Capitolo 10

Struttura dei sistemi di elaborazione

Seguendo una definizione classica chiamiamo *sistema di elaborazione dell'informazione*, o brevemente *sistema* (se chiaramente limitato al nostro ambito di studio), una rete logica che può eseguire sequenze di operazioni sotto la guida di *istruzioni* esterne.

Come ogni rete, un sistema comunica con l'esterno attraverso morsetti di ingresso e di uscita: le operazioni sono infatti eseguite su dati comunicati dall'esterno, e forniscono risultati verso l'esterno. Il sistema acquisisce però attraverso i morsetti di ingresso anche le istruzioni, che rappresentano un'importante novità rispetto a quanto abbiamo fin qui studiato sulle reti.

L'impiego delle istruzioni non rende la rete più potente, o comunque diversa in linea di principio, delle altre reti logiche: i sistemi, infatti, altro non sono che reti sequenziali, e potrebbero essere studiati con le tecniche che riguardano queste, se la loro grande complessità non consigliasse di affrontarne l'esame in modo differente.

Scomporremo così i sistemi in parti interconnesse tra loro, esamineremo queste parti indipendentemente e studieremo l'effetto del loro collegamento. Particolarmente utile sarà la distinzione tra dati e istruzioni all'ingresso, che giocheranno un ruolo diverso nel funzionamento delle parti interconnesse. Solo quando giungeremo ad analizzare sistemi di grande complessità, come il "sistema calcolatore di tipo von Neumann" (cioè la struttura dei calcolatori costruiti fino ad oggi), la distinzione tra dati e istruzioni sarà in parte o in tutto perduta. I semplici sistemi che studieremo ora possono comunque essere parte di sistemi più complessi, e ne costituiscono in qualche modo il cuore.

10.1 Parte operativa e parte controllo

Secondo una prima fondamentale scomposizione si divide il sistema in due parti, dette *parte* (o *unità*) *operativa* e *parte* (o *unità*) *di controllo*. Tali parti sono reti sequenziali collegate con l'esterno e tra loro secondo lo schema della figura 10.1.

La parte operativa *esegue* le sequenze di operazioni per cui il sistema è costruito, trasformando i *dati X* in *risultati Z*. La parte controllo *interpreta* le istruzioni *I* e le traduce in sequenze di *comandi* α per la parte operativa. Sovente l'esecuzione di un'istruzione deve tener conto di risultati intermedi dell'elaborazione, che sono generati nella parte operativa; questa parte deve quindi comunicare alla parte controllo opportune informazioni, dette *condizioni* β, sullo stato del calcolo, che influenzeranno le sequenze di comandi α.

La struttura complessiva dei collegamenti del sistema, descritta per ora in modo del tutto generico, acquisterà significato preciso quando discuteremo i primi esempi pratici. Notiamo per il momento che l'emissione di comandi α influenzati da condizioni β risponde alle usuali strutture linguistiche di controllo, come la "**if** condizione **then** esecu-

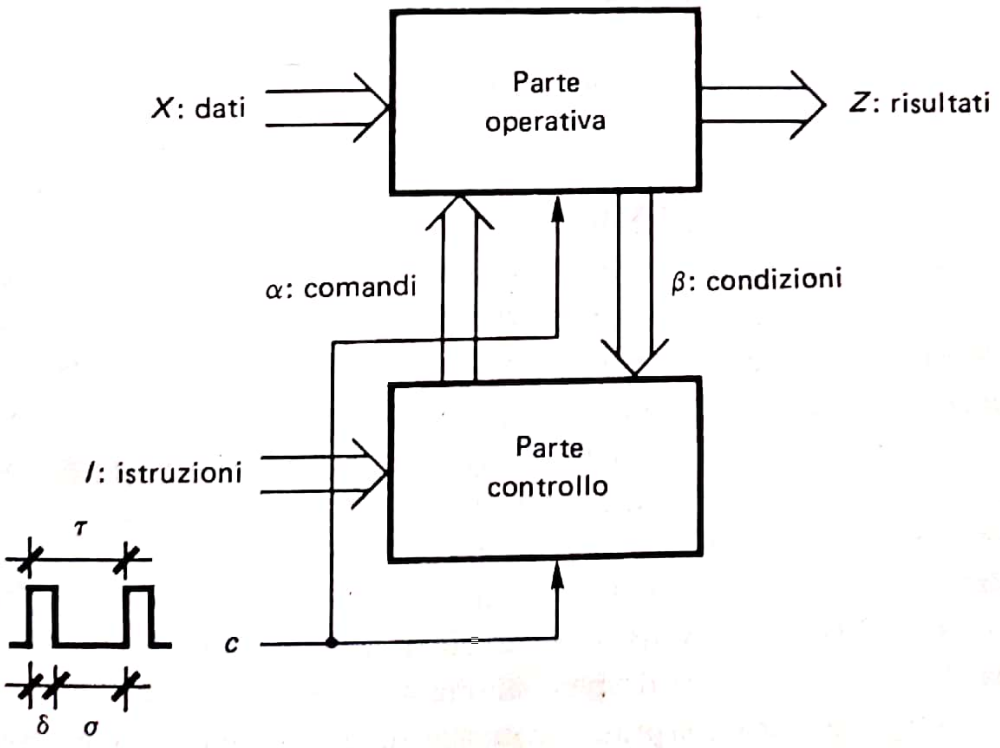


Figura 10.1
Scomposizione di un sistema in parte operativa e parte controllo.

zione 1 **else** esecuzione 2", ove i costrutti "condizione", "esecuzione 1", "esecuzione 2" sono formulati su variabili note alla parte operativa.

Ammetteremo nel nostro studio che la parte operativa e la parte controllo siano reti sequenziali sincrone, temporizzate dal medesimo impulso c (vedi § 4.5). Tutte le parti del sistema funzionano quindi con lo stesso ritmo, e le operazioni elementari sono distanziate di un tempo τ pari al periodo dell'impulso (fig. 10.1). I canali di collegamento α e β formano anelli nella rete complessiva, scorporati dalle reti sequenziali operativa e di controllo. Per rispettare le regole di funzionamento sincrono anche tali anelli devono contenere un registro comandato dall'impulso c, che verrà tipicamente aggiunto agli ingressi β della parte controllo, se non ne esiste già uno nel sistema.

Ancorché l'ipotesi di sincronicità di tutte le parti di un sistema sia in genere eccessivamente semplificativa, è tuttavia accettabile per sistemi di piccole dimensioni, ed è sempre verificata per opportuni sottosistemi di ogni sistema dato, che, a loro volta, possono essere visti come sistemi elementari. Sarà quindi perfettamente accettabile nel nostro studio, che ha lo scopo di illustrare concetti di base.

10.2 Parte operativa: descrizione a livello di registri

La parte operativa di un sistema può essere studiata scomponendola a sua volta in parti che eseguano operazioni specifiche, come moduli combinatori e sequenziali che realizzano funzioni standard, o funzioni speciali di volta in volta richieste.

Tra tali moduli poniamo i registri e le memorie, che sono sempre presi in esame singolarmente perché il loro contenuto evidenzia lo stato dell'elaborazione in ogni periodo τ. E' infatti usuale studiare la parte operativa di un sistema come composta di registri connessi attraverso moduli combinatori e sequenziali di vario genere, e descrivere il suo funzionamento a *livello di trasferimento di informazione tra registri*. Per tale descrizione si fa uso di un linguaggio detto RTL (per *Register Transfer Language*), i cui elementi atomici si riferiscono alle parti elementari del sistema e variano in funzione di esso, mentre i costrutti a più alto livello sono quelli tipici dei linguaggi programmativi.

Il linguaggio RTL esprime sequenze di operazioni, ciascuna delle quali si svolge in un intervallo di tempo elementare τ: esse sono dette *μsequenze* di *μoperazioni* (μ si legge "micro"); ogni μoperazione può

impegnare diverse parti del sistema, ed essere quindi costituita di un insieme di operazioni singole da eseguire nello stesso intervallo τ. In linea di massima una μsequenza esprime le azioni elementari della parte operativa del sistema, in risposta a un'istruzione presentata dall'esterno alla parte controllo.

Impiegheremo in questo testo un linguaggio RTL molto ridotto, ma sufficiente per lo studio di numerosi casi pratici, lasciando il lettore libero di immaginarne le estensioni che dovessero rendersi necessarie caso per caso. Nel linguaggio, descritto qui di seguito, i costrutti sintattivi ad alto livello sono rappresentati in "forma normale di Backus",[1] mentre i costrutti sintattici elementari e la semplicissima semantica saranno espressi a parole.

IL LINGUAGGIO RTL

⟨μsequenza⟩ ::= ⟨μpasso⟩ | ⟨μpasso⟩; ⟨μsequenza⟩

⟨μpasso⟩ ::= ⟨μoperazione⟩ | ⟨etichetta⟩ : ⟨μoperazione⟩
| ⟨frase if⟩ | ⟨etichetta⟩ : ⟨frase if⟩

⟨μoperazione⟩ ::= ⟨assegnamento⟩ | ϕ
| ⟨assegnamento⟩, ⟨frase go⟩ | ϕ, ⟨frase go⟩

⟨assegnamento⟩ ::= ⟨trasferimento⟩
| ⟨trasferimento⟩, ⟨assegnamento⟩

⟨trasferimento⟩ ::= ⟨dato⟩ → ⟨registro⟩
| ⟨dato⟩ ⟨op⟩ ⟨dato⟩ → ⟨registro⟩
| ⟨funzione⟩ (⟨argomento⟩) → ⟨registro⟩

⟨dato⟩ ::= ⟨registro⟩ | ⟨ingresso⟩ | '⟨costante⟩'

⟨argomento⟩ ::= ⟨dato⟩ | ⟨dato⟩, ⟨argomento⟩

⟨frase if⟩ ::= **if** ⟨condizione⟩ **then** ⟨μsequenza⟩
else ⟨μsequenza⟩ **fi**

⟨condizione⟩ ::= ⟨dato⟩ ⟨rel⟩ ⟨dato⟩
| ⟨funzione⟩ (⟨argomento⟩) ⟨rel⟩ ⟨dato⟩

⟨frase go⟩ ::= **goto** ⟨etichetta⟩

Nella sintassi mancano le definizioni dei metasimboli di più basso livello, cioè ⟨registro⟩, ⟨ingresso⟩, ⟨costante⟩, ⟨funzione⟩, ⟨op⟩, ⟨rel⟩ e ⟨etichetta⟩. Tali definizioni si pongono informalmente come segue, in funzione della particolare rete da descrivere.

[1] Chi non avesse familiarità con la descrizione sintattica di un linguaggio può consultare qualsiasi testo del campo.

⟨registro⟩ indica un qualsiasi registro della rete, o una porzione di registro, o una cella di memoria. Nel seguito useremo una notazione del tipo:

B, MBR, ACC; [10.1]

MAR_k, MAR_{h-k}; [10.2]

M[MAR]; [10.3]

a indicare: registri, [10.1]; il k-esimo bit di un registro, o la porzione di registro compresa tra i bit h e k, [10.2]; la cella della memoria M di indirizzo (specificato in) MAR, [10.3].

⟨ingresso⟩ indica un canale di ingresso di dati dall'esterno.

⟨costante⟩ indica una stringa di caratteri di lunghezza tale da essere contenuta in un registro o cella.

⟨funzione⟩ indica una funzione combinatoria calcolata da un modulo del sistema su uno o più dati. Nel seguito useremo le funzioni:

SD, SCD, SS, SCS; [10.4]

INCR, DECR; [10.5]

COMP; [10.6]

OR, AND; [10.7]

calcolate su un solo dato in costrutti sintattici del tipo SD(A) ecc.; tali funzioni indicano: lo spostamento destro, circolare destro, sinistro, circolare sinistro del dato, [10.4]; l'incremento o decremento aritmetico di 1, [10.5]; la complementazione bit a bit, [10.6]; l'OR o l'AND tra tutti i bit del dato, [10.7].

Funzioni su più dati saranno per esempio: MAX(A, B), MIN(A, B), che indicano il massimo e il minimo tra due dati, e così via.

⟨op⟩ indica un operatore aritmetico o logico applicato a due dati; tipicamente:

$+, -, *, /$;

$\vee, \wedge$;

di ovvio impiego.

⟨rel⟩ indica un operatore di relazione; ci limiteremo ai seguenti:

$=, \neq$ (tra stringhe o numeri);

$<, \leqslant, >, \geqslant$ (tra numeri).

⟨etichetta⟩ è nel nostro caso un numero intero.

La definizione semantica del linguaggio è in larga parte ovvia, poiché ricalca i costrutti caratteristici dei linguaggi di programmazione. Sottolineiamo comunque alcuni punti.

ϕ è la ⟨μoperazione⟩ vuota. Essa non richiede alcun trasferimento, cioè impone che il contenuto dei registri e della memoria rimanga inalterato.

I costrutti di controllo sono: la ⟨frase if⟩, ove la ⟨condizione⟩ è limitata al confronto tra i *valori* di due ⟨dati⟩ (vedi oltre), o di una ⟨funzione⟩ e di un ⟨dato⟩; e la ⟨frase go⟩ correlata a ⟨etichetta⟩. Tali frasi hanno significato usuale, e la loro esecuzione è affidata alla parte controllo del sistema. Notiamo in particolare che la verifica della ⟨condizione⟩ è eseguita dalla parte controllo, attraverso l'esame di un opportuno segnale binario costruito dalla parte operativa; tale verifica non richiede un proprio intervallo di tempo, come apparirà chiaro in seguito quando studieremo diversi esempi di sistemi. Poiché la ⟨μsequenza⟩ che segue il simbolo **then**, o **else**, può contenere a sua volta una ⟨frase if⟩, e così via, può essere richiesta la verifica di una catena di condizioni subordinate, che comunque non richiedono alcun tempo aggiuntivo.

Consideriamo ora la ⟨μoperazione⟩. Essa comprende un insieme di ⟨trasferimenti⟩, eventualmente seguiti da una ⟨frase go⟩. I trasferimenti devono essere eseguiti tutti in uno stesso intervallo di tempo elementare τ; quindi nel linguaggio la virgola separa azioni contemporanee, mentre il punto e virgola indica il passaggio al tempo successivo. A sua volta il ⟨trasferimento⟩ descrive il trasferimento di informazione tra registri, cioè l'operazione centrale del linguaggio. Il *valore* del ⟨dato⟩, o del risultato del calcolo, che appare a sinistra del simbolo →, è trasferito nel ⟨registro⟩ specificato a destra del simbolo stesso. Il trasferimento opera cioè, in un tempo elementare, sul *contenuto* di registri e canali, o su costanti, mediante i connettivi ⟨op⟩ e ⟨funzione⟩. Poiché tali operazioni sono eseguite da moduli della rete, come per esempio l'unità aritmetica logica (⟨op⟩), i registri a spostamento (⟨funzione⟩ SD e simili), i contatori (⟨funzione⟩ INCR), i significati che possono assumere ⟨op⟩ e ⟨funzione⟩ sono direttamente definiti dai moduli che li eseguono.

Per quanto riguarda la temporizzazione, ricordiamo che il funzionamento sincrono dei registri prevede che l'operazione di trasferimento (→) sia eseguita nel tempo δ in cui appare l'impulso (fig. 10.1). Il calcolo della parte sinistra del trasferimento, e l'istradamento dell'informazione attraverso il sistema, richiedono in genere il funzionamento di porzioni combinatorie della rete (ALU, selettori ecc.), che ha luogo

nell'intervallo σ che *precede* l'impulso di trasferimento. In tutto l'operazione si sviluppa in un tempo $\sigma + \delta = \tau$.

Invitiamo il lettore a controllare le regole sopra esposte sulle seguenti semplici μsequenze.

μsequenza 1:

DATO $\rightarrow N$;

1: **if** OR(N) = '0' **then** ϕ **else** DECR(N) $\rightarrow N$, $A + B \rightarrow A$, **goto** 1 **fi**

Questa μsequenza descrive un ciclo di N passi (N intero $\geqslant 0$), in ciascuno dei quali si esegue $A + B \rightarrow A$; N è caricato da un canale esterno DATO, e decrementato a ogni passo: la frase if controlla il valore dell'OR di tutti i bit di N, interrompendo il ciclo quando tale valore è divenuto 0. Complessivamente la sequenza calcola $A + NB$ e lascia questo valore in A. I componenti e le principali connessioni della parte operativa di un sistema che può realizzare le operazioni richieste sono mostrati nella figura 10.2: N è un registro contatore che consente di decrementare il dato; A e B sono registri; ALU è un'unità aritmetica logica specializzata in questo caso a eseguire l'addizione.

μsequenza 2:

if A_0 = '0' **then if** B_0 = '0' **then** $B \rightarrow C$ **else** $A \rightarrow C$ **fi else** $A \rightarrow C$ **fi**

Questa μsequenza esamina il bit meno significativo di due dati numerici A e B, che indica la parità (0) o disparità (1) di tali dati. Se A e B sono entrambi pari, B è trasferito in C, altrimenti A è trasferito

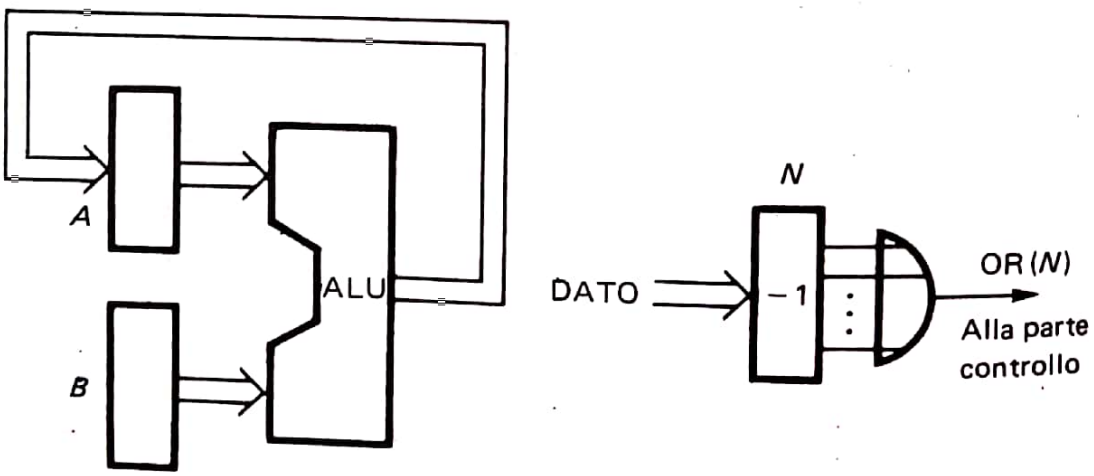


Figura 10.2
Schema di massima di una rete logica per l'esecuzione della μsequenza 1.

in C. La parte operativa di un sistema che può eseguire tale μsequenza deve includere un selettore interposto tra i registri A, B e C, e la parte controllo deve avere accesso ai bit A_0 e B_0 (fig. 10.3).

La seguente μsequenza 3, più complessa delle precedenti, descrive le operazioni necessarie a disporre in ordine crescente due dati numerici contenuti in due celle consecutive di una memoria M, ove l'indirizzo della prima cella è comunicato dal canale esterno DATO. M[MAR]→ →MBR e MBR→M[MAR] indicano rispettivamente la lettura e la scrittura nella memoria; si ammette che quest'ultima sia eseguita in un intervallo di tempo elementare, pur non essendo comandata dall'impulso c (fig. 6.14).

μsequenza 3:

1: DATO→A;
2: A→MAR, INCR(A)→A;
3: M[MAR]→MBR, A→MAR, DECR(A)→A;
4: MBR→B, M[MAR]→MBR, A→MAR, INCR(A)→A;
5: **if** $B \leqslant$ MBR **then** ϕ
else 6: MBR→M[MAR], B→MBR, A→MAR;
7: MBR→M[MAR] **fi**

Una rete che può realizzare queste operazioni è indicata nella figura 10.4. A è un registro contatore abilitato all'incremento e decremento

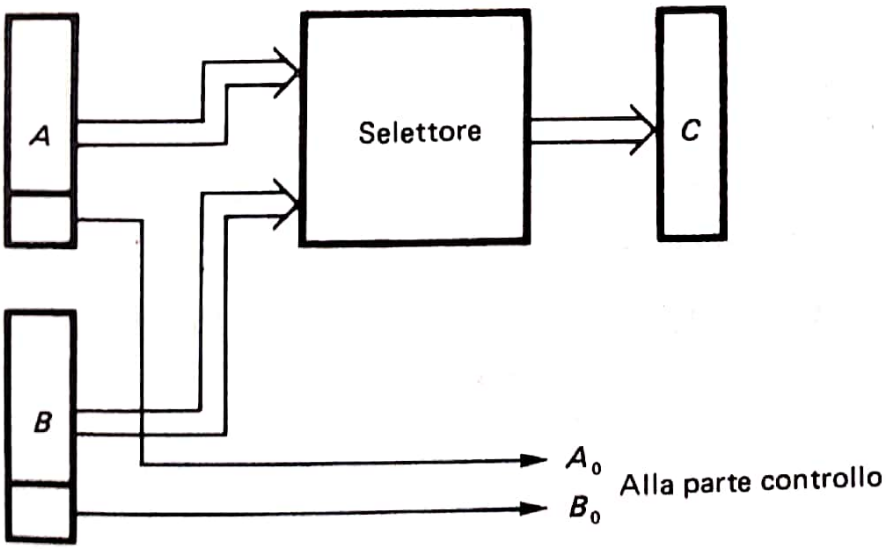


Figura 10.3
Rete logica per l'esecuzione della μsequenza 2.

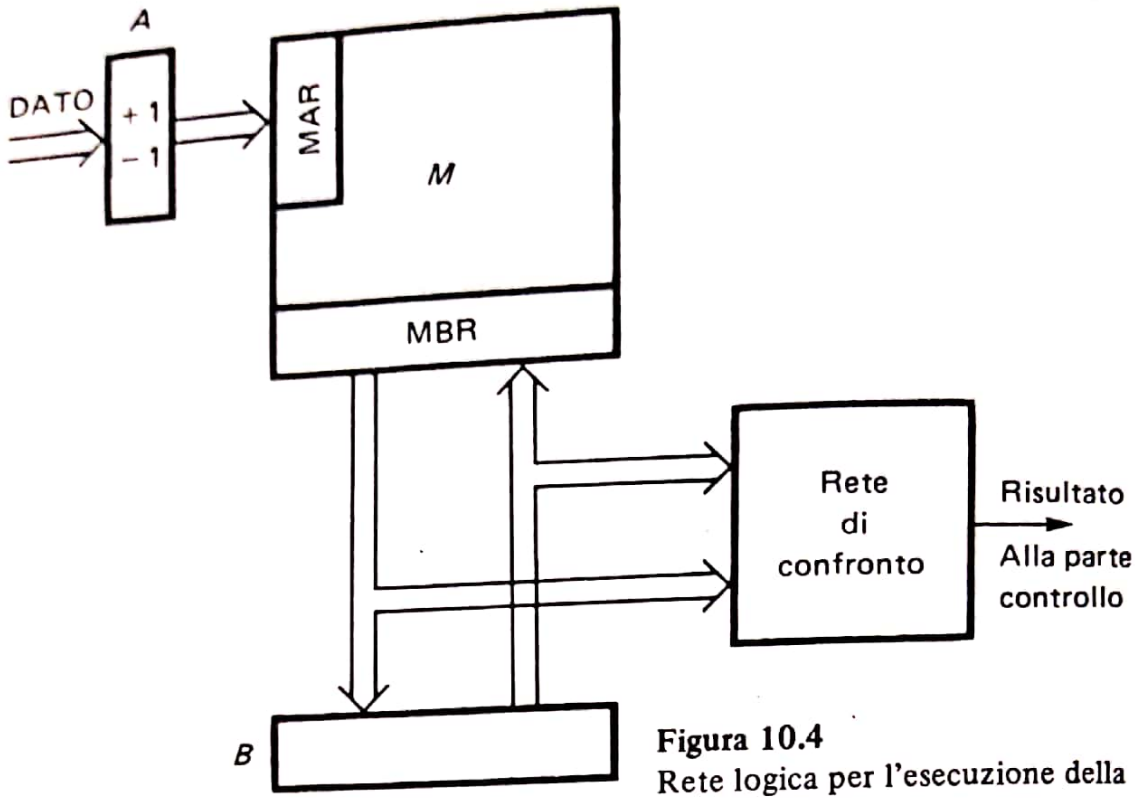


Figura 10.4
Rete logica per l'esecuzione della μsequenza 3.

di un dato. Il confronto tra B e MBR è eseguito da una rete combinatoria appositamente costruita, che invia il risultato alla parte controllo. (Se il sistema contiene una ALU, tale confronto si esegue in genere mediante la sottrazione tra i dati e l'esame del segno del risultato.)

Si noti la contemporaneità degli assegnamenti contenuti nelle μoperazioni 2, 3, 4, 6, che impegnano uno stesso registro in operazioni di ingresso e uscita del dato contemporaneamente (per esempio nella μoperazione 2 si carica A con il nuovo valore INCR(A), e nello stesso tempo si trasferisce in MAR il valore originale di A; come sappiamo, ciò è congruente con il funzionamento sincrono dei registri.

10.3 Microistruzioni e parte controllo

Le μsequenze del linguaggio RTL descrivono il funzionamento di reti complesse, ma non indicano esplicitamente i comandi necessari per imporre tale funzionamento. In un sistema scomposto in parte operativa e parte controllo (§ 10.1), il funzionamento della parte operativa è imposto da sequenze di segnali di comando α generati dalla parte controllo; per definire i compiti della parte controllo dovremo quindi *tradurre* ciascuna μoperazione della parte operativa in un insieme di comandi specifici: tale insieme è detto *μistruzione*.

I comandi inclusi in una μistruzione guideranno il funzionamento dei moduli combinatori, come selettori e ALU; dei registri a spostamento, contatori e simili; delle memorie; e abiliteranno alla lettura i registri in cui si deve effettuare un trasferimento di dati, attraverso i segnali A dei flip-flop FAc (vedi § 4.3). Tali comandi consistono di fatto nei valori di un insieme di segnali binari, scelti in funzione dei moduli e registri della parte operativa.

Le eventuali frasi di controllo (frase if o frase go) saranno per il momento ricopiate senza variazioni accanto alla μistruzione corrispondente: la loro trasformazione in termini di segnali sarà discussa nei prossimi capitoli, in relazione alla struttura interna della parte controllo. Notiamo però che la condizione contenuta in una frase if può richiedere che una μistruzione includa i comandi per la *generazione* del segnale di condizione, che deve poi essere esaminata dalla parte controllo.

Tutti questi concetti saranno ora chiariti attraverso alcuni esempi.

Riprendiamo in esame la μsequenza 1 del paragrafo precedente, realizzata dallo schema della figura 10.2. Per mettere in funzione i componenti dello schema dobbiamo assegnare valore opportuno ai segnali (non indicati nella figura):

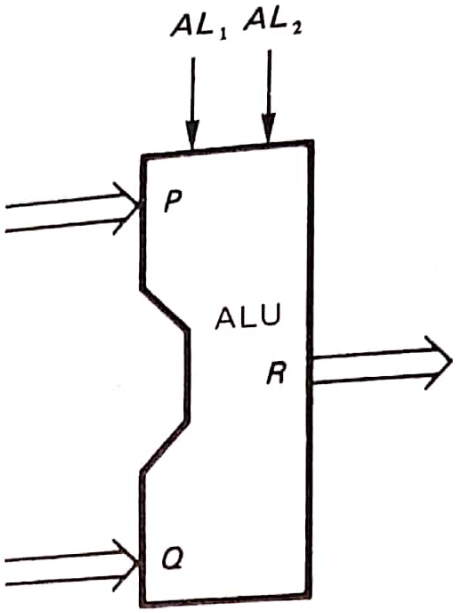
A_A e A_N, per abilitare il caricamento dei registri A e N;

K_N, connesso al registro N, per impiegare tale registro come contatore all'indietro ($K_N=1$, $A_N=1$, vedi fig. 6.8);

AL_1 e AL_2 per comandare il funzionamento di una semplice ALU, derivata da quella della figura 3.17, che può eseguire il trasferimento o l'incremento di un dato, l'addizione o la sottrazione tra due dati, secondo quando indicato nella figura 10.5.

La μsequenza 1 sarà quindi tradotta in μistruzioni che specificano i valori dei segnali A_A, A_N, K_N, AL_1, AL_2. Tali μistruzioni si rappresentano con un vettore binario che contiene ordinatamente i valori dei segnali. La traduzione è la seguente:

μoperazione	μistruzione				
	A_A	A_N	K_N	AL_1	AL_2
DATO→N;	0	1	0	–	–
1: **if** OR(N)='0' **then** ϕ	0	0	–	–	–
else DECR(N)→N, $A+B$→A, **goto** 1 **fi**	1	1	1	1	0



R	AL_1	AL_2
Q	0	0
Q + 1	0	1
P + Q	1	0
P − Q	1	1

Figura 10.5
Operazioni di una ALU e relativi valori dei segnali di comando.

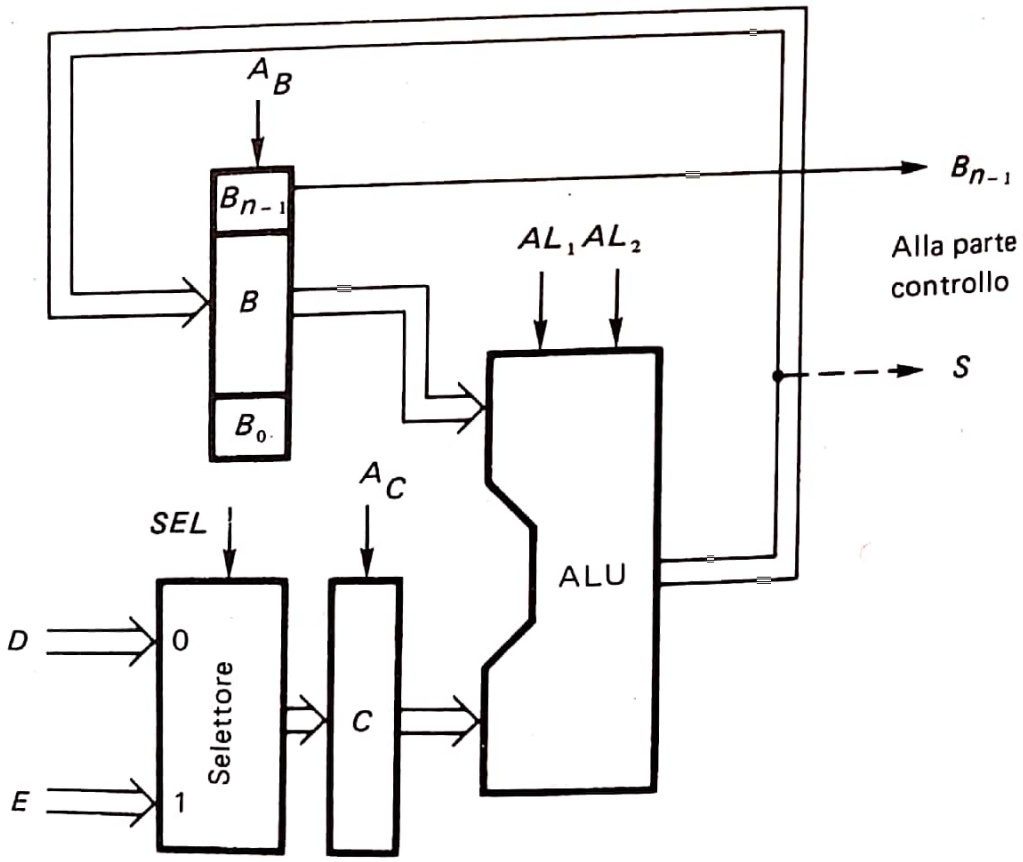


Figura 10.6
Rete logica per eseguire la μsequenza 4.

A proposito di questa traduzione notiamo che nelle prime due μistruzioni i valori di AL_1 e AL_2 non sono specificati, in quanto il risultato prodotto dalla ALU non è utilizzato in questi passi (nello schema di fig. 10.2 tale risultato può unicamente essere trasferito in A, ma questo registro è bloccato: $A_A = 0$). Notiamo inoltre che la seconda μistruzione realizza l'operazione ϕ di congelamento dei registri (quindi anche il segnale K_N può essere non specificato), mentre il segnale di condizione OR(N) è automaticamente approntato dalla parte operativa.

Consideriamo ora una nuova μsequenza che include il confronto tra due dati B e C, eseguito attraverso la sottrazione tra tali dati e la verifica del bit di segno del risultato (0 indica $B - C \geqslant 0$, 1 indica $B - C < 0$).

μsequenza 4:

1: $B - C \rightarrow B$;
2: **if** B_{n-1} = '0' **then** $D \rightarrow C$ **else** $E \rightarrow C$ **fi**

(Nota: B_{n-1} è il bit di segno di B.)

Questa μsequenza può essere eseguita dalla rete della figura 10.6, che, guidata dagli opportuni segnali di controllo, attua le seguenti μistruzioni (i comandi della ALU sono quelli di fig. 10.5):

controllo	μistruzioni				
	A_B	A_C	SEL	AL_1	AL_2
1:	1	0	–	1	1
2: **if** B_{n-1} = '0' **then**	0	1	0	–	–
else	0	1	1	–	–
fi					

La μsequenza 4 impone che il confronto tra B e C sia "preparato" in un intervallo τ, ove si genera e si memorizza (in B) la differenza $B - C$; e che le azioni conseguenti al confronto siano eseguite nell'intervallo τ successivo, ove la condizione è ridotta al test su un bit già pronto (B_{n-1}). Consideriamo ora la seguente μsequenza, con cui si vuole ottenere lo stesso risultato in un singolo intervallo τ (cioè con una sola μoperazione):

μsequenza 5:

if $B \geqslant C$ **then** $D \rightarrow C$ **else** $E \rightarrow C$ **fi**

Questa μsequenza è semanticamente corretta se la condizione può essere preparata, e verificata, nello stesso intervallo τ in cui si esegue l'azione conseguente a **then** o **else**. A tale scopo la parte controllo invia alla ALU i comandi di sottrazione ($AL_1 = AL_2 = 1$), e utilizza come segnale di condizione il primo bit S dell'uscita della ALU (collegamento tratteggiato in fig. 10.6); in risposta al valore di S, la parte controllo emette poi gli ulteriori comandi necessari a completare la frase if. Perché la μsequenza 5 sia accettabile, *tutte le operazioni dette devono poter avvenire in sequenza in un singolo intervallo* σ, tra due successivi impulsi di sincronismo.

Più precisamente si hanno le μistruzioni:

	A_B	A_C	SEL	AL_1	AL_2
if S = '0' **then**	0	1	0	1	1
else	0	1	1	1	1
fi					

che differiscono dalle μistruzioni della μsequenza 4 perché i comandi $AL_1 = AL_2 = 1$ sono generati assieme ai comandi per **then** o **else**. Il funzionamento del sistema è ora più complesso: all'inizio del periodo σ il valore di S, residuo di situazioni precedenti, può essere qualsiasi, ma la μistruzione contiene in ogni caso $AL_1 = AL_2 = 1$, e impone comunque che sia eseguita la sottrazione; al termine di questa operazione il segnale S ha assunto stabilmente il suo nuovo valore, e la parte controllo produce i valori definitivi di A_B, A_C e SEL. Solo a questo punto può intervenire l'impulso, che impone il trasferimento tra registri.

Un funzionamento così compatto è possibile se i moduli combinatori (in particolare la ALU) sono sufficientemente veloci rispetto alla cadenza dell'impulso, ma è in genere sconsigliabile. Infatti l'anello che si forma tra parte operativa e parte controllo, attraverso i segnali AL_1, AL_2 e S della figura 10.6, non include alcun registro; ciò richiede un funzionamento asincrono sull'anello, che deve essere studiato molto attentamente perché non sorgano i problemi citati nel paragrafo 4.4 (in particolare la μsequenza 5 funziona correttamente).

Concludendo, se l'interpretazione della condizione di una frase if richiede un'elaborazione comandata dalla parte controllo, è opportuno eseguire tale elaborazione in un intervallo preparatorio τ, ove la condizione si trasforma in un valore binario, memorizzato in un flip-flop all'arrivo dell'impulso (nella μsequenza 4, il flip-flop B_{n-1} del registro

B); la frase if si esegue quindi nell'intervallo successivo, guidata dal contenuto del flip-flop. Nel caso che sia sconsigliabile alterare il contenuto dei flip-flop della parte operativa, il flip-flop di condizione può essere aggiunto come pezzo necessario al controllo. (Si noti per esempio che la μsequenza 4 altera il contenuto del registro B, per rendere disponibile il bit di condizione in B_{n-1}; la μsequenza 5 non altera tale contenuto.)