A_2 (A_1 e A_0 indirizzano il byte entro la parola). Ne consegue $b = 11$, come indicato in Figura 6.1. Il numero di bit di Tag per ciascuna via è dato dal prodotto $b \times L = 11 \times 2^9$. In totale si hanno dunque $11 \times 2^9 \times 2^3 = 44$ Kbit (si ricordi che $1\text{K} = 2^{10} = 1024$).

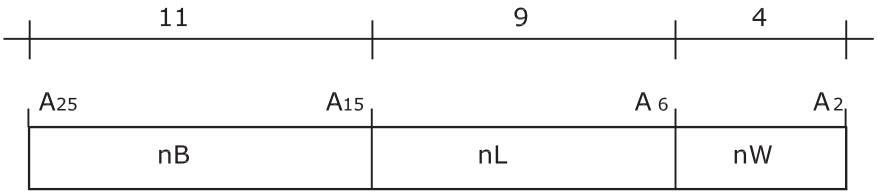

Figura 6.1 Struttura degli indirizzi per l'Esercizio 6.7.

6.8 Si deve assumere che all'inizio la cache sia vuota.

Nel caso (a) si numerano le sedici linee da 0 a 15. I primi 4 riferimenti danno miss; il quinto riferimento (20) va nella linea 4 di cache, che contiene il dato all'indirizzo 4 di memoria centrale: si ha dunque un miss. E' facile verificare che i successivi riferimenti fino al 43 incluso determinano miss. Il riferimento 5 dà hit, seguono un miss e due hit. In conclusione si ha questa sequenza:

1	4	8	5	20	17	19	56	9	11	4	43	5	6	9	17
m	m	m	m	m	m	m	m	m	m	m	m	h	m	h	h

Il contenuto finale della cache è quello che segue (non si riportano le linee non indirizzate)

Linea:	1	3	4	5	6	8	9	11
Contenuto:	17	19	4	5	6	56	9	43

Nel caso (b) si hanno 4 linee di cache, su ciascuna delle quali si mappano 4 linee di memoria. Ad esempio: sulla linea 0 di cache si mappano le due linee di memoria corrispondenti a questi due gruppi di parole {0,1,2,3}, {16,17,18,19}. I primi tre riferimenti danno miss, mentre il quarto (5) trova la linea in cache e perciò determina hit. Seguono poi due miss e quindi un hit sul 19. Gli altri hit si hanno con 11, 5, 6 e 17. Si ha dunque questa sequenza:

1	4	8	5	20	17	19	56	9	11	4	43	5	6	9	17
m	m	m	h	m	m	h	m	m	h	m	m	h	h	m	h

Contenuto finale della cache:

Linea:	0	1	2	3
Contenuto:	{16,17,18,19}	{4,5,6,7}	{8,9,10,11}	-

Nel caso (c) le due vie sono di 8 linee di una parola. Il primo hit si ha sul secondo 4. Complessivamente si ha questa sequenza:

1	4	8	5	20	17	19	56	9	11	4	43	5	6	9	17
m	m	m	m	m	m	m	m	m	m	h	m	h	m	h	h

Contenuto finale della cache:

Via 0	Linea:	0	1	2	3	4	5	6	7
	Contenuto:	8	9	-	43	4	5	6	-
Via 1	Linea:	0	1	2	3	4	5	6	7
	Contenuto:	56	17	-	11	20	-	-	-

Nel caso (d) ogni via contiene 2 linee di 4 byte. La sequenza di miss/hit è questa:

1	4	8	5	20	17	19	56	9	11	4	43	5	6	9	17
m	m	m	h	m	m	h	m	m	h	h	m	h	h	h	m

Contenuto finale della cache:

Via 0	Linea:	0	1
	Contenuto:	{8,9,10,11}	{4,5,6,7}
Via 1	Linea:	0	1
	Contenuto:	{16,17,18,19}	{20,21,22,23}

6.9 Nel caso a) ogni via è di 8 parole e di 4 linee. I primi tre riferimenti danno miss e vanno a occupare posizioni nella via 0; il quarto e quinto riferimento danno ancora miss e vanno a occupare posizioni nella via 1; il sesto dà miss e occupa l'ultima linea di via 0; il riferimento successivo (19) dà miss sulla via 1; l'ottavo riferimento (48) dà miss e occupa la prima posizione nella via 3; il successivo (9) dà hit sulla via 0. Al riferimento 34 (che deve andare nella riga 1) si ha miss e tutte le posizioni sono occupate; l'algoritmo LRU sostituisce la linea di via 1 in quanto è su questa via che si è avuto l'ultimo riferimento (i contatori LRU per la riga indicano 0,3,2,1). Il riferimento 12 è ancora un miss; ad esso spetta un posto in riga 2, i cui contatori sono 1,3,2,0; viene quindi sostituita la linea di via 1. Con analogo ragionamento si trova che il riferimento 28 prende posto nella via 2 (linea 2).

2	4	8	16	20	6	19	48	9	42	11	4	43	44	2	5	33	34	36	6	12	14	15	28	9
m	m	m	m	m	m	m	m	h	m	m	h	h	m	h	h	m	m	m	h	m	m	h	m	h

Contenuto finale della cache:

Via 0	Linea:	0	1	2	3
	Contenuto:	{8,9}	{2,3}	{4,5}	{6,7}
Via 1	Linea:	0	1	2	3
	Contenuto:	{16,17}	{34,35}	{12,13}	{14,15}
Via 2	Linea:	0	1	2	3
	Contenuto:	{48,49}	{42,43}	{28,29}	-
Via 3	Linea:	0	1	2	3
	Contenuto:	{32,33}	{10,11}	{36,37}	-

Nel caso b) ci sono 2 linee per via. La sequenza di miss/hit è questa:

2	4	8	16	20	6	19	48	9	42	11	4	43	44	2	5	33	34	36	6	12	14	15	28	9
m	m	m	m	m	h	h	m	h	m	h	h	h	m	m	h	m	h	m	h	m	h	h	m	h

Contenuto finale della cache:

Via 0	Linea:	0	1
	Contenuto:	{40,42,42,43}	{4,5,6,7}
Via 1	Linea:	0	1
	Contenuto:	{8,9,10,11}	{12,13,14,15}
Via 2	Linea:	0	1
	Contenuto:	{0,1,2,3}	{29,29,30,31}
Via 3	Linea:	0	1
	Contenuto:	{32,33,34,35}	{36,37,38,39}

6.10 In Tabella 6.1 si da una rappresentazione dell'evoluzione del contenuto dello stack LRU. Per ogni evento si riporta lo stato che ne consegue. Se la via è non valida si riporta "−". Ovviamente questa è una schematizzazione, in quanto la posizione contiene comunque un numero da 0 a 3, ma al tempo stesso il corrispondente bit di validità indica non valido. Sotto al contenuto dello stack viene riportato il numero di blocco di cui attualmente si tiene traccia (questo numero è il valore del corrispondente TAG).

6.11 Per la sequenza: m, m, m, h_2, h_1, m, m, h_2, m, m, h_2, h_1, h_3, si hanno gli eventi di Tabella 6.2.

Si può verificare che se la condizione di partenza è che tutte le linee sono valide, dopo il sesto evento l'evoluzione diventa identica a quella riportata in Tabella 6.2.

	Evento	S_0	S_1	S_2	S_3
1	invalidazione di tutte le linee	–	–	–	–
	(tag corrispondente)	–	–	–	–
2	riferimento alla linea del blocco 5	0	–	–	–
	(tag corrispondente)	(5)	–	–	–
3	riferimento alla linea del blocco 6	1	0	–	–
	(tag corrispondente)	(6)	(5)	–	–
4	riferimento alla linea del blocco 7	2	1	0	–
	(tag corrispondente)	(7)	(6)	(5)	–
5	riferimento alla linea del blocco 5	0	2	1	–
	(tag corrispondente)	(5)	(7)	(6)	–
6	invalidazione della linea di via 0	–	2	1	–
	(tag corrispondente)	–	(7)	(6)	–
7	riferimento alla linea del blocco 9	0	2	1	–
	(tag corrispondente)	(9)	(7)	(6)	–
8	riferimento alla linea del blocco 5	3	0	2	1
	(tag corrispondente)	(5)	(9)	(7)	(6)
9	riferimento alla linea del blocco 7	2	3	0	1
	(tag corrispondente)	(7)	(5)	(9)	(6)
10	riferimento alla linea del blocco 4	1	2	3	0
	(tag corrispondente)	(4)	(7)	(5)	(9)
11	invalidazione della linea di via 3	1	2	-	0
	(tag corrispondente)	(4)	(7)	(-)	(9)
12	riferimento alla linea del blocco 7	2	1	-	0
	(tag corrispondente)	(7)	(4)	(-)	(9)

Tabella 6.1 Evoluzione dello stack LRU (Esercizio 6.10) a seguito della sequenza di eventi. La seconda riga di ogni evento riporta il tag contenuto alla posizione x nella via corrispondente.

	Evento	B_0	B_1	B_2	V_0	V_1	V_2	V_3
0	Condizione iniziale	0	0	0	0	0	0	0
1	Miss	1	1	0	1	0	0	0
2	Miss	1	0	0	1	1	0	0
3	Miss	0	0	1	1	1	1	0
4	Hit sulla linea di via 2	0	0	1	1	1	1	0
5	Hit sulla linea di via 1	1	0	1	1	1	1	0
6	Miss	0	0	0	1	1	1	1
7	Miss	1	1	0	1	1	1	1
8	Hit sulla linea di via 2	0	1	1	1	1	1	1
9	Miss	1	0	1	1	1	1	1
10	Miss	0	0	0	1	1	1	1
11	Invalidazione di via 2	0	0	0	1	1	0	1
12	Hit sulla linea di via 1	1	0	0	1	1	0	1
13	Hit sulla linea di via 3	0	0	0	1	1	0	1

Tabella 6.2 (Esercizio 6.11) Sequenza di modifica dei B_i e V_i in base alla sequenza di eventi sopra riportata.

6.12 Svolgiamo l'esercizio senza tenere conto della validità delle linee. Essendoci 2 sole vie, basta un flip-flop per ogni posizione I_L. Stabilito che il FF dia 0 se l'ultimo riferimento è stato alla via 0 e 1 se è stato alla via 1, e tenuto conto che hit e miss sono mutuamente esclusivi, usando un JK si può ragionare nel modo seguente.

- in caso di hit sulla via 0 occorre dare un ingresso di reset (J=0, K=1);
- in caso di hit sulla via 1 occorre dare un ingresso di set (J=1, K=0)
- in caso di miss occorre far commutare il flip-flop (J=1, K=1)

Dunque

$$J = h_1 + m \qquad K = h_0 + m$$

Le due espressioni sono relative alla linea indirizzata, pertanto è necessario che solo il contatore della linea in posizione I_L venga effettivamente comandato, lasciando gli altri nello stato in cui si trovano (basta che i corrispondenti J e K siano ambedue a 0). Si ottiene dunque lo schema di Figura 6.2, dove "sel" è asserito solo per la posizione indirizzata, ovvero decodificando il campo I_L con un decodificatore $1/n$.

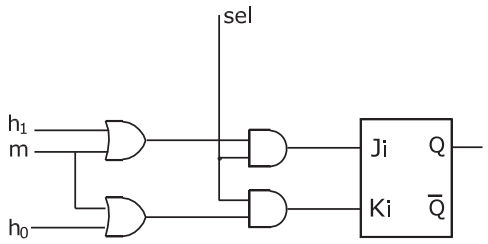

Figura 6.2 Contatore per la generica linea (Esercizio 6.12).

Si faccia attenzione al fatto che la precedente soluzione non tiene conto della condizione di validità delle linee. Si vedano a tale proposito gli esercizi **??** e 6.15.

6.13 Ciascuna delle due vie (parte DATI) è di 128 KB. Essendo le linee di 32 byte, il numero di linee per via è: $L = 2^{17}/2^5 = 2^{12} = 4$ Klinee. La struttura degli indirizzi è illustrata in Figura 6.3.

Nella costruzione della parte DATI si deve tener conto che, essendo le parole di 4 byte, i 128 KB di una via devono essere costruiti con 4 colonne da 32 kbyte, ciascuna delle quali viene selezionata tramite il corrispondente BE. La parte TAG viene costruita con due integrati da 4 KB, come illustrato in Figura 6.4. In Figura 6.5 viene dato il dettaglio dell'indirizzamento di un integrato della parte DATI.

6.14 Anzitutto osserviamo che una linea contiene $16 \times 4 = 64 = 2^6$ byte. Dunque si hanno $512K/64 = 2^{19}/2^6 = 2^{13} = 8K$ linee.

La struttura risultante dell'indirizzo è questa:

$[A_{5-2}]$: numero di parola nella linea.

$[A_{18-6}]$: numero di linea nel blocco.

$[A_{31-19}]$: numero di blocco (TAG).

La parte DATI ha parallelismo 4 byte e quindi ogni corsia di byte richiede $512K/4 = 128K$ byte.

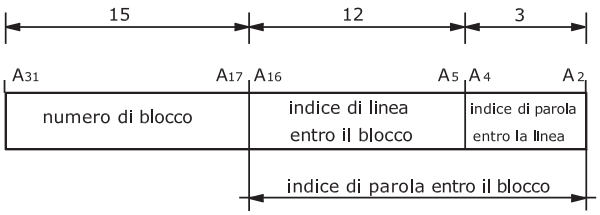

Figura 6.3 Struttura degli indirizzi per l'Esercizio 6.13.

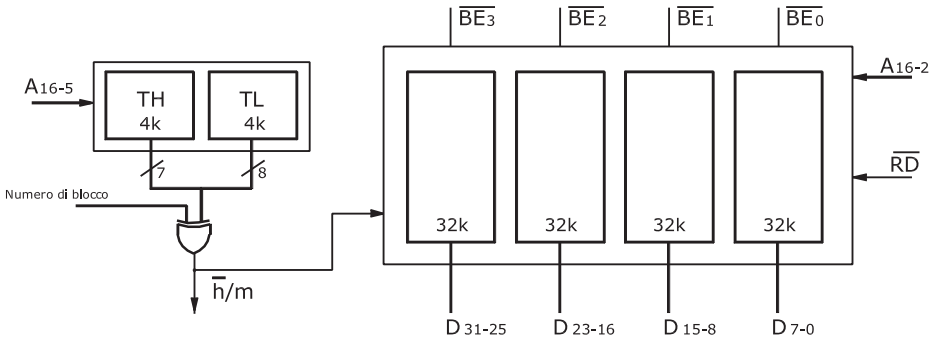

Figura 6.4 Struttura di una via (Esercizio 6.13). La figura mostra solo la parte relativa alla lettura. L'indirizzo di linea individua la posizione nel TAG. Tramite A_{16-2} si si indirizzano direttamente le parole in DATA .

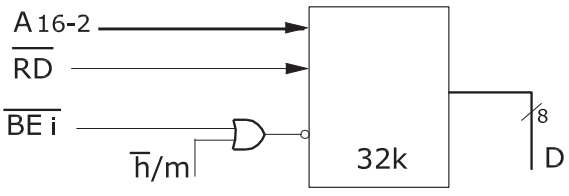

Figura 6.5 Indirizzamento dell'integrato della parte DATI (Esercizio 6.13) selezionato tramite $\overline{\mathrm{BE_i}}$.

La parte TAG ha un parallelismo di 13 bit $(31-19-1)$. Pertanto servono 2 integrati (*byte-wide*) da 8 KB. Assumiamo che i bit TAG_{12-0} venga usata come TAG e che il bit TAG_{15} venga usato come bit di validità, assumendo che 1 indichi linea valida e 0 linea non valida. (I bit TAG_{14-13} risultano inutilizzati.)

La logica di hit è così fatta. Ai piedini degli indirizzi degli integrati che costituiscono il TAG vengono portate le linee $[A_{18-6}]$ del bus degli indirizzi. I Bit TAG_{12-0} letti (occorre asserire il comando READ al TAG anche in presenza di una operazione di scrittura in cache) vengono confrontati con le linee $[A_{31-19}]$. Il risultato del confronto viene messo in AND con il bit TAG_{15}. Se l'uscita dell'AND risulta a 1 si ha hit. Se risulta 0 si ha un miss o perché la linea non è valida o perché il TAG non corrisponde.

Con la precedente soluzione, l'invalidazione della linea i si ottiene scrivendo nella posizione del TAG corrispondente una parola che ha 0 nel bit più significativo.

6.15 La struttura degli indirizzi è identica a quella dell'esercizio 6.13.

Per quanto si riferisce ai contatori LRU, si può osservare che, avendosi due sole vie, un contatore di 1 bit è sufficiente a identificare le via in corrispondenza di ogni posizione di linea. Questo indicatore (realizzabile con un flip-flop opportunamente comandato) rappresenta a tutti gli effetti lo stack LRU . Il numero complessivo di bit per i contatori è dunque pari al numero di linee per via, ovvero 4 k.

Si indichi con:

- c il contatore LRU. relativo alla coppia di linee in posizione i, con la convenzione che $c = 0$ indica che l'ultimo hit alla posizione i è stato sulla via 0, mentre $c = 1$ indica che l'ultimo hit alla medesima posizione è stato sulla via 1;
- v_0 e v_1 i bit di validità della linea in posizione i in cache, per la via 0 e 1 rispettivamente. $v_0 = 0$ (ovvero $v_1 = 0$) indica che la linea alla posizione i sulla via 0 (ovvero sulla via 1) non è valida, mentre $v_0 = 1$ e $v_1 = 1$ indicano che le linee corrispondenti sono valide;
- m la condizione di miss alla posizione i;
- h_0 e h_1 indicano rispettivamente l'avverarsi di un hit sulla via 0 o 1 (alla posizione i).

Si noti che h_0, h_1, v_0 e v_1 sono relativi alla singola linea in posizione i nell'una o l'altra via, c e m sono relativi alla corrispondente posizione i.

Se ora si conviene di realizzare ogni singolo contatore tramite un flip-flop SR, si tratta di progettare la rete di Figura 6.6. La rete fornisce ha in uscita lo stato del flip-flop. Per le ragioni che saranno chiare dal modo in cui si ricavano gli ingressi al flip-flop, conviene usare un SR oppure un JK. Usiamo un SR.

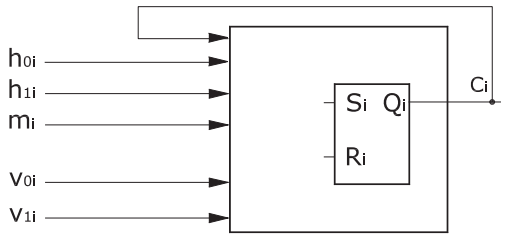

Figura 6.6 (Esercizio 6.15) Schema di principio per il progetto del contatore LRU associato a ciascuna posizione di linea.

Mostriamo come si ricavano le espressioni di R e S in riferimento al reset (R). L'ingresso di reset deve essere asserito se c'è hit sulla via 0, oppure quando, in presenza di

miss, la linea di via 0 non è valida oppure le due linee sono ambedue valide e il contatore è a 1 (indicando che il precedente hit è stato sulla via 1 e quindi deve essere rimpiazzata la linea di via 0.). Si ha dunque:

$$R = h_0 + m(\overline{v_0} + v_0 v_1 c) = h_0 + m(\overline{v_0} + v_1 c) = h_0 + m\overline{v_0} + m v_1 c$$

Ragionando in modo analogo si ottiene per S questa espressione:

$$S = h_1 + m v_0(\overline{v_1} + v_1 \overline{c}) = h_1 + m v_0(\overline{v_1} + \overline{c}) = h_1 + m v_0 \overline{v_1} + m v_0 \overline{c}$$

Dalle precedenti espressioni si ottiene lo schema di Figura 6.7. È immediato verificare che se non c'è né un hit né un miss sulla linea (ovvero $h_0 = 0$, $h_1 = 0$ e $m = 0$), allora sia l'ingresso S sia l'ingresso R sono a zero e dunque il FF non commuta, mantenendo il precedente c.

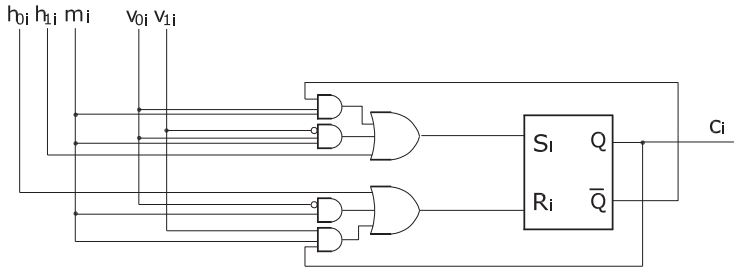

Figura 6.7 (Esercizio 6.15) Schema per il generico contatore modulo 2 associato alla coppia di linee in posizione i.

Con riferimento alla Figura 6.7, seguendo lo stesso ragionamento fatto nella soluzione dell'Esercizio 6.6, l'impiego dei FF SR (o JK) suggerisce che l'ingresso 00, che non determina cambiamenti di stato, potrebbe essere assicurato anche mettendo in AND le uscite dalle due porte OR con una linea asserita solo quando viene indirizzata la posizione i, come in Figura 6.8. Ovviamente la linea in questione, indicata come I in figura, deriva dalla decodifica dell'indirizzo i; dunque in un dato momento solo una delle possibili 2^l linee è asserita (ove x è il numero di bit che dà l'indice di linea). Ne consegue che solo la posizione indirizzata potrà eventualmente avere la coppia RS diversa da zero.

Dunque, h_0, h_1 e m di Figura 6.8 possono essere non gli hit o il miss relativi alla posizione i, bensì gli hit e il miss relativi a tutta la cache. Ovviamente v_0, v_1 e c restano quelli relativi alla posizione i.

6.16 Anzitutto cominciamo con l'osservare che, anche se lo schema della Figura 6.6 è stato tracciato nell'ipotesi che le quattro linee sulle differenti vie siano tutte valide, il meccanismo di gestione dello "LRU stack", consistente nell'inserire a sinistra il numero d'ordine della via riferita più di recente facendo scorrere il contenuto secondo quanto illustrato nel libro, la presenza di una linea o più linee non valide è irrilevante. Essa ha solo questo effetto:

a) in caso di hit, lo stack viene aggiornato facendo scorrere verso destra i contenuti degli S_i di ordine inferiore a quello contenente il numero di via che ha dato hit. Il fatto che una o più posizioni di quelle che scorrono contenga il numero di via marcata (altrove) come non valida è ininfluente;

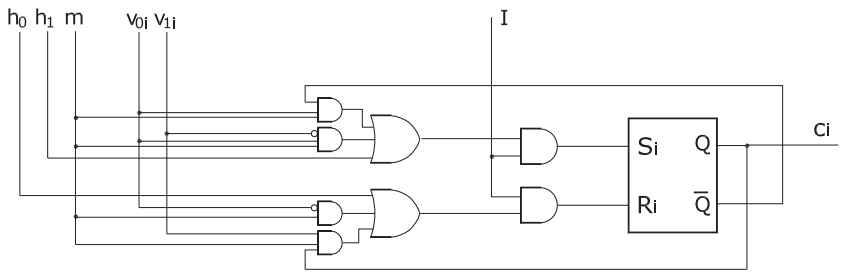

Figura 6.8 (Esercizio 6.15) Modifica della soluzione di Figura 6.7. Prevedendo la porta AND a valle di S e R valutati come sopra, si garantisce che venga attivato il solo contatore relativo alla posizione i. La linea I è l'uscita del decodificatore dell'indice i di linea e, ovviamente, è asserita (essa sola) quando viene indirizzata la posizione i. Il clock viene trasmesso a tutti i FF, ma solo quello in questione può eventualmente cambiare stato. Ciò permette di prendere per h_0, h_1 e m le corrispondenti entità riferite a tutta la cache e non solo alla posizione i (con ciò semplificando enormemente la rete risultante).

b) nel caso di miss, la linea invalida più a sinistra viene selezionata per essere portata in S_0 (come J); la propagazione a quel punto equivale a quella del caso di hit.

Vale la pena di rimarcare che il precedente ragionamento si applica anche durante la fase iniziale di funzionamento della cache. Inizialmente tutte le vie sono invalide. Il primo riferimento determina un miss, a seguito del quale viene presa la via 0, per ipotesi più prioritaria; ne consegue che in S_0 viene messo 0, quello che succede a destra è indeterminato, in quanto è indeterminato il contenuto di S_1–S_3[1]. Il successivo miss determina la scelta della via 1; come risultato si avrà $S_0 = 1$, $S_1 = 0$, S_2 e S_3 indeterminati. Il prossimo miss porterà a $S_0 = 2$, $S_1 = 1$, $S_2 = 0$, S_3 indeterminato. In sintesi, a seguito dei miss, S_0–S_3 assumono necessariamente 4 valori diversi. Ovviamente gli hit che arrivano prima di aver raggiunto questo stato, lasciano invariata la parte a destra dello S_i che viene sostituito.

Trattiamo prima il caso del solo hit e cominciamo col definire i simboli delle entità che ci servono. Con riferimento alla Figura 6.9 si indichi con:

- x la generica linea di uscita dal decodificatore del campo L dell'indirizzo che contiene l'indice della posizione in cache L (ovviamente è sempre asserita una e una sola linea x delle 2^l linee che corrispondono alla decodifica del campo L, essendo l la dimensione del campo);
- h_0, h_1, h_2, h_3 le linee che dicono se c'è un hit sulla corrispondente via; ovviamente, al più potrà essere asserita una sola di tali linee;
- h_{0x}, h_{1x}, h_{2x}, h_{3x} le linee che si ottengono mettendo in AND le precedenti con x; esse indicano rispettivamente che alla posizione data da I c'è stato un hit nella corrispondente via;
- h_x l'OR dei precedenti ovvero la linea che indica che c'è stato un hit alla posizione considerata.

[1] Può essere che una parte scorra e una no.

Lo schema di Figura 6.9 aggiunge a quello di Fig 6.6 del testo la parte di logica che identifica J e che determina il clock allo stack LRU. Si noti che il segnale *hclk* di Figura 6.6 del testo è ottenuto filtrando il clock con la condizione "hit alla posizione x" (cioè h_x).

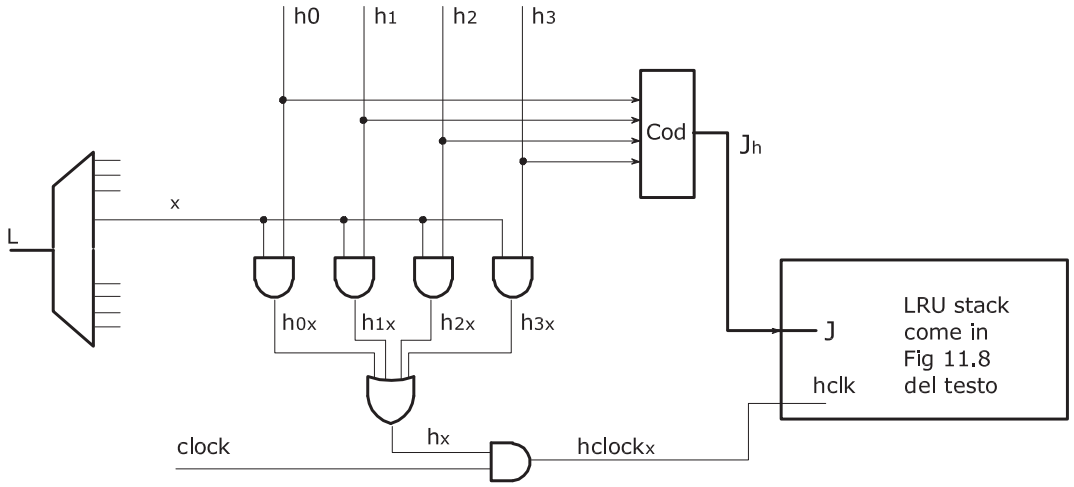

Figura 6.9 (Esercizio 6.16) Ricostruzione completa dello schema di Figura 6.6 del testo, con l'aggiunta della logica che genera J e hclck con riferimento ai soli hit. La parte di Fig 6.6 non viene ripetuta. Si noti che la generazione degli h_{ix} è del tutto superflua e che $hclock_x$ poteva essere generato direttamente mettendo in AND *clock* con l'uscita di un OR avente in ingresso gli h_i.

Per quanto si riferisce al codificatore Cod che genera J_h, indichiamo con J_{1h} e J_{0h} rispettivamente il bit più significativo e il meno significativo. In tabella 6.3 viene data la tabella di verità per J_{1h} e J_{0h} solo per i valori degli ingressi in corrispondenza ai quali l'uscita è definita.

h_0	h_1	h_2	h_3	J_{h1}	J_{h0}
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Tabella 6.3 (Esercizio 6.16) Codifica dei numero di via su cui si ha hit

Tenuto conto delle condizioni di indifferenza (sono molte perché, sulle mappa di Karnaugh di ordine 4 le due uscite sono definite solo in corrispondenza delle 4 caselle ove uno solo degli ingressi è a 1), la copertura ottima delle mappe produce

$$J_{1h} = h_3 + h_2 \qquad\qquad J_{0h} = h_3 + h_1$$

Che esprime l'ovvia conseguenza deducibile dalla Tabella 6.3, ovvero che J_{1h} è asserito quando sono asseriti h_3 o h_2, mentre J_{0h} è asserito quando lo sono h_1 o h_3. Notare che Cod genera comunque un valore in uscita, anche quando tutti gli h_i sono nulli; è facile verificare che con tale configurazione degli ingressi, le due funzioni precedenti producono

$J_{1h} = 1$ e $J_{0h} = 0$. Ma ciò è irrilevante in quanto, essendo $h_x = 0$, $hclock_x$ non viene asserito (dunque lo stack resta immutato).

Consideriamo ora il caso del miss, indipendentemente dall'hit; con riferimento alla Figura 6.10 si indichi con:

- m il miss in cache e con m_x il fatto che il miss sia alla posizione x. Ovviamente m_x si ottiene come AND di m e di x.
- $mclock_x$ l'AND di m_x con il clock.
- v_{0x}, v_{1x}, v_{2x}, v_{3x} gli indicatori di validità delle 4 linee in posizione x.
- S_{0x} e S_{3x} rispettivamente (i contenuti de) le posizioni estreme dello stack.
- J_m il numero che si deve introdurre come S_{0x} in presenza di miss.

In Figura 6.10 il Mux serve a selezionare come ingresso allo stack una delle 5 possibili alternative corrispondenti al numero d'ordine della via più prioritaria tra quelle non valide, ovvero a S_{3x} in caso di assenza di linee non valide. Mux è comandato da 5 selettori il cui significato è evidente. È immediato scrivere le espressioni per i 5 selttori

$Sel_0 = \overline{v}_{0x} m_x$ $Sel_1 = v_{0x}\overline{v}_{1x} m_x$ $Sel_2 = v_{0x}v_{1x}\overline{v}_{2x} m_x$ $Sel_3 = v_{0x}v_{1x}v_{2x}\overline{v}_{3x} m_x$

$SelS_3 = v_{0x}v_{1x}v_{2x}v_{3x} m_x$

Il segnale $mclock_x$ comanda lo LRU stack esattamente come nella Figura 6.6 del testo.

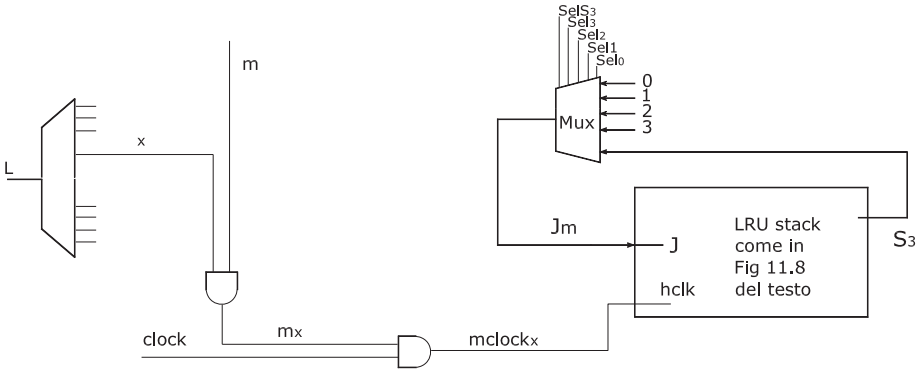

Figura 6.10 (Esercizio 6.16) Ricostruzione completa dello schema di Figura 6.6 del testo, con l'aggiunta della logica che genera J_m e $mclock_x$ con riferimento ai soli miss. La parte di Fig 6.6 non viene ripetuta.

La ricomposizione dei due schemi precedente richiede un ulteriore multiplexer per scegliere tra J_h e J_m, come in Figura 6.11, dove non sono riportate le parti mostrate in precedenza. Il clock per lo stack ($clock_x$) è la combinazione di $hclock_x$ e $mclock_x$. l'ingresso allo stack è selezionato attraverso h_x: se h_x è asserito viene presentato J_h, se è disasserito viene presentato J_m. In ogni caso lo stack viene comandato solo se ($clock_x$) è asserito, ovvero se c'è stato hit o miss riferito alla posizione x.

Conviene ora fare questa considerazione. Nei precedenti ragionamenti sono stati considerati gli hit e i miss con riferimento alla specifica posizione e alla fine è stato applicato $clock_x$ in base al fato che sia stata selezionata la posizione x. Si può osservare che $clock_x$

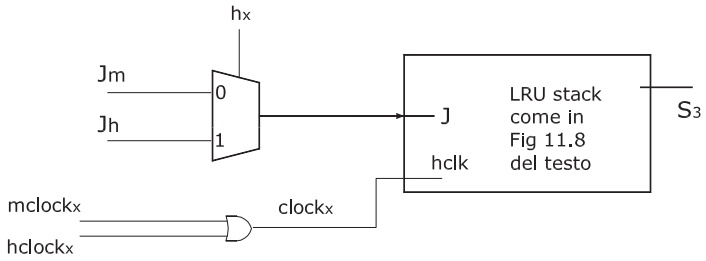

Figura 6.11 (Esercizio 6.16) Combinazione degli schemi di Fig 6.9 e 6.10 per trattare sia hit che miss.

si può ottenere semplicemente filtrando *clock* attraverso x. Ma allora al posto degli hit e del miss riferiti alla specifica posizione x e si può prendere per essi gli hit riferiti all'intera via (h_0, h_1, h_2, h_3) e il miss generale (m). Infatti qualunque cosa produca la rete combinatoria che genera gli ingressi agli stack, solo lo stack alla posizione x risulterà comandato.

Seguendo questo ragionamento si ottiene la rete di Figura 6.12. Si noti che, come in precedenza, J_h codifica la via su cui si ha l'hit, mentre h indica semplicemente che c'è stato hit. Allo stesso modo V_x codifica il numero di via non valida selezionato secondo il criterio della priorità, mentre $v_x = 0$ indica che ci sono linee non valide. In tal modo i due multiplexer della soluzione precedente si riducono alla rete riportata in alto a destra di Figura 6.12.

In Figura 6.12 i simboli che hanno il pedice x sono relativi alla posizione x, mentre tutti gli altri si riferiscono all'intera cache. Le reti nell'angolo in alto a destra e in basso a sinistra sono ripetute per ogni posizione x, le reti negli altri due angoli sono uniche per tutta la cache.

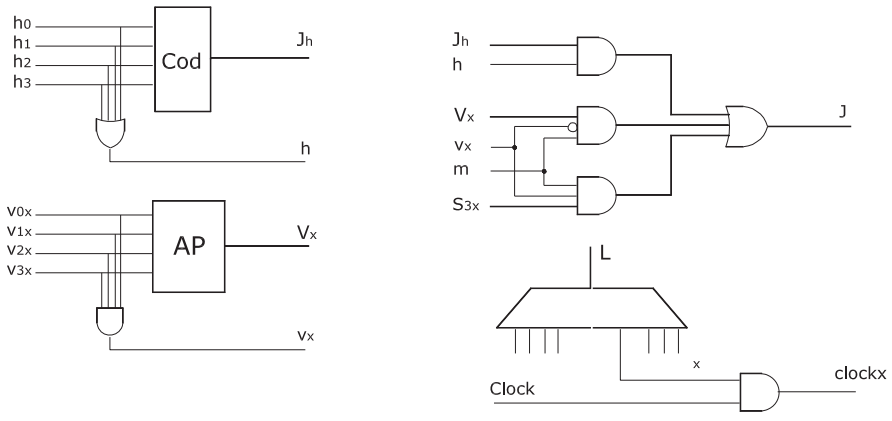

Figura 6.12 (Esercizio 6.16) Soluzione ottimizzata. La soluzione si basa sul fatto che il clock viene applicato solo allo stack LRU della linea x.

6.17 Riferiamoci alla Figura 6.6 del testo. Diamo una prima soluzione facendo l'ipotesi che, per quanto attiene allo scorrimento, i registri vengano riguardati come composti da flip-flop D.

Mentre l'ingresso a S_0 è dato da J [2], per gli ingressi agli altri contatori valgono le seguenti osservazioni:

- S_1 deve ricevere in ingresso:
  - l'uscita del contatore S_0 se J$\neq S_0$;
  - l'uscita di se stesso se J$=S_0$.
- S_2 deve ricevere in ingresso:
  - l'uscita del contatore S_1 se J$\neq S_0 \wedge$ J$\neq S_1$;
  - l'uscita di se stesso se J$=S_0 \vee$ J$=S_1$.
- S_3 deve ricevere in ingresso:
  - l'uscita del contatore S_2 se J$\neq S_0 \wedge$ J$\neq S_1 \wedge$ J$\neq S_2$;
  - l'uscita di se stesso se J$=S_0 \vee$ J$=S_1 \vee$ J$=S_2$.

Partendo dalle precedenti relazioni si possono tracciare le reti che danno gli ingressi ai registri dello stack LRU; basta impiegare un circuito di confronto in luogo degli operatori "$\oplus$" e "$\equiv$" oltre a porte AND e OR per gli operatori "$\wedge$" e "$\vee$" rispettivamente. Ad esempio, per il contatore S_2 si ha la rete di Figura 6.13.

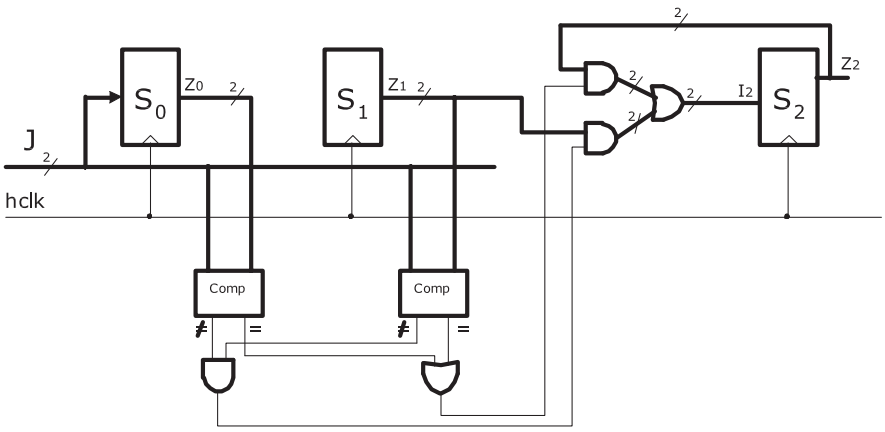

Figura 6.13 (Esercizio 6.17) Rete per la generazione dell'ingresso al contatore S_2. Si noti il parallelismo di 2 bit sulle linee che portano codificato un numero (0, 1, 2, 3); queste linee sono tracciate più grosse rispetto a quelle che portano un singolo segnare binario.

In Figura 6.13 si è adottata la convenzione di portare alle due porte AND/OR in fronte a S_2 la coppia di segnali che codifica un numero e il segnale avente effetto di selezione, come illustrato in Figura 6.14, dove viene anche presentata la realizzazione del comparatore di Figura 6.13.

Possiamo dare anche le espressioni dettagliate per l'ingresso al generico contatore S_i. A tale scopo conviene indicare con J_H e J_L (rispettivamente il bit più e meno significativo) i due bit che danno J, con I_{Hi} e I_{Li} $(i = 0, 1, 2, 3)$ i due ingressi al contatore $i-$mo e con Z_{Hi} e I_{Li} le relative uscite. Si ricavano le espressioni per I_{Hi} e I_{Li}.

Riferendoci alla Figura 6.13:

[2] Ovviamente, si assume che hclk (Figura fig:cache:lru:stack del testo) venga asserito solo quando se c'è hit alla posizione a cui è associato lo specifico LRU stack.

Figura 6.14 (Esercizio 6.17) A sinistra la selezione, a destra il confronto tra due contatori.

- la condizione J$\neq$ S_0 si esprime come: $((J_L \oplus Z_{L0}) + (J_H \oplus Z_{H0}))$;
- la condizione J$\neq$ S_1 si esprime come: $((J_L \oplus Z_{L1}) + (J_H \oplus Z_{H1}))$;
- la condizione J=S_0 si esprime come: $((J_L \equiv Z_{L0}) \cdot (J_H \equiv Z_{H0}))$;
- la condizione J=S_1 si esprime come: $((J_L \equiv Z_{L1}) \cdot (J_H \equiv Z_{H1}))$.

$$
\begin{aligned}
I_{L2} &= Z_{L1} \cdot (((J_L \oplus Z_{L0}) + (J_H \oplus Z_{H0})) \cdot ((J_L \oplus Z_{L1}) + (J_H \oplus Z_{H1}))) \; + \\
&+ Z_{L2} \cdot (((J_L \equiv Z_{L0}) \cdot (J_H \equiv Z_{H0})) + ((J_L \equiv Z_{L1}) \cdot (J_H \equiv Z_{H1}))) \\
I_{H2} &= Z_{H1} \cdot (((J_L \oplus Z_{L0}) + (J_H \oplus Z_{H0})) \cdot ((J_L \oplus Z_{L1}) + (J_H \oplus Z_{H1}))) \; + \\
&+ Z_{H2} \cdot (((J_L \equiv Z_{L0}) \cdot (J_H \equiv Z_{H0})) + ((J_L \equiv Z_{L1}) \cdot (J_H \equiv Z_{H1})))
\end{aligned}
$$

In modo analogo si scrivono le espressioni per (I_{H1}, I_{L1}) e per (I_{H3}, I_{L3}).

Si può arrivare a una soluzione meno costosa se i flip-flop che compongono i contatori sono di tipo SR o JK. Infatti, in questo caso non c'è bisogno di riportare in ingresso lo stato di ciascun elemento quando esso non deve eseguire una transizione di stato, basta dare l'ingresso S=R=0 ovvero J=K=0. Restano naturalmente valide le condizioni di selezione dell'ingresso, solo che, nel caso in cui lo stato non debba cambiare, basta presentare al registro l'ingresso 0. In Figura6.15 viene presentato lo schema corrispondente. Ai registri S_1, S_2 e S_3 viene portato lo stato del precedente quando c'è da scorrere, altrimenti viene portato 0.

6.18 In Tabella 6.4 si riporta l'evoluzione dei contatori a partire dalla condizione iniziale. Per confrontare il contenuto di Tabella 6.4, con quello di Tabella 6.2 del testo si osservi che:

- la riga 0 di 6.4 (1, 0, 3, 2), dice che l'ordine dei riferimenti è stato sulle vie, dalla più recente alla meno recente, 1 0 3 2 (esattamente coincidente con l'ordine espresso dallo stack LRU);
- la riga 2 (1, 2, 0, 3) dice che l'ordine dei riferimenti è stato 2 0 1 3 (come espresso dallo stack LRU)

La precedente osservazione mostra che una data sequenza può portare a tabelle all'apparenza diverse anche se di identica interpretazione. Quindi **non** è vero che si producano identiche tabelle.

L'ultima riga di Tabella 6.4, identica all'ultima riga dello stack LRU (Tabella 6.2 del testo), dice che l'ordine dei riferimenti è stato 1 0 3 2 come riportato nello stack. Se

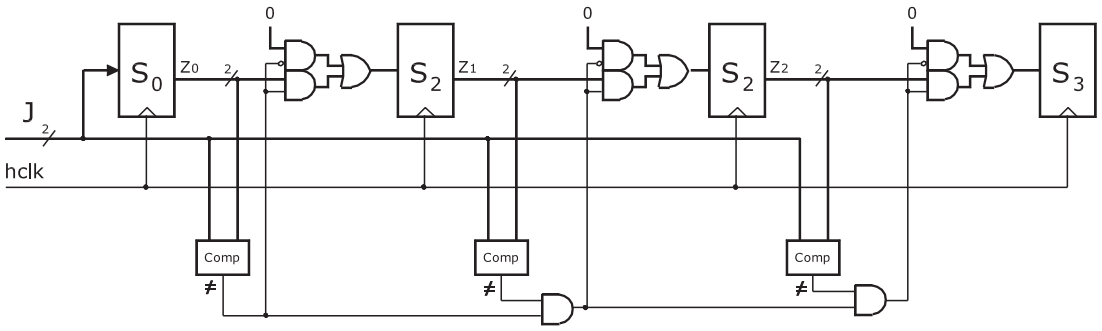

Figura 6.15 (Esercizio 6.17) Miglioramento dell arete di Figura 6.13.

ora si aggiungesse l'evento "hit sulla linea di via 3" i contatori raggiungerebbero questa configurazione $C_0\ C_1\ C_2\ C_3 = 2\ 1\ 3\ 0$, a indicare l'ordine 3 1 0 2 che apparirebbe in modo naturale nello stack.

In sintesi una sequenza ordinata di miss, a partire da identica configurazione, porta a una evoluzione identica nella forma oltre che nella sostanza, in quanto i contatori sono incrementati modulo 4, ciò che corrisponde a ruotare (spostando verso destra) il contenuto dello stack. Un hit non rispetta necessariamente tale modo di incrementare (ruotare) e dunque può dare luogo a una forma diversa anche se l'interpretazione è la medesima.

	Evento	C_0	C_1	C_2	C_3	Bit Val.
0	condizione iniziale	1	0	3	2	1111
1	hit sulla linea di via 0	0	1	3	2	1111
2	miss	1	2	0	3	1111
3	invalidazione linea di via 1	1	2	0	3	1011
4	hit sulla linea di via 0	0	2	1	3	1011
5	invalidazione linea di via 3	0	2	1	3	1010
6	miss	1	0	2	3	1110
7	hit sulla linea di via 2	2	1	0	3	1110
8	miss	3	2	1	0	1111
9	miss	0	3	2	1	1111
10	miss	1	0	3	2	1111

Tabella 6.4 (Esercizio 6.18) Esempio di politica LRU per una cache a 4 vie. Aggiornamento dei contatori relativi alle 4 linee in posizione I_L. L'evento si manifesta a partire nella condizione della riga precedente rispetto a quella su cui l'evento è riportato e determina la condizione della riga stessa.

6.19 Una sequenza ordinata di soli hit fa sostituire una volta da un lato e la successiva dall'altro, incluse le due metà. Ma, se dopo un miss che ha sostituito, per esempio, a destra (lasciando una traccia che direbbe che al prossimo miss la sostituzione è da fare a sinistra), si verifica un hit nella parte sinistra, esso determina la condizione per cui, se segue un miss, il miss va a sostituire sul lato destro; in altre parole, va a sostituire non

la linea più stagionata, ma quella immediatamente precedente nell'ordine temporale dei riferimenti.

6.20 Se la scrittura è write-through il protocollo non è più MESI in quanto la politica write-through fa scomparire lo stato M, mantenendo le linee di cache sempre coerenti con la memoria. Il protocollo si modifica come in Figura 6.16.

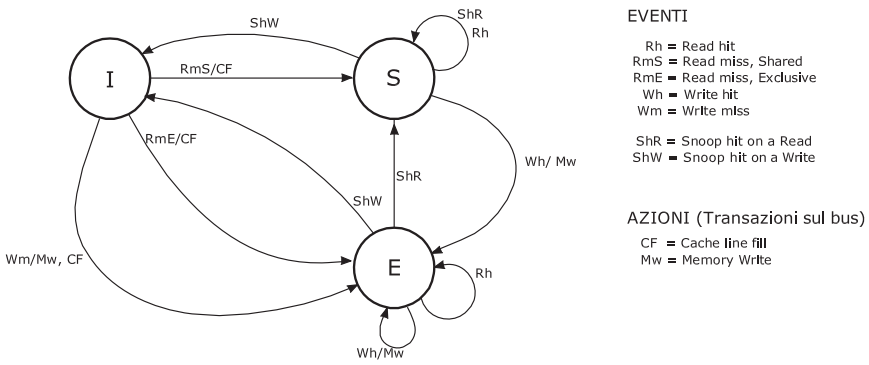

Figura 6.16 (Esercizio 6.20) Modifica del protocollo MESI a seguito dell'impiego della politica di write-through.

Nel tracciare la Figura 6.16 abbiamo assunto che in caso di scrittura questa determini la transazione Mw (Memory write sul bus). Nel caso di miss (possibile solo nell stato I) ciò determina anche il caricamento della linea in cache che passa quindi allo stato E (perché è l'unica copia ed è coerente con la memoria). Notare che un Wh in stato S fa passare la linea in stato E. In questo le altre linee in stato S si accorgono di questa scrittura (evento ShW) e passano allo stato I.

6.21 Come nel caso precedente il diagramma di stato presenta 3 stati: I, E e S.

- Per via della politica scelta, un Wm (possibile solo nello stato I) determina la scrittura della parola in memoria (Mw) e l'immediato, successivo caricamento della linea (CF). La linea dovrà passare allo stato E se essa risulterà essere l'unica copia in cache, ovvero allo stato S, se esiste altra copia in stato S o E (che pure si dovrà portare allo stato S). In altre parole si deve differenziare l'evento Wm in WmE e WmS; il primo è un write miss quando nessuna copia della linea è nelle cache, il secondo è un write miss quando nelle altre cache c'è almeno una copia in stato E o S. La differenziazione tra i due eventi è data dal fatto che nel primo caso nessun processore risponde (asserisce un opportuno segnale sul bus), mentre nel secondo caso almeno un processore segnala che ha copia della linea nella sua cache (e che questa viene aggiornata con il dato scritto in memoria –che sta passando sul bus– portandosi nello stato S). Ovviamente, sulla cache locale il WmE porta nello stato E, mentre il WmS porta nello stato S.

- Una linea in stato E che osserva un ShW (ovvero che un altro processore sta scrivendo in memoria ad una posizione ad essa corrispondente), aggiorna il suo contenuto (Cw) e e si porta in stato S (il Wm è un WmS). Notare che se una linea in stato E osserva ShW, esso corrisponde necessariamente a un miss di scrittura su un altro processore.
- Una linea in stato S che osserva un ShW, deve solo aggiornare il suo contenuto (Cw) con il dato con il dato che passa (oltre, naturalmente a segnalare il riconoscimento del Mw); anche in questo caso il Wm è un WmS. Notare che per una linea in stato S il ShW può corrispondere a un WmS su altro processore con linea in stato I, ovvero a un Wh/Mw su altro processore con linea in stato S.

Un simile protocollo è illustrato in Figura 6.17. Esso è puramente teorico (in altre parole, lo scrivente non ha verificato se viene realmente seguito da qualche processore reale, cosa peraltro improbabile dato l'alto traffico sul bus che esso comporta).

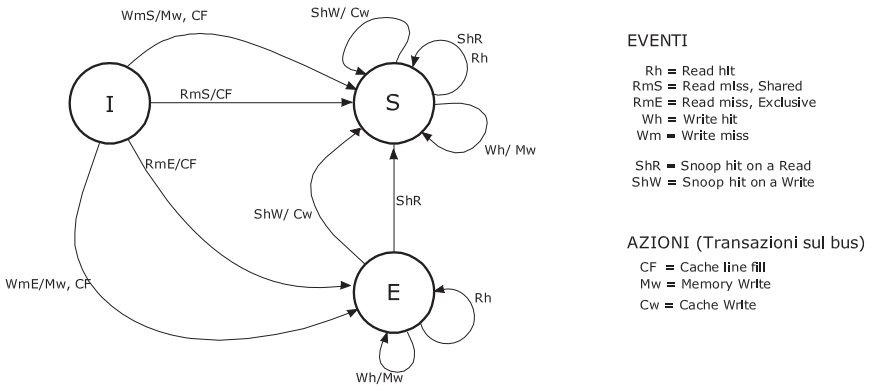

Figura 6.17 (Esercizio 6.21) Diagramma di stato del protocollo di coerenza con politica di write-through, write-allocate e propagazione delle modifiche.