Capitolo 12

Controllo microprogrammato e sistemi complessi

La parte controllo di un sistema ha natura di rete sequenziale, ed è stata progettata nel capitolo precedente con i metodi classici relativi a queste reti. Quando il sistema diviene complesso la parte controllo cresce di dimensioni e viene realizzata con una tecnica nuova, che non ha più l'obiettivo di ottenere una rete minima, ma di facilitare lo sviluppo del progetto e di comprendere più facilmente la struttura interna del controllo. Questa nuova tecnica è basata sul concetto di *controllo microprogrammato* (*μprogrammato*), e utilizza una ROM per la parte combinatoria della rete.

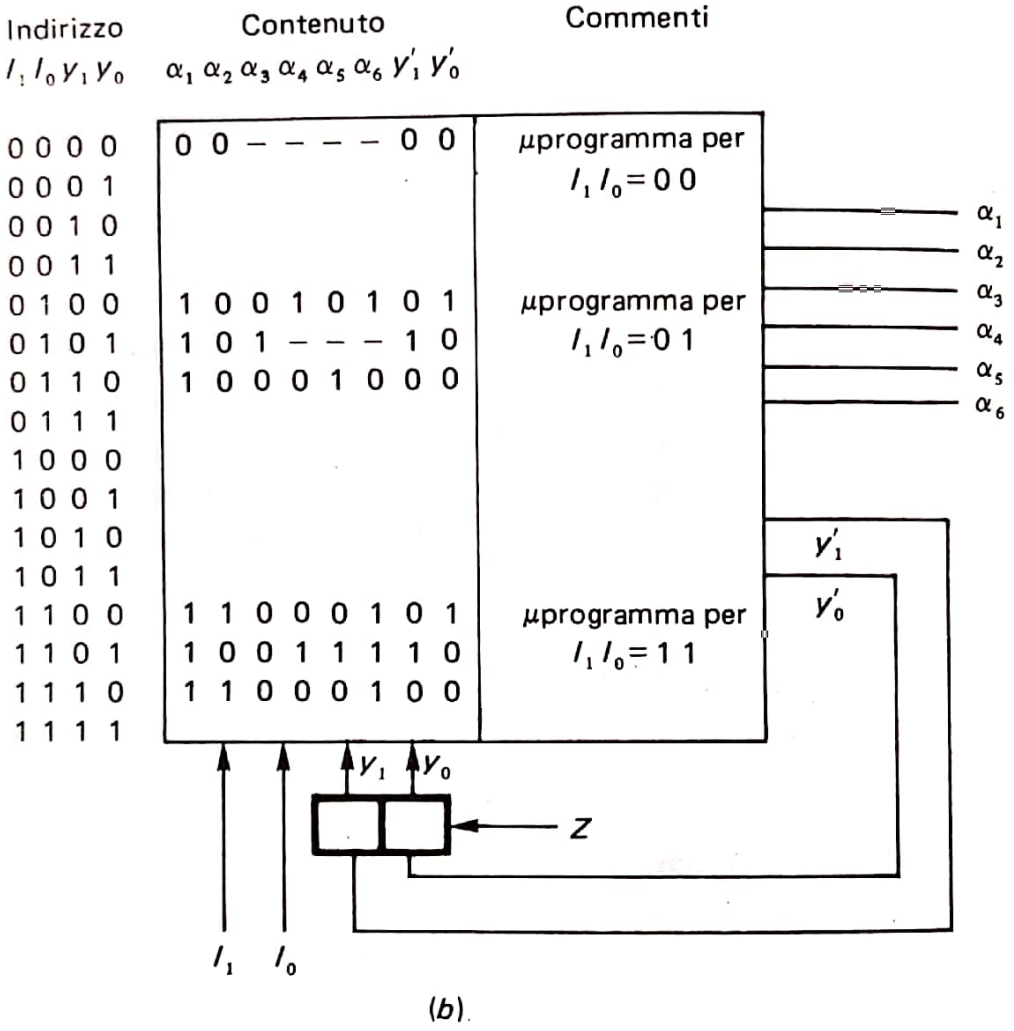
12.1 ROM e μprogrammi

Come sappiamo, la rete sequenziale di controllo riceve come ingressi i segnali I che rappresentano le istruzioni e i segnali β che rappresentano le condizioni che provengono dalla parte operativa, e produce come uscite i segnali α che costituiscono le μistruzioni. Nel controllo μprogrammato la parte combinatoria della rete è costituita da una ROM, in cui ogni cella "memorizza" direttamente una μistruzione e i segnali y' che specificano il prossimo stato: come vedremo, tale stato è ora interpretato come parte dell'"indirizzo" della cella contenente la prossima μistruzione da eseguire.

Realizziamo per esempio il controllo μprogrammato per il problema 11.3 (cap. 11): è questo un caso particolarmente semplice, perché il sistema è privo di segnali di condizione β. La figura 12.1*a* ripete la tabella di flusso della parte controllo, e la figura 12.1*b* mostra la corri-

Stato	$I_1 I_0$ = 00	01	11	$Y_1 Y_0$
ϵ	ϵ, 00 – – – –	γ, 1 0 0 1 0 1	γ, 1 1 0 0 0 1	ϵ = 0 0
γ	–, – – – – – –	δ, 1 0 1 – – –	δ, 1 0 0 1 1 1	γ = 0 1
δ	–, – – – – – –	ϵ, 1 0 0 0 1 0	ϵ, 1 1 0 0 0 1	δ = 1 0

(*a*)



(*b*)

Figura 12.1
Problema 11.3: (*a*) tabella di flusso della parte controllo; (*b*) parte controllo μprogrammata.

spondente parte controllo μprogrammata, ove è specificato, cella per cella, il contenuto della ROM (le celle lasciate in bianco non sono utilizzate). La struttura generale così ottenuta coincide con quella standard della rete sequenziale di controllo in assenza di segnali β (confronta le figg. 12.1*b* e 11.1), e il contenuto della ROM viene stabilito come segue.

I bit più significativi degli indirizzi della ROM sono forniti dai segnali di istruzione $I_1 I_0$, e i meno significativi dai segnali di stato $y_1 y_0$. Codificato con 00 lo stato iniziale ϵ per ogni istruzione (tale stato si carica con un segnale Z di azzeramento del registro), le μistruzioni relative all'istruzione $I_1 I_0$ sono memorizzate in celle che partono dall'indirizzo $I_1 I_0 0\,0$, e hanno tutte indirizzi che iniziano con $I_1 I_0$. Tali μistruzioni, estese per includere i segnali y' di prossimo stato, costituiscono il *μprogramma* relativo all'istruzione. Per quanto detto, ogni μprogramma si trova in indirizzi della ROM che iniziano con i relativi bit $I_1 I_0$.

Per esempio il μprogramma dell'istruzione $I_1 I_0 = 01$ è memorizzato nelle celle: 0100, 0101, 0110. Nella prima di queste si trova la μistruzione 10010101, con cui inizia l'esecuzione dell'istruzione: i segnali α: 100101 specificano le azioni della parte operativa, mentre i segnali y': 01 indicano che la prossima μistruzione deve essere reperita nella cella 0101. La μistruzione contenuta in questa cella contiene i bit y': 10, e rimanda perciò alla cella 0110 che contiene l'ultima μistruzione, e rimanda a sua volta alla cella iniziale 0100.

Si noti che le funzioni combinatorie realizzate dalla ROM sono largamente non specificate, e quindi molte celle non sono utilizzate (fig. 12.1*b*). In termini di struttura interna della ROM (vedi fig. 8.3), è indifferente che esistano o meno degli OR sulle linee uscenti dal decodificatore, corrispondenti alle celle non utilizzate; tuttavia la dimensione complessiva del componente non si riduce. Come a suo tempo spiegato, questa cattiva utilizzazione della piastrina è inevitabile nel progetto di reti combinatorie mediante ROM; la comodità d'uso di questi componenti è d'altronde particolarmente evidente nel controllo μprogrammato, dove i μprogrammi sono direttamente memorizzati e quindi "visibili" all'interno della ROM.

Nell'esempio precedente abbiamo costruito la parte controllo μprogrammata di un sistema, a partire dalla sua tabella di flusso già nota. In genere però la parte controllo viene costruita direttamente scrivendo il μprogramma per ogni istruzione, e memorizzandolo nella ROM. (A tale scopo si usano diversi *linguaggi di μprogrammazione*, che sono tutti equivalenti nelle capacità di esprimere μprogrammi e non vengono qui esaminati: il lettore può riferirsi alla bibliografia.) Inoltre il sistema può prevedere la presenza di condizioni β che complicano la stesura del μprogramma. Per studiare il progetto generale di una parte controllo μprogrammata, esaminiamo il seguente problema.

Problema 12.1

Costruire un sistema a controllo μprogrammato con tre registri P, Q, R contenenti interi positivi, che esegue le istruzioni:

I_1	I_0	
0	0	**alt**
0	1	**if** ($P+Q$ non dà supero) **then** $P+Q \to P$ **else** $P/2+Q/2 \to P$
1	0	$P/2^R \to P$

Non indichiamo esplicitamente la struttura della parte operativa, perché è ovvia. Essa contiene: una ALU per l'esecuzione dell'addizione, con uscita SUP di supero; i due registri P e Q a spostamento destro per eseguire la divisione per due; il registro R con funzioni di contatore a decremento, dotato di un blocco OR sulle uscite per controllare, attra-

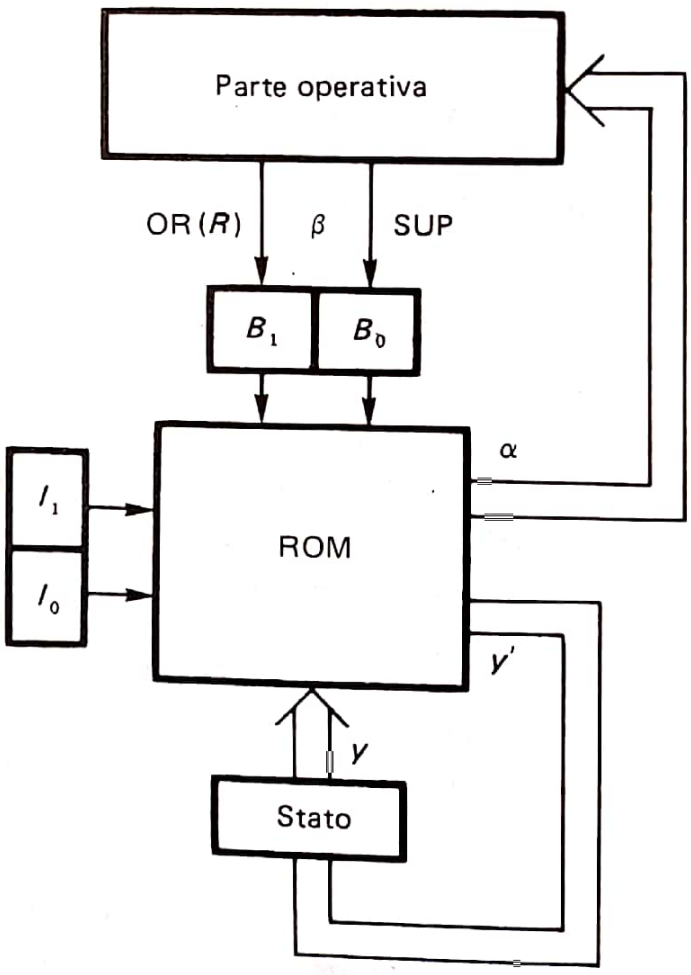


Figura 12.2
Problema 12.1: struttura del sistema.

verso la condizione $OR(R)=0$, quando il contenuto del registro si è azzerato (ciò consente di ripetere un ciclo R volte: in questo caso si deve ripetere R volte la divisione di P per 2).

Le condizioni $\beta_1 = OR(R)$ e $\beta_0 = SUP$ sono memorizzate nei flip-flop $B_1 B_0$ di un registro di condizione, e di qui inviate alla parte controllo. Similmente le istruzioni sono memorizzate nei flip-flop $I_1 I_0$ di un registro di istruzione. Lo schema generale del sistema è quindi quello riportato nella figura 12.2 (non sono indicate le connessioni per la distribuzione dell'impulso di sincronismo).

Descriviamo anzitutto il funzionamento della parte operativa attraverso le μsequenze relative alle istruzioni. Tali μsequenze, riportate nella tabella 12.1, sono formulate nell'ipotesi che i registri di stato e di condizione siano inizialmente azzerati. Le corrispondenti μistruzioni sono indicate con i simboli $\mu_1, \ldots, \mu_8$ senza specificare i valori dei segnali α, che possono essere ovviamente dedotti dalle μsequenze.

Il contenuto della ROM della parte controllo è riportato nella figura 12.3. Per seguire l'evoluzione dei μprogrammi è stato indicato il tracciato seguito da un punto operativo, per le diverse istruzioni. Notiamo anzitutto che, a causa dell'azzeramento iniziale dei registri contenenti i segnali y e β, i μprogrammi sono memorizzati a partire dalle celle di indirizzo $I_1 I_0 0000$. L'istruzione 00 richiede semplicemente il congelamento di ogni elaborazione (μistruzione μ_1). L'istruzione 01 genera un μprogramma che segue due diversi tracciati, a seconda che la prima μistruzione μ_2 abbia caricato il valore 0 o 1 nel flip-flop B_0.

Tabella 12.1 μsequenze per il problema 12.1

I_1 I_0	μsequenze	segnali α
0 0	ϕ	μ_1
0 1	1: SUP → B_0;	μ_2
	if B_0 = '0' then $P+Q \to P$, goto 1	μ_3
	else SD(P) → P, SD(Q) → Q;	μ_4
	$P+Q \to P$, '0' → B_0, goto 1	μ_5
	fi	
1 0	1: OR(R) → B_1;	μ_6
	if B_1 = '0' then ϕ, goto 1	μ_7
	else SD(P) → P, DECR(R) → R, goto 1	μ_8
	fi	

Indirizzo $I_1 I_0 B_1 B_0 Y_1 Y_0$	Contenuto α	$Y_1' Y_0'$	Tracciati μprogramma
0 0 0 0 0 0	μ_1	0 0	
0 1 0 0 0 0	μ_2	0 1	
0 1 0 0 0 1	μ_3	0 0	
0 1 0 1 0 1	μ_4	1 0	
0 1 0 1 1 0	μ_5	0 0	
1 0 0 0 0 0	μ_6	0 1	
1 0 0 0 0 1	μ_7	0 0	
1 0 1 0 0 0	μ_6	0 1	
1 0 1 0 0 1	μ_8	0 0	

Figura 12.3
Contenuto della ROM del controllo per il problema 12.1. (Le celle non indicate hanno contenuto non specificato.)

La μistruzione successiva si trova perciò, nei due casi, in uno degli indirizzi 01 00 01 oppure 01 01 01 (che differiscono per il solo bit B_0).

Più complessa è la struttura del μprogramma per l'istruzione 10. La prima μistruzione μ_6 carica il valore 0 o 1 nel flip-flop B_1, e la μistruzione successiva si trova quindi in uno degli indirizzi 10 00 01 oppure 10 10 01. Nel primo caso si torna al passo iniziale senza eseguire alcuna operazione (μistruzione μ_7, identica alla μ_1). Nel secondo caso si esegue la μistruzione μ_8, dopo la quale deve aver luogo un nuovo caricamento del flip-flop B_1 (vedi tab. 12.1, dove la μoperazione corrispondente alla μ_8 termina con **goto** 1). Si noti d'altra parte che l'esecuzione della μ_8 non altera il contenuto di B_1, e quindi la μistruzione seguente si trova nell'indirizzo 10 10 00, ove si trova nuovamente la μ_6 per il caricamento di B_1. Da questo punto si raggiunge il nuovo indirizzo 10 00 01 oppure 10 10 01, a seconda che la μ_6 abbia caricato 0 oppure 1 in B_1.