Capitolo 7

SISTEMI DI RETI SEQUENZIALI

7.1 - Introduzione

Con il termine **ASM (**Algorithmic State Machine**)** si indica un sistema digitale in grado di realizzare uno o più processi di calcolo (algoritmi) attraverso l'esecuzione di appropriate sequenze di operazioni che manipolano dati di ingresso (istanze dell'informazione da elaborare) per produrre nuovi dati in uscita (istanze dell'informazione risultante dall'elaborazione). Nel lessico comune ci si riferisce ad un sistema di questo genere con il termine di **processore**.

Le operazioni sono caratteristiche di ogni sistema (**operazioni del sistema**) e in genere sono indipendenti tra di loro, per cui le sequenze che si possono formare con esse sono virtualmente illimitate. Poichè un sistema non può decidere autonomamente quali operazioni eseguire ed in quale ordine, occorre fornire dall'esterno, oltre ai dati, anche opportune **direttive** attraverso ingressi specifici per questo tipo di informazione, che diremo di **controllo**. Le direttive sono fornite sotto forma di **codici di operazione,** *OP.* (Fig. 7.1)

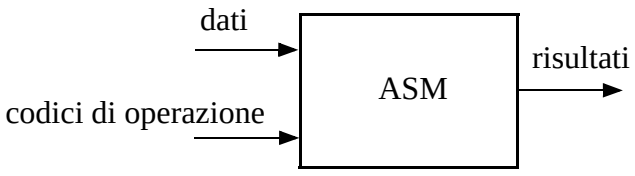


Fig. 7.1

Nel seguito ci proponiamo di esaminare i sistemi schematizzati secondo il modello di Fig. 7.1 dal punto di vista strutturale e dal punto di vista del funzionamento.

Per quanto riguarda il primo punto, poichè qualsiasi struttura logica è formata esclusivamente da reti combinatorie e sequenziali (registri), anche un'ASM deve essere costituita interconnettendo logica combinatoria e registri. Più precisamente, si tratta di un **sistema di due reti sequenziali** connesse in ciclo.

Le due reti componenti sono in genere di tipo sincronizzato e per semplicità di trattazione assumeremo che tutti i registri siano scritti simultaneamente da un unico segnale di clock, il cui periodo definisce il **tempo di ciclo** T del sistema (Fig. 7.2) ed è la somma del **tempo elementare** T_S, ossia l'intervallo tra due impulsi consecutivi durante il quale gli elementi di registro mantengono lo stato (*latching*), e della durata ΔM dell'impulso, durante il quale gli elementi di registro cambiano stato (*enabling*).

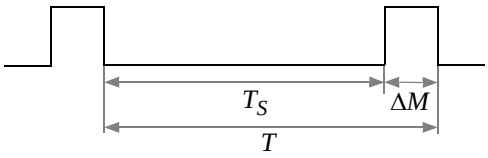


Fig. 7.2

In realtà la restrizione ad un unico segnale di clock, che tra l'altro implica l'uso di elementi di registro del tipo edge-triggered, non è assoluta, ma esistono anche sistemi per i quali viene utilizzato un **clock a più fasi**, ossia un insieme $\{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ di $k \geq 2$ segnali impulsivi periodici o **fasi**, dove φ_i è la i-esima fase, aventi tutti lo stesso periodo T. In ogni fase si distinguono ancora i due intervalli, l'uno attivo ΔM e l'altro passivo T_S (Fig. 7.3). Il numero di fasi deve soddisfare la disuguaglianza $k \cdot \Delta M \leq T$.

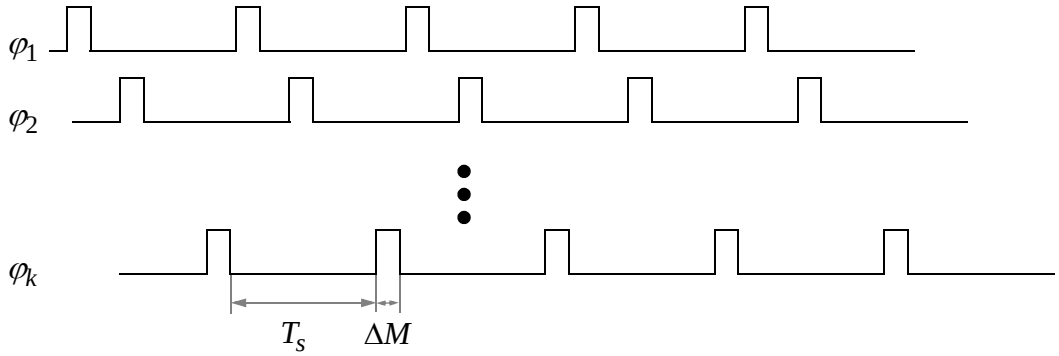


Fig. 7.3

Invece riguardo alle caratteristiche logico-funzionali, si distingue tra sistemi progettati per risolvere un problema o una classe limitata di problemi simili (**sistemi special purpose**) e sistemi previsti per una varietà virtualmente illimitata di problemi (**sistemi general purpose**).

I primi sono caratterizzati completamente dagli algoritmi, in genere in numero abbastanza piccolo, la cui descrizione pertanto costituisce la descrizione del sistema stesso; questa viene specificata in modo da evidenziare tanto il flusso dei dati quanto quello dei controlli e la loro interazione. A sua volta un'ASM speciale è la realizzazione di un algoritmo sotto forma di una struttura digitale specializzata.

Invece in un sistema general purpose gli algoritmi non sono implementati in strutture fisse, bensì sono organizzati sotto forma di liste di *direttive* o *ordini*, note come *programmi*, le quali determinano l'esecuzione di appropriate sequenze di operazioni del sistema.

Risolvere problemi mediante programmi fatti eseguire ad un sistema general purpose significa seguire un approccio **software**, mentre il ricorso ad un sistema speciale costituisce una soluzione **hardware**. La soluzione software ha doti superiori di flessibilità e di economicità, essendo il costo totale ripartito su un numero elevatissimo di applicazioni, contro la maggiore velocità della soluzione hardware.

Nel seguito esaminiamo come vengono eseguite le operazioni del sistema.

7.2 - Linguaggio di trasferimento tra registri

Un'operazione consiste nell'eseguire una successione di azioni elementari dette **micro-operazioni** (μ-operazioni), ciascuna delle quali dura un tempo di ciclo e ha l'effetto di sostituire il contenuto di un registro o di più registri con il risultato prodotto da reti combinatorie o con il contenuto di altri registri o con dati esterni. Il cambiamento del contenuto (stato) di un registro viene specificato, in un linguaggio speciale per la descrizione dell'hardware o RTL, mediante una **relazione di trasferimento**, basata su un operatore di trasferimento, la cui sintassi è la seguente:

<identificatore di registro> ← <contenuto di registro> | <espressione> | <dato esterno>| <costante>

L'**identificatore di registro** è un nome simbolico con il quale si fa riferimento ad un registro, mentre il contenuto di un registro è specificato dal suo identificatore racchiuso tra parentesi quadrate: per esempio [R] significa il contenuto del registro R; inoltre, quando è richiesto, si può fare riferimento al valore del singolo bit di un registro, facendo seguire il suo identificatore da un pedice numerico, oppure a gruppi di bit facendo seguire l'identificatore del registro dagli estremi dell'intervallo di bit considerati tra parentesi angolate; per esempio $R_{<0,4>}$ indica i cinque bit meno significativi del registro R. In taluni casi può essere utile adoperare un registro con scopi di indirizzamento di locazioni di memoria o di altri registri: in questi casi con relazioni di trasferimento del tipo

$$M[R] \leftarrow [A]$$

si intenderà che la locazione della memoria M indirizzata mediante il contenuto del registro R viene scritta con il contenuto del registro A, oppure con la relazione

$$A \leftarrow R5[R1]$$

si intenderà che il registro A viene aggiornato con il contenuto del registro R5, selezionato

mediante R1.

Una μ-operazione può essere descritta formalmente da una lista di relazioni di trasferimento che usano l'operatore di trasferimento sopra definito e che si ammette siano eseguite tutte simultaneamente.

Per esempio la μ-operazione descritta a parole come "trasferisci nel registro A la somma del suo contenuto e del dato esterno D e trasferisci nel registro B il contenuto iniziale di A incrementato di 1" viene descritta nel linguaggio dalla seguente lista di relazioni di trasferimento, nella seconda delle quali [A] rappresenta correttamente il valore iniziale di A in virtù della simultaneità delle esecuzioni:

$$A \leftarrow [A]+D,\ B \leftarrow [A]+1$$

L'esecuzione della lista di relazioni di trasferimento che forma una μ-operazione avviene nel tempo di ciclo del sistema in due passi: a) durante l'intervallo T_S le parti combinatorie del sistema calcolano i valori delle espressioni scritte a destra dell'operatore di trasferimento; b) nell'intervallo ΔM in cui è presente l'impulso di sincronizzazione tali valori vengono scritti nei registri specificati a sinistra dell'operatore di trasferimento. Poichè tutti i registri sono impulsati insieme, è garantita la simultaneità dei trasferimenti. Inoltre dal momento che lo stesso designatore di registro può figurare a destra e a sinistra dell'operatore di trasferimento, è necessario garantire che l'uscita dei registri non cambi prima che l'impulso di sincronizzazione sia rimosso, in modo che sia realizzata una effettiva separazione tra lo stato attuale (da cui dipende il valore dell'espressione nella parte destra della relazione di trasferimento) e lo stato successivo. Questo richiede che i registri siano realizzati mediante dispositivi di memoria non trasparenti (flip-flop master-slave).

Nel linguaggio RTL sopra descritto le espressioni coinvolgono registri, dati esterni, costanti numeriche e/o logiche, legati insieme da operatori logico-aritmetici, senza peraltro entrare nel merito della struttura delle reti che realizzano tali operatori. In altri termini il linguaggio non scende ai livelli di dettaglio propri dell'algebra della commutazione, ma rimane al livello dei componenti combinatori e sequenziali che sono stati definiti nella prima parte del corso come componenti «modulari» (Decodificatori, Selettori, ROM, PLA, Addizionatori, Moltiplicatori, Registri, Contatori, RAM, ecc.).

7.3 - Il diagramma di flusso

Un algoritmo, sia che venga tradotto in un programma, sia che venga realizzato in hardware, è descrivibile mediante un **diagramma di flusso**, un grafo orientato sul quale è possibile seguire il flusso dei dati ed il flusso dei controlli, utilizzando due tipi di nodi, detti rispettivamente **blocchi operativi** e **blocchi di decisione**.

Il diagramma di flusso, o diagramma ASM, è in grado tanto di evidenziare la struttura logica del sistema quanto di esprimere in maniera precisa il concetto di sequenza di intervalli

di tempo. In altre parole esso rappresenta compiutamente il comportamento sia del circuito (combinatorio o sequenziale) che esegue operazioni sui dati, sia di quello che controlla le operazioni secondo le direttive previste dall'algoritmo.

La costruzione del diagramma di flusso costituisce il punto di partenza del progetto di un sistema e deve essere condotta con molta attenzione, cercando di evitare errori concettuali e ridondanze inutili che influenzerebbero negativamente la struttura da progettare e le sue prestazioni. La difficoltà principale Lconsiste nel definire un algoritmo per il problema in oggetto e nel formalizzarlo partendo dalla sua descrizione a parole. Invece la traduzione dell'algoritmo nel diagramma di flusso è un'operazione più semplice, se ci si attiene alle seguenti regole.

Prima di tutto si osservi che un blocco operativo, indicato da un rettangolo, rappresenta sia una μ-**operazione**, descritta come lista di relazioni di trasferimento simultanee e avente la durata di un tempo di ciclo, sia uno **stato** della ASM; un diagramma di flusso ASM descrive pertanto un algoritmo in modo tale che, dato lo stato attuale, lo stato successivo è determinato in modo non ambiguo dalle variabili di stato. Per esempio un algoritmo puramente sequenziale potrebbe essere descritto dal seguente diagramma (Fig. 7.4):

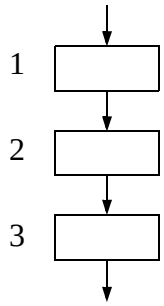


Fig. 7.4

cui corrisponde la ripartizione dell'asse dei tempi tra i vari stati come segue:

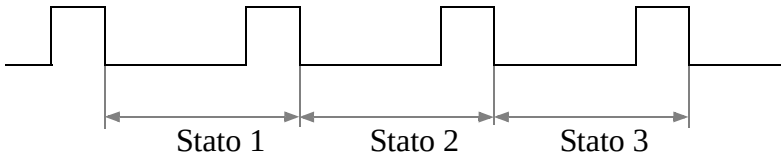


Fig. 7.5

In genere, per comodità, in un blocco operativo non viene scritta per esteso la μ-operazione che esso rappresenta, ma il suo nome simbolico seguito dall'elenco dei registri destinatari dei trasferimenti che essa implica, per esempio $O_i/R_1, R_2, \ldots, R_k$, mentre la lista delle relazioni di trasferimento viene scritta a parte.

Un blocco di decisione, che corrisponde allo statement condizionale **if** *c* **then** e_1 **else** e_2 oppure **if** *c* **then** *e* di un linguaggio ad alto livello, è il tipo di nodo che consente di realizzare un salto condizionato e di uscire dalla stretta sequenzialità; in riferimento alla ASM, esso

realizza la situazione in cui lo stato successivo è determinato non soltanto dallo stato attuale, ma anche dal valore di una o più variabili di test di ingresso. Un blocco di decisione è rappresentato con un nodo a forma di rombo che contiene un'espressione logica, in generale funzione del contenuto dei registri R_1, R_2, …, R_k del sistema, del codice di operazione *OP* e dei dati esterni *D*, nella forma $f(R_1, R_2, \ldots, R_k, OP, \ldots, D)$; il valore *vero* (*T*) o *falso* (*F*) di tale espressione determina la selezione di uno dei due punti di uscita dal blocco.

Per esempio la seguente figura mostra un frammento di diagramma di flusso nel quale viene calcolato in un registro B il valore assoluto di un numero in complemento a due contenuto in un registro A:

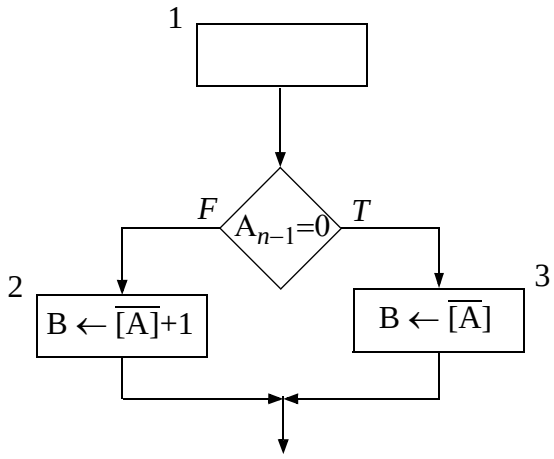


Fig. 7.6

In questo caso la ripartizione del tempo tra i vari stati della macchina è la seguente:

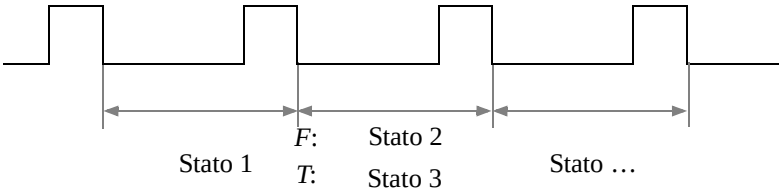


Fig. 7.7

Notare che un test non ha bisogno di un tempo di ciclo proprio, ma esso avviene in parallelo alla μ-operazione del blocco operativo che lo precede: vedremo in seguito che questo fatto ha forti conseguenze sulla struttura del sistema.

In alternativa a quanto indicato nell'esempio di Fig. 7.6, un blocco di decisione può anche essere contrassegnato con una variabile logica *c*, scrivendo separatamente l'espressione che ne definisce il valore in una relazione di traferimento del tipo $c \leftarrow f(R_1, R_2, \ldots, R_k, OP, \ldots, D)$; in questo caso la variabile *c* è realizzata mediante un registro che riceve in ingresso l'uscita di una rete combinatoria che calcola l'espressione. Invece nel caso dell'esempio di Fig. 7.6 è l'uscita stessa delle rete combinatoria oggetto del test: questi due diversi approcci hanno un'importanza fondamentale nel definire la classe di sistemi che ne deriva, come si vedrà nel prossimo

paragrafo.

Può accadere spesso che si realizzi una molteplicità di scelte, a seguito delle quali devono essere eseguite μ-operazioni differenti; nel diagramma ciò si traduce nella presenza di un albero di decisione con un ingresso e più uscite, nel quale ogni scelta binaria è determinata dal valore di una variabile logica, secondo lo schema illustrato nella Fig. 7.8a.

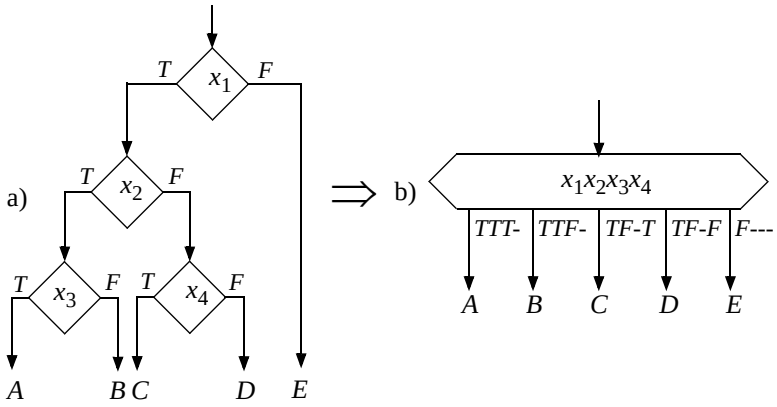


Fig. 7.8

In questi casi l'albero di decisione può essere sostituito da un **blocco di decisione complesso** (che in un linguaggio al alto livello corrisponde allo statement **case**), nel quale la condizione risulta costituita dal prodotto logico delle singole condizioni e le uscite, in numero pari alle foglie dell'albero, sono contrassegnate da una stringa di valori binari che codificano globalmente le scelte (Fig. 7.8b).

Un esempio significativo di diagramma ASM è quello della moltiplicazione binaria; come sappiamo, dati due numeri naturali X ed Y di n bit, un algoritmo per calcolarne il prodotto P fa uso di somme e traslazioni, ed è basato sulla relazione:

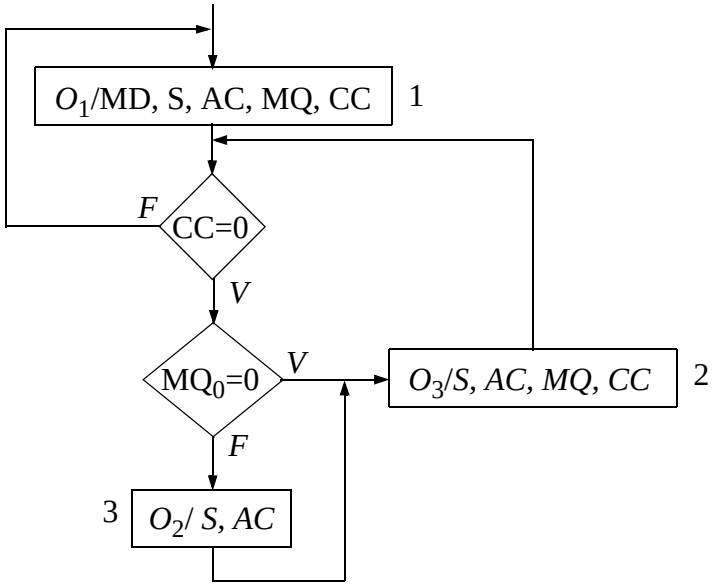
$$P = \sum_{j=0}^{n-1} Y \cdot x_j \cdot 2^j$$

Essendo $x_j \in \{0,1\}$, i termini $Y \cdot x_j \cdot 2^j$ sono nulli oppure sono uguali al moltiplicando Y traslato verso sinistra di j posizioni: il prodotto P viene calcolato come somma dei termini non nulli e per ragioni di semplicità possiamo pensare di effettuare le somme in maniera iterativa, partendo da 0 e addizionando ad ogni passo il moltiplicando al risultato parziale traslato di una posizione binaria verso destra; ai fini di ciascuna somma è infatti equivalente traslare il moltiplicando verso sinistra o il risultato parziale verso destra. Il risultato è espresso su $2n$ bit.

La realizzazione hardware dell'algoritmo fa uso di un addizionatore parallelo di n+1 bit e di tre registri di n bit, di cui uno, denominato MD, contiene il moltiplicando e gli altri due, AC ed MQ, concatenati insieme e dotati di capacità di shift destro, all'inizio dell'operazione contengono rispettivamente 0 ed il moltiplicatore, al termine la metà più significativa e quella meno significativa del prodotto. Inoltre è necessario un registro S da 1 bit che contiene il bit

più significativo di ciascuna somma parziale e che viene traslato nella posizione più significativa di AC.

Il diagramma di flusso e la lista delle μ-operazioni della moltiplicazione sono mostrati in Fig. 7.9.



O_1: MD ← *Y*, S ← 0, AC ← 0, MQ ← *X*, CC ← 4

O_2: AC ← [AC]+[MD], S ← c_n

O_3: S_AC_MQ$_i$ ← S_AC_MQ$_{i+1}$, S ← 0, CC ← [CC]-1

Fig. 7.9

Da questi due elementi è immediato ricavare anche la struttura della Parte Operativa del moltiplicatore, rappresentata nel seguente schema a blocchi:

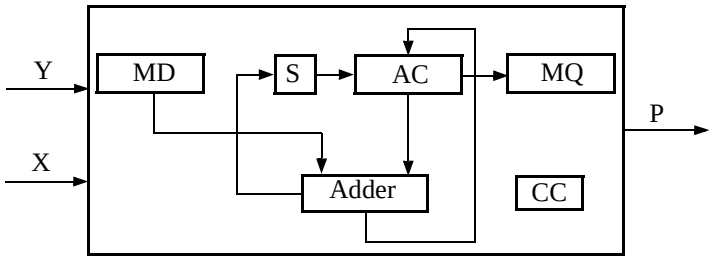


Fig. 7.10

mentre il procedimento di calcolo può essere compreso esaminando il contenuto dei registri ad ogni passo; per esempio nel caso dei fattori X = 13 ≡ 1101 e Y = 14 ≡ 1110, il cui prodotto è P = 182 ≡ 10110110, avremo il seguente schema, nel quale c_n rappresenta il bit di riporto più significativo:

	MD	S	AC	MQ	CC
$MD \leftarrow Y, S \leftarrow 0, AC \leftarrow 0, MQ \leftarrow X, CC \leftarrow 4$	1110	0	0000	1101	4
$MQ_0 = 1$: $AC \leftarrow [AC]+[MD, S \leftarrow c_n$		0	1110	1101	4
$S\_AC\_MQ_i \leftarrow S\_AC\_MQ_{i+1}, S \leftarrow 0, CC \leftarrow [CC]-1$		0	0111	0110	3
$MQ_0=0$: $S\_AC\_MQ_i \leftarrow S\_AC\_MQ_{i+1}, S \leftarrow 0,$ $CC \leftarrow [CC]-1$		0	0011	1011	2
$MQ_0=1$: $AC \leftarrow [AC]+[MD, S \leftarrow c_n$		1	0001	1011	2
$S\_AC\_MQ_i \leftarrow S\_AC\_MQ_{i+1}, S \leftarrow 0, CC \leftarrow [CC]-1$		0	1000	1101	1
$MQ_0=1$: $AC \leftarrow [AC]+[MD, S \leftarrow c_n$		1	0110	1101	1
$S\_AC\_MQ_i \leftarrow S\_AC\_MQ_{i+1}, S \leftarrow 0, CC \leftarrow [CC]-1$		0	1011	0110	0

7.4 - Caratterizzazione delle operazioni

Le operazioni che devono essere eseguite per la risoluzione di un problema o per il calcolo di una funzione sono definite trasmettendo dall'esterno la lista dei rispettivi codici operativi (in un sistema general purpose questa lista costituisce il programma e varia da un problema all'altro; in un sistema speciale la lista è fissa e realizzata in hardware).

Il sistema deve essere in grado di esaminare e riconoscere ogni codice operativo *OP* in ingresso e di ordinare l'esecuzione dell'operazione corrispondente; alla fine dell'operazione corrente deve essere riconosciuto il codice successivo per dare luogo ad un'altra operazione e così via. L'attività che precede l'esecuzione di qualunque operazione viene detta **decodifica** ed è realizzata direttamente nella struttura del sistema.

Le attività relative alla decodifica possono essere descritte nel diagramma di flusso mediante un blocco operativo contrassegnato con una μ-operazione il cui effetto è il trasferimento del codice operativo presente in ingresso al sistema in un registro OR (operation register) e da un blocco di decisione complesso (o in alternativa da un albero di decisione) che, sulla base del contenuto di OR, attiva l'esecuzione dell'operazione appropriata. In alternativa si può fare a meno di memorizzare il codice operativo, dal momento che esso deve comunque rimanere presente in ingresso per almeno un tempo di ciclo sia dal momento dell'avvio del sistema, sia dal termine di ogni operazione. Abbiamo così due schemi possibili, illustrati in Fig. 7.11, nell'ipotesi semplificativa che il sistema esegua due sole operazioni.

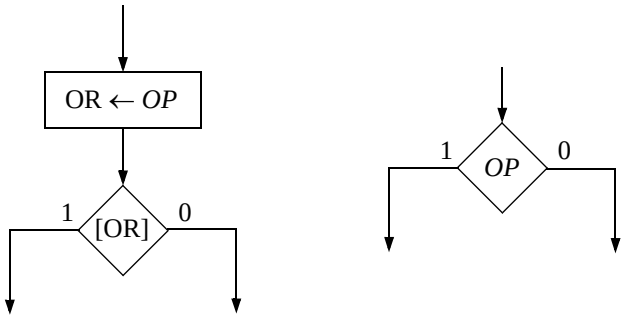


Fig. 7.11

Le attività che complessivamente hanno luogo dalla successione delle singole operazioni che il sistema può eseguire dal momento in cui entra in attività al momento in cui si arresta consistono nell'alternanza di una fase di decodifica e di una fase di esecuzione di una delle operazioni dirette dal codice operativo.

Perciò il diagramma di flusso complessivo del sistema si ottiene facendo seguire ai blocchi relativi alla decodifica quelli delle singole operazioni istruite dall'esterno in base all'algoritmo da eseguire e stabilendo un arco di richiusura dall'uscita di ciascuna operazione all'ingresso del diagramma di flusso (Fig. 7.12).

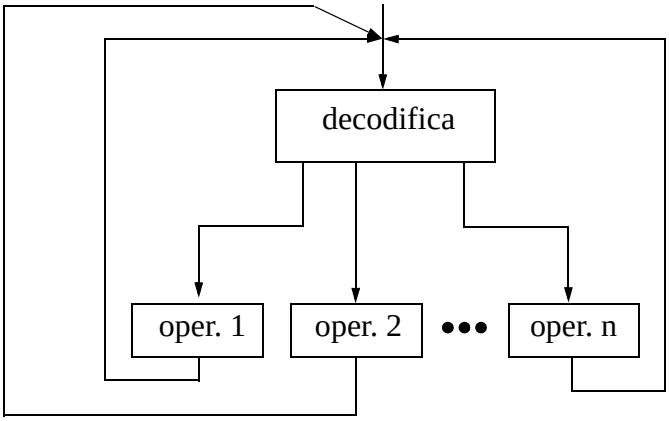


Fig. 7.12

Un diagramma di flusso ASM ben strutturato deve rispettare alcuni vincoli:

1) esiste un solo ingresso da cui inizia il funzionamento e nessuna uscita; il sistema è sincronizzato con l'esterno in modo che quando viene eseguita un'operazione, il nuovo codice e i dati siano disponibili stabilmente in ingresso;

2) non possono esistere catene cicliche formate solo da blocchi di decisione: un insieme interconnesso di blocchi di decisione deve avere necessariamente una struttura aperta ad albero, oppure il ciclo deve essere interrotto con un blocco operativo non condizionale caratterizzato dalla μ-operazione nulla O_0 (No-operation, o NOP), che lasciando inalterato il contenuto dei registri, ha il compito di dare un significato operativo al tempo di ciclo e di

rendere definito lo stato il cui si porta la ASM per effetto della condizione complessa che risulta vera;

3) se in un ciclo è presente come unico blocco operativo un blocco caratterizzato dalla μ-operazione nulla O_0, deve esistere nel ciclo almeno un blocco di decisione la cui condizione è funzione di informazioni esterne al ciclo, tipicamente di registri modificati da μ-operazioni esterne al ciclo.

La condizione 2) è soddisfatta per l'intero diagramma se lo è per ogni singola operazione e se la decodifica avviene con la memorizzazione del codice operativo. Se invece il codice operativo non viene memorizzato, ossia manca un registro che separi le entrate dalle uscite, ritardando queste ultime di un tempo di ciclotempo di ciclo, possono sorgere cicli, anche se non sono presenti nei diagrammi delle singole operazioni, quando l'uscita di un singolo diagramma è collegata all'ingresso del diagramma totale (Fig. 7.13); ciò può rendere non definito lo stato della ASM al termine dell'operazione corrente.

In questa situazione può trovarsi anche la sola porzione di diagramma di flusso relativa alla decodifica, qualora a certi codici operativi non corrisponda alcuna operazione, come indicato in figura; per esempio se un sistema ha 6 operazioni, occorrono tre bit per la loro codifica e pertanto a due codici non corrisponde alcuna operazione. Anche in questi casi occorre rompere il ciclo inserendovi il blocco operativo della μ-operazione nulla.

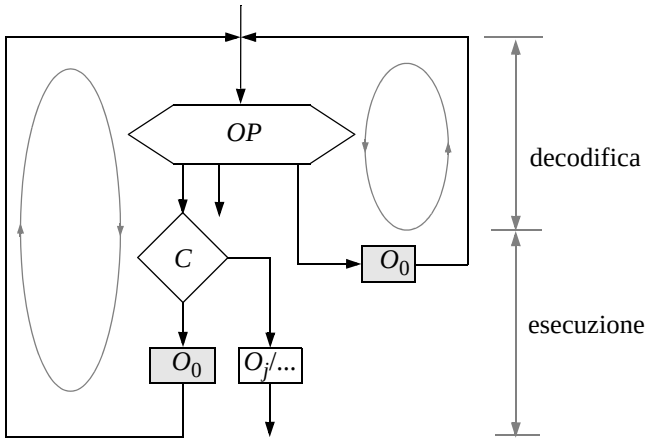


Fig. 7.13

La condizione 3) infine è richiesta per impedire che il sistema rimanga permanentemente bloccato in ciclo, senza poterne uscire (Fig. 7.14):

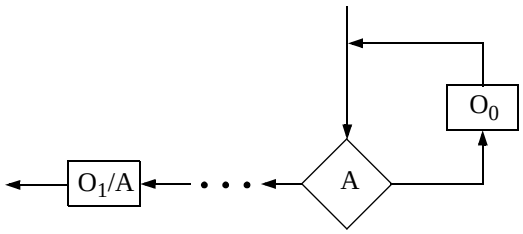


Fig. 7.14

7.5 - Modello generale del sistema

La separazione logica tra flusso dei dati e flusso dei controlli e la loro interazione, che sono stati individuati a livello del diagramma di flusso, si riflette nella struttura realizzativa del sistema e giustifica l'affermazione, fatta nel paragrafo 7.1, che un'ASM è un sistema formato da due reti sequenziali interconnesse. Infatti esso può essere diviso in due parti: una **Parte Operativa** (PO) che esegue le operazioni sui dati, mediante una sequenza di μ-operazioni; una **Parte di Controllo** (PC) che comanda e dirige l'esecuzione delle operazioni. Questa distinzione comporta che l'intero sistema debba essere formato da due macchine sequenziali connesse in ciclo, secondo lo schema di Fig. 7.15.

Le connessioni tra le due parti sono di due tipi: un tipo è costituito da un insieme di **variabili binarie di controllo** $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ prodotte in uscita da PC e trasferite in ingresso a PO; esse codificano le μ-operazioni O_j, $j = 1, \ldots, p$ che devono essere eseguite da PO in ciascun tempo di ciclo. I nomi delle μ-operazioni coincidono con i codici, in genere non binari, $\alpha_0^* \alpha_1^* \ldots \alpha_{m-1}^*$ che le comandano; tra questi viene annoverato anche il codice O_0 associato alla μ-operazione nulla.

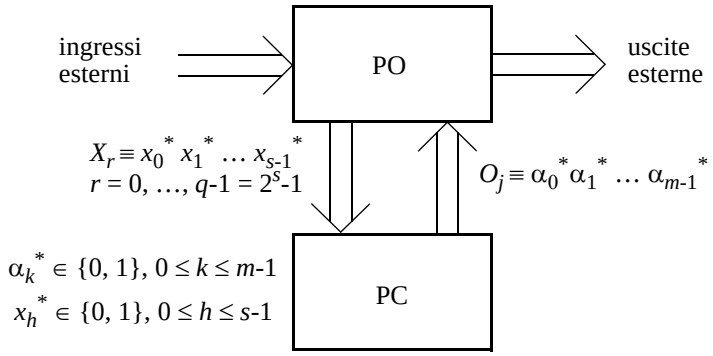


Fig. 7.15

L'altro tipo è un insieme di **variabili binarie di condizione** $x_0, x_1, \ldots, x_{s-1}$ che sono trasmesse come uscita da PO a PC e definiscono il codice binario X_r, $r = 1, \ldots, q = 2^s$ delle condizioni logiche in cui può trovarsi PO, il quale insieme allo stato interno di PC determina la scelta del codice operativo della prossima operazione da eseguire, nonchè lo stato successivo di PC stessa.

La parte di controllo PC ha dunque una doppia funzione:

1) generare il codice operativo della μ-operazione che deve essere eseguita da PO in ogni ttempo di ciclo;

2) decidere la sequenza delle μ-operazioni che realizzano ogni operazione. Questa sequenza dipende in ogni periodo dalle variabili di condizione e poichè queste sono funzione anche del contenuto dei registri di PO dopo ogni μ-operazione, la scelta operata da PC è influenzata dall'esito dell'ultima μ-operazione eseguita da PO.

Le informazioni esterne (dati e codici operativi) entrano in PO dagli ingressi esterni e i

risultati dell'elaborazione (contenuti in registri di PO) sono resi disponibili attraverso le uscite esterne. I codici operativi, se non sono memorizzati nel registro OR di PO, attraversano PO senza modifiche e si presentano all'ingresso di PC codificati da un sottoinsieme delle variabili di condizione. In questo schema si noti una differenza, peraltro inessenziale, rispetto a quello di Fig. 7.1

Inoltre, in riferimento ai due modelli di macchine sequenziali e al fatto che in un sistema di reti sequenziali non possono esistere cicli di informazione diretti, cioè che non si chiudono attraverso registri (il che equivale a dire che in una connessione chiusa non tutte le reti possono essere di Mealy), si può osservare che:

1) PO è sempre una rete sequenziale, perchè si è supposto che contenga almeno un registro. In particolare se dati e codici operativi sono memorizzati in registri e nessuna variabile di condizione dipende dalle variabili di controllo fornite da PC, PO è una macchina di Moore, in quanto le sue uscite esterne e quelle di condizione dipendono solo dal contenuto di registri, ossia dallo stato interno.

2) PO e PC formano un sistema costituito da un ciclo chiuso di due reti sequenziali sincronizzate dallo stesso impulso. Pertanto se PO è di Moore, PC può essere indifferentemente di Moore o di Mealy. Viceversa, se esiste almeno una variabile di condizione che dipende dalle variabili di controllo, per cui PO è una macchina di Mealy, allora PC deve essere di Moore perchè altrimenti si avrebbe un ciclo di informazione chiuso attraverso le parti combinatorie delle due reti che renderebbe instabile il funzionamento del sistema. Questa situazione si verifica quando l'uscita delle reti combinatorie di PO che calcolano i membri destri delle relazioni di trasferimento delle μ-operazioni (addizionatori, moltiplicatori, comparatori, ecc.) viene inviata direttamente in ingresso a PC.

Possiamo dunque individuare tre classi di sistemi, a seconda del tipo di macchina sequenziale su cui sono modellate PO e PC: sistemi Moore-Mealy, sistemi Moore-Moore e sistemi Mealy-Moore.

I sistemi Mealy-Mealy, privi di variabili di condizione dipendenti da variabili di controllo, quindi non soggetti a instabilità a causa di cammini chiusi di informazione tra PO e PC non interrotti da registri, ma dipendenti solo da ingressi esterni, sono assimilabili di fatto a sistemi Moore-Mealy.

Infine PC può essere in certi casi realizzata come una rete combinatoria, da considerarsi come caso degenere di macchina di Mealy ad un solo stato.

7.6 - Realizzazione della Parte Operativa

La parte operativa può essere realizzata in base alla lista delle μ-operazioni e ai moduli logici disponibili. Questo approccio è per lo più seguito per realizzare sistemi special-purpose, nei quali un requisito importante è la velocità di risposta, e porta all'organizzazione generale mostrata in Fig. 7.16a. Invece nei sistemi general-purpose la parte operativa è costituita per lo

più da un componente detto **ALU** (Arithmetic Logic Unit) che racchiude i moduli logici necessari per le operazioni del sistema e da un componente che è formato da un insieme di registri identici (Local Store) utilizzati per memorizzare operandi, risultati parziali, indirizzi, ecc. I moduli dell'ALU ed i registri della Local Store richiesti da ogni operazione sono riferiti attraverso opportuni segnali di controllo e di selezione. Dati e controlli sono scambiati tra ALU, registri e mondo esterno mediante un sistema di comunicazione detto **bus** (Fig. 7.16b).

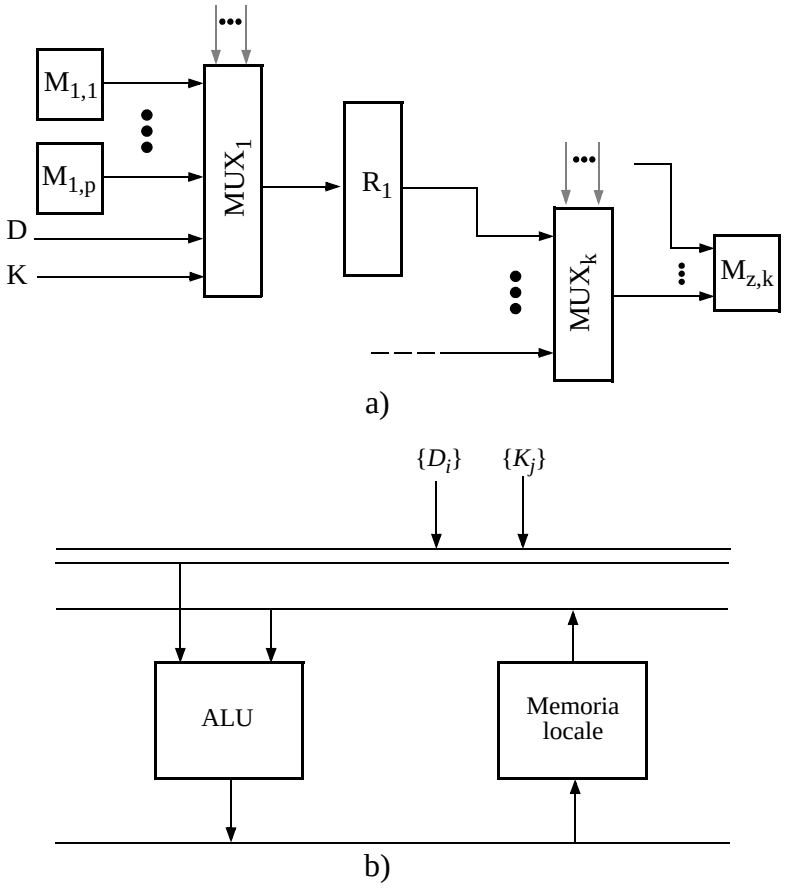


Fig. 7.16

Nel seguito ci occuperemo prevalentemente del progetto della PO di sistemi speciali.

Per ogni μ-operazione si considera l'espressione che figura a destra in ciascuna relazione di trasferimento e si individua il modulo logico (addizionatore, moltiplicatore, ecc.) necessario per il calcolo di tale espressione, a meno che essa non si riduca ad una semplice costante o ad un dato esterno da scrivere in un registro.

Al fine di ridurre ingombro, probabilità di guasti, costo del sistema è opportuno determinare il numero di moduli differenti strettamente necessari per la realizzazione di PO; per questo ci si basa sulle μ-operazioni che richiedono il massimo numero di esemplari per ciascun tipo di modulo contemporaneamente, dato che le μ-operazioni sono eseguite in tempi di ciclo diversi ed un modulo impegnato in una μ-operazione può essere utilizzato in cicli diversi per altre.

Tuttavia non sempre è possibile ridurre il numero dei moduli, e ciò dipende strettamente dalla forma delle μ-operazioni. In particolare quando una μ-operazione consiste in più relazioni di trasferimento e in una (o più) di esse è richiesto un modulo la cui funzionalità è identica a quella di altri moduli designati per la stessa μ-operazione, esso non può essere sostituito con uno di quelli, poichè le singole relazioni di trasferimento sono simultanee. Per esempio nel seguente caso:

$$O_1: A \leftarrow [A]+[B], B \leftarrow [A]+1$$

la μ-operazione richiede due addizionatori che non possono essere ridotti ad uno soltanto a causa della simultaneità dei trasferimenti.

I criteri di riduzione del numero dei moduli logici hanno avuto importanza soprattutto nel passato, quando i sistemi digitali erano realizzati con componenti discreti a tecnologia SSI (porte logiche) e/o MSI (moduli combinatori e sequenziali). Con l'avvento delle tecnologie a larghissima scala di integrazione (VLSI, ULSI), questo approccio ha perduto gran parte della sua importanza, in quanto i criteri di ottimo di un progetto si sono spostati a considerare altri parametri direttamente dipendenti dalla tecnologia di fabbricazione dei componenti e che tengono conto della loro assai più grande complessità logica.

Oltre ai moduli logici, la parte operativa contiene un numero di registri pari al numero degli identificatori diversi che figurano a sinistra nelle relazioni di trasferimento. Infine tra moduli operativi e registri e tra registri e moduli operativi sono necessari dei multiplexer il cui numero e le cui caratteristiche, in riferimento alla Fig. 7.16a, possono essere definiti con i seguenti criteri.

a) Multiplexer all'ingresso dei registri

Per determinare il numero e le caratteristiche dei multiplexer tra moduli operativi e registri, come MUX_1 nella Fig. 7.16a, si raggruppano tutte le relazioni che trasferiscono nello stesso registro R e per ciascun gruppo si ricava il numero *n* costituito dalla somma dei moduli distinti, dei registri, dei dati esterni e delle costanti trasferiti in R. In questo conto rientrano anche relazioni del tipo R ← T(A), dove T( ) rappresenta l'operazione di traslazione del contenuto di un registro. Se *n* è maggiore di 1, si deve prevedere un multiplexer ad *n* ingressi tra le uscite di certi moduli, dati e costanti ed il registro R. È da notare che le variabili logiche possono essere ottenute sotto forma di variabili di controllo trasmesse da PC, mentre le costanti *K* (logiche e/o aritmetiche possono essere cablate in PO, oppure costituire un ingresso esterno.

Una riduzione del numero dei multiplexer e/o del numero dei loro ingressi si può ottenere osservando che spesso i registri figurano in relazioni di trasferimento del tipo R ← 0; in questo caso è inutile prevedere un ingresso aggiuntivo per il multiplexer all'entrata di R o addirittura un multiplexer in ingresso ad R per consentire la scrittura in R anche dello 0, ma basta scegliere un registro dotato di un ingresso asincrono di *clear*. Un punto a sè è la realizzazione delle μ-operazione nulla O_0: finchè è possibile conviene realizzarla inibendo la scrittura dei registri con un'apposito segnale di controllo (*load*), oppure riscrivendoli in se stessi.

Un'altra situazione molto frequente è quella in cui devono essere implementati dei *flag*: in questi casi la scelta cadrà su flip-flop, tipicamente di tipo JK, o in certi casi anche su latch SR; saranno esclusi flip-flop D per i quali è complicato assicurare il mantenimento dello stato, cosa viceversa molto semplice per un JK, applicando 0 agli ingressi *J* e *K*.

Infine è frequente la richiesta della funzione di conteggio (tipicamente per il controllo di un ciclo), la quale figura in relazioni di trasferimento del tipo:

$$\text{COUNT} \leftarrow N$$
$$\text{COUNT} \leftarrow [\text{COUNT}] + 1$$

Apparentemente è coinvolto nella sua realizzazione un registro, il cui contenuto è inizializzato ad un valore *N* (spesso negativo) e modificato per somma di 1 ad ogni passo; potrebbe perciò sembrare necessario in ingresso al registro un multiplexer a due vie attraverso le quali far fluire il valore iniziale e quello modificato ed un addizionatore per l'aggiornamento; in realtà è molto più semplice utilizzare un contatore up o down con caricamento parallelo.

b) Multiplexer all'ingresso dei moduli operazionali

La determinazione dei multiplexer, come MUX_k in figura, tra registri e moduli operazionali viene ottenuta raggruppando per ogni modulo le parti destre di tutte le relazioni di trasferimento che fanno riferimento ad esso, dopo averle normalizzate in modo da definire tutti gli ingressi del modulo. Per ciascun ingresso si raggruppano tutte le variabili (contenuti di registri e/o dati esterni) e le costanti distinte che possono essere applicate a quell'ingresso come operandi.

Per esempio nel caso delle μ-operazioni:

$$O_1: A \leftarrow [A]+[B]$$
$$O_2: A \leftarrow [A]+[B]+1$$
$$O_3: B \leftarrow [A]+1$$

indicando nell'ordine con I_1, I_2, I_3 gli ingressi dell'addizionatore identificato come unico modulo operazionale necessario, ad I_1 risulta associato il contenuto del registro A, a I_2 quello del registro B e la costante 0, a I_3 le costanti 1 e 0: queste ultime si possono raggruppare sotto forma di una sola variabile, scegliendone opportunamente il valore.

Se un gruppo contiene solo una costante o una variabile, all'ingresso corrispondente si applica un valore costante o una variabile di tipo α, a seconda che tale ingresso rimanga costante oppure vari da una μ-operazione all'altra. Invece se un gruppo comprende *n* tra variabili e costanti, tra esse e l'ingresso corrispondente del modulo si deve interporre un multiplexer ad *n* ingressi. Così nell'esempio precedente a I_1 è collegato solo il registro A, a I_2 l'uscita di un multiplexer a due ingressi alimentati da B e dalla costante 1, a I_3 una sola variabile α.

Questo procedimento è sensibile, per quanto riguarda numero e caratteristiche dei multiplexer, alle scelte possibili che si presentano durante la fase di ridefinizione dei moduli

logici e all'ordine con cui si scrivono gli operandi nelle espressioni che figurano nelle relazioni di trasferimento e pertanto non garantisce risultati sempre ottimali.

Per quanto riguarda infine le variabili di condizione, esse possono essere a) uscite di qualche modulo già presente in PO, b) ingressi esterni del sistema (per esempio il codice di operazione *OP*), c) uscite di reti aggiuntive previste espressamente per questo scopo in PO e definite dalle funzioni logiche che corrispondono alle variabili di condizione stesse.

Esempio

Si vuole progettare la parte operativa di un sistema che esegue la moltiplicazione di due numeri relativi *X* e *Y* rappresentati in modulo e segno in un campo di *n* bit e il calcolo del valore assoluto del contenuto di un registro accumulatore AC, rappresentato in complemento a due. La moltiplicazione viene eseguita nel seguente modo. I segni dei fattori vengono preventivamente confrontati per stabilire il segno del risultato che viene mantenuto in un registro fino al termine del calcolo. I moduli dei due fattori, lunghi ciascuno *n*-1 bit, sono moltiplicati secondo l'algoritmo di somme e traslazioni utilizzando, oltre all'accumulatore AC, due registri da *n* bit MD e MQ ed un registro da 1 bit C per il riporto dalla posizione più significativa di AC durante le somme. Inoltre un registro contatore CC serve per controllare il numero dei passi di calcolo che ammontano ad *n*-1. I registri MD ed MQ contengono all'inizio il modulo dei due fattori nelle *n*-1 posizioni meno significative, mentre nel bit di segno è scritto 0 per comodità, essendo l'informazione sul segno già stata elaborata a parte. Al termine del calcolo le *n*-1 posizioni più significative di MQ contengono la metà meno significativa del prodotto (la posizione meno significativa di MQ contiene il bit di segno del fattore inizialmente caricato), mentre le *n*-1 posizioni meno significative di AC contengono la metà più significativa. Un ulteriore passo di calcolo inserisce il segno nella posizione più significativa di AC: il prodotto con il suo segno occupa dunque 2*n*-1 posizioni. Per quanto riguarda l'altra operazione, essa è eseguita semplicemente complementando bit a bit l'accumulatore e sommandovi 1 quando è negativo, lasciandolo inalterato altrimenti.

Il diagramma ASM e la lista delle μ-operazioni sono mostrati in Fig. 7.17.

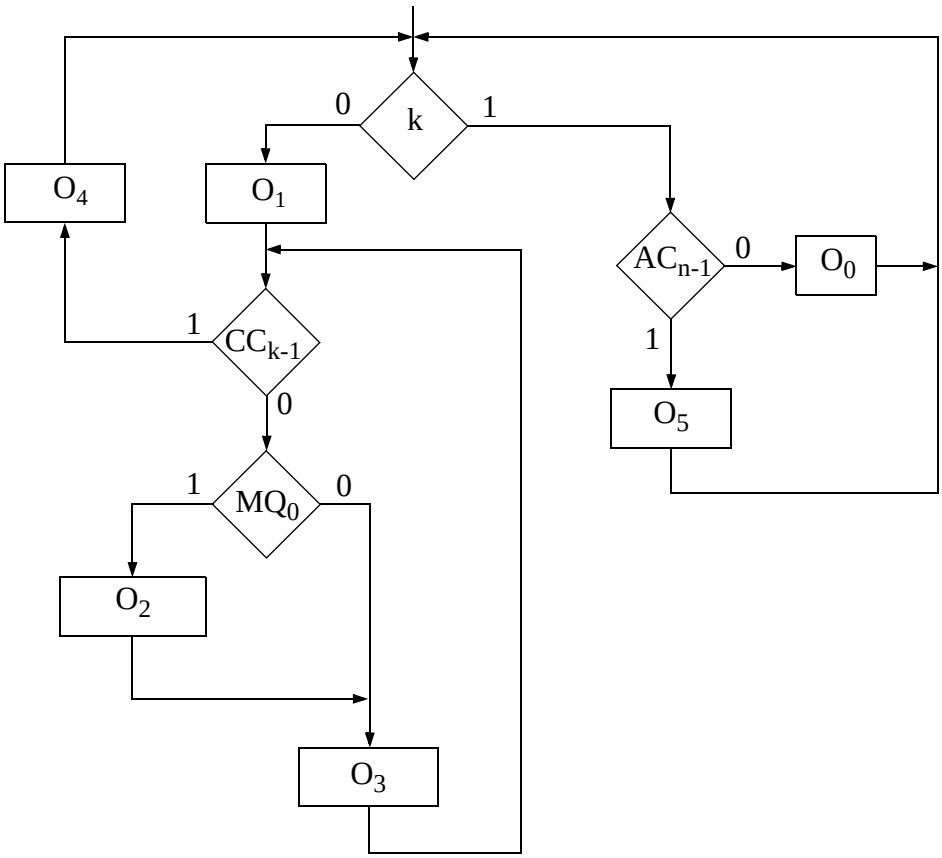
Dalla lista, seguendo il procedimento sopra illustrato, si individuano due moduli operazionali: un addizionatore ed una porta XOR per la determinazione del segno; inoltre sono richiesti i seguenti registri:

- MD, registro parallelo di *n* bit, con un ingresso di controllo *load* (una variabile di controllo α_1).
- MQ, registro di *n* bit con ingressi e uscite paralleli e seriali e possibilità di shift destro: quindi ha bisogno di due segnali di controllo *load* e *shift* (α_2 e α_3).
- AC, registro di *n* bit con ingressi e uscite seriali e paralleli come MQ; AC è coinvolto nelle seguenti relazioni di trasferimento:

$$AC \leftarrow 0$$
$$AC \leftarrow [AC]+[MD]$$

$AC \leftarrow T_d(AC)$, $AC_{n-1} \leftarrow C$
$AC_{n-1} \leftarrow S$
$AC \leftarrow \overline{[AC]}+1$

per realizzare la prima conviene che sia dotato di ingresso *clear* (α_4); inoltre deve possedere ingressi di controllo *load* e *shift* (α_5 e α_6) e si dovrà prevedere sull'ingresso parallelo un multiplexer a due vie (α_7), in modo da poter caricare il risultato dell'addizione (relazioni 2 e 5) oppure il bit del segno, lasciando inalterati tutti gli altri, che di fatto verranno riscritti (relazione 4).



O_1: $MD \leftarrow |Y|$, $MD_{n-1} \leftarrow 0$, $MQ \leftarrow |X|$, $MQ_{n-1} \leftarrow 0$, $S \leftarrow Y_{n-1} \oplus X_{n-1}$, $AC \leftarrow 0$, $C \leftarrow 0$, $CC \leftarrow \overline{n}+2$
O_2: $AC \leftarrow [AC]+[MD]$, $C \leftarrow c_n$
O_3: $AC \leftarrow T_d(AC)$, $AC_{n-1} \leftarrow C$, $C \leftarrow 0$, $MQ \leftarrow T_d(MQ)$, $MQ_{n-1} \leftarrow AC_0$, $CC \leftarrow [CC]+1$
O_4: $AC_{n-1} \leftarrow S$
O_5: $AC \leftarrow \overline{[AC]}+1$

Fig. 7.17

- CC, contatore con preset parallelo (α_8) e abilitazione al conteggio (α_9).
- C può essere realizzato con un flip-flop JK applicando c_n all'ingresso *J* ed una variabile di controllo (α_{10}) all'ingresso *K*: essa varrà 1 quando C deve essere azzerato, altrimenti vale sempre 0.
- S è semplicemente un flip-flop D con un ingresso di abilitazione (α_{11}).

Si identificano quindi i multiplexer tra modulo e registro, raggruppando le relazioni di trasferimento come segue:

$MD \leftarrow \|Y\|$, $MD_{n-1} \leftarrow 0$	nessun multiplexer
$MQ \leftarrow \|X\|$, $MQ_{n-1} \leftarrow 0$	
$MQ \leftarrow T_d(MQ)$, $MQ_{n-1} \leftarrow AC_0$	1 multiplexer a due ingressi
$AC \leftarrow 0$	
$AC \leftarrow AC]+[MD]$	
$AC \leftarrow T_d(AC)$, $AC_{n-1} \leftarrow C$	
$AC_{n-1} \leftarrow S$	
$AC \leftarrow \overline{[AC]}+1$	1 multiplexer a quattro ingressi (in due relazioni AC riceve una somma)
$S \leftarrow Y_{n-1} \oplus X_{n-1}$	nessun multiplexer
$CC \leftarrow \overline{n}+2$	
$CC \leftarrow [CC]+1$	nessun multiplexer
$C \leftarrow 0$	
$C \leftarrow c_n$	nessun multiplexer, per quanto detto sulla sua scelta

Per determinare i multiplexer tra registri e moduli operazionali, c'è da considerare il solo addizionatore il quale, non dovendo provvedere all'aggiornamento di CC, è coinvolto nelle seguenti relazioni di trasferimento:

$$AC \leftarrow [AC]+[MD]+0$$
$$AC \leftarrow [AC]+1+0$$

pertanto il suo ingresso I_1, che riceve [AC] o $\overline{[AC]}$, richiede un multiplexer a due ingressi (α_{12}) e analogamente I_2 (α_{13}), che deve ricevere [MD] oppure 1.

La codifica delle μ-operazioni è mostrata nella seguente tabella. Poichè le colonne (1, 2, 4, 8, 10, 11) e (3, 6, 9) sono identiche e le colonne 5 e 7 possono essere rese uguali rispettivamente alle colonne 13 e 12 togliendo le non specificazioni, di fatto sono sufficienti le variabili di controllo α_1, α_3, α_5, α_7, che sono ridenominate come α_1, α_2, α_3, α_4;

	MD	MQ		AC			MUX_3	CC		C	S	MUX_1	MUX_2
	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}	α_{11}	α_{12}	α_{13}
O_1	1	1	0	1	0	0	–	1	0	1	1	–	0
O_2	0	0	0	0	1	0	1	0	0	0	0	1	–
O_3	0	0	1	0	0	1	–	0	1	0	0	–	–
O_4	0	0	0	0	1	0	0	0	0	0	0	–	–
O_5	0	0	0	0	1	0	0	0	0	0	0	0	1
O_0	0	0	0	0	0	0	–	0	0	0	0	–	–

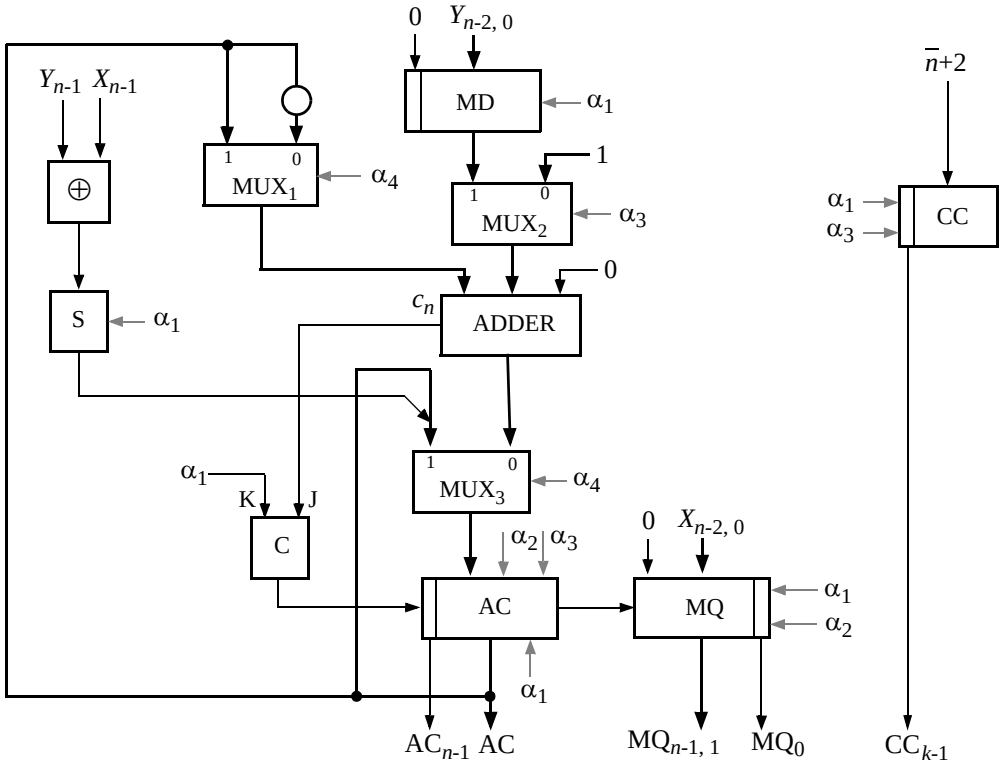


Fig. 7.18

7.7 - Realizzazione della parte di controllo

La parte di controllo di un sistema può essere realizzata seguendo due approcci: 1) può essere strutturata in **hardware** come una rete sequenziale (di Mealy o di Moore, secondo i casi); 2) può essere realizzata in una forma detta **microprogrammata** o **firmware**, la quale consiste essenzialmente nel descrivere il sistema mediante un particolare linguaggio di programmazione, detto di **microprogrammazione**, e memorizzare tale descrizione (microprogramma) in una ROM che costituisce il cuore della parte di controllo.

7.7.1 - Realizzazione hardware

La realizzazione in hardware richiede che dal diagramma di flusso venga opportunamente estratta l'informazione relativa alle variabili di controllo e alla loro evoluzione temporale, cosa necessaria perchè le azioni previste per la parte operativa possano avere luogo nell'ordine stabilito.

Infatti il diagramma di flusso descrive mediante i blocchi operazionali le azioni elementari che il sistema deve compiere e mediante i blocchi di decisione la successione temporale di queste azioni in accordo al presentarsi di certi eventi; non è peraltro in grado di esprimere

dinamicamente l'evoluzione del sistema in termini dell'evoluzione delle sue variabili di controllo.

D'altra parte, dal momento che le azioni elementari (μ-operazioni) sono realizzate utilizzando moduli logici e i dati, nei vari passi dell'elaborazione, vengono mantenuti in registri e instradati sui vari percorsi con l'aiuto di multiplexer, è necessario che ai moduli, ai registri e ai multiplexer arrivino negli istanti opportuni i segnali (a livelli e/o ad impulsi) richiesti perchè essi possano svolgere correttamente le loro funzioni.

Si tratta in sostanza di ricavare dal diagramma di flusso, ossia dalla conoscenza delle azioni, l'informazione contenuta sinteticamente nella tabella delle transizioni e nella tabella delle uscite di una rete sequenziale, in particolare della rete che costituisce la parte di controllo. A tale scopo si parte ancora dalla costruzione di un grafo, che chiameremo **grafo di stato del controllo**, i cui nodi rappresentano gli stati della ASM e ai quali sono associati, nodo per nodo, i segnali di controllo coinvolti nel tempo di ciclo in cui ogni stato è stato attuale per la macchina; inoltre, con l'aiuto della lista dei componenti logici che sono stati scelti per realizzare la parte operativa, si specifica se quei segnali devono essere livelli oppure si comportano come impulsi.

Riguardo al numero dei nodi del grafo, se si fa riferimento al modello di Moore, in generale c'è corrispondenza uno ad uno tra i blocchi operazionali del diagramma di flusso e gli stati del controllo (infatti ogni stato, con l'insieme delle variabili di controllo definite in esso, caratterizza univocamente l'uscita della parte di controllo, costituita dal codice della μ-operazione), con alcune eccezioni:

a) due o più μ-operazioni in sequenza possono essere eseguite sotto la medesima configurazione di un sottoinsieme di segnali di controllo, per esempio quando un segnale ha un livello costante alto o basso: in questi casi a due o più blocchi operazionali corrisponde uno stesso stato del controllo, che quindi permane per più di un tempo di ciclo;

b) può essere necessario introdurre uno o più stati che non corrispondono all'esecuzione di μ-operazioni, ma nei quali viene definito il valore di segnali di controllo che devono trovarsi già stabili negli stati successivi; per esempio se un modulo operazionale deve calcolare una funzione in certi momenti ed un'altra in altri momenti, sulla base del livello di un segnale di selezione, occorre prevedere uno stato in cui quel segnale viene definito al livello opportuno, prima di entrare nella sequenza di stati che si riferiscono a ciascuno dei due tipi di elaborazione.

c) ogni volta che una μ-operazione in cui viene modificato il contenuto di un registro è seguita da un blocco di decisione (o da un intero albero di decisione) nel quale viene effettuato un test sul contenuto di quel registro, è necessario introdurre di seguito al nodo relativo alla μ-operazione un nodo aggiuntivo che non corrisponde ad alcuna operazione della parte operativa, ma ha lo scopo di separare l'aggiornamento del registro ed il suo test, assegnandoli a periodi di clock consecutivi; questo evita l'incongruenza di un test su un valore in corso di modifica (non si dimentichi che i registri sono non trasparenti). Dal nodo aggiuntivo si dipartono gli archi che corrispondono alle varie condizioni logiche legate al valore modificato del registro (Fig. 7.19).

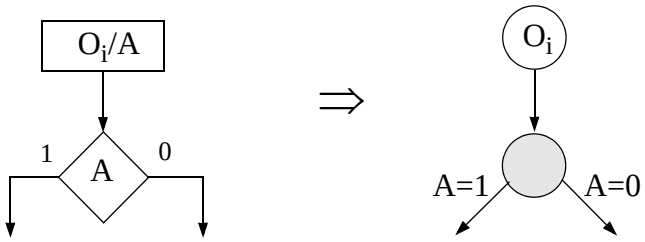


Fig. 7.19

Se invece il modello è quello di Mealy, ad uno stato possono essere associati uno o più blocchi operazionali in quanto, come si è visto in altra parte, tutte o parte delle uscite del controllo dipendono dalle condizioni di ingresso; in questo caso per ogni sottoinsieme di segnali di controllo, che devono essere attivi per una delle μ-operazioni implicate dallo stato attuale, sarà specificato il particolare stato di ingresso da associare ad esso: questa informazione si desume dai blocchi operazionali, che sono racchiusi sotto il medesimo stato.

Per specificare nel grafo se i segnali di controllo sono attivi a livello alto o basso, ci possono essere vari metodi, per esempio si può fare seguire il nome di un segnale di controllo dal suffisso H o L rispettivamente, oppure più semplicemente se ne può dichiarare il nome nello stato o negli stati in cui esso deve essere attivo, purchè tutti i moduli logici richiedano segnali con lo stesso livello di attivazione (alto o basso).

Un segnale che viene dichiarato con il suo livello attivo in uno stato ma non nel precedente e nel seguente si configura come un impulso della durata di un ciclo e se l'elemento su cui agisce è sensibile al fronte (per esempio un contatore) questa dichiarazione deve essere fatta ogni volta che quell'elemento deve riceverlo.

Viceversa un segnale dichiarato attivo per più stati consecutivi, subisce una variazione di livello nel primo stato della sequenza (quindi ha un fronte), e mantiene poi il livello raggiunto per i cicli corrispondenti, ritornando al livello iniziale nel primo stato successivo in cui non è più asserito. A questo proposito è da osservare che se un segnale deve lavorare a livelli per più di un tempo di ciclo, occorre definirlo con il valore con cui è attivo in tutti gli stati consecutivi in cui è coinvolto; in alternativa esso può essere considerato l'uscita di un latch e occorrono allora due segnali a impulsi, asseriti in stati opportuni, per il set ed il reset del latch.

Per esempio il seguente spezzone di grafo formato dalla successione di cinque stati, in tre dei quali è identificato il segnale α con la specifica H, definisce l'andamento temporale di quel segnale come è mostrato in Fig. 7.20.

Come si vede il segnale α rimane a livello alto nello stato 1 per un ciclo e negli stati 3 e 4 per due cicli; inoltre l'interposizione dello stato 2 vuoto tra gli stati 1 e 3 determina un secondo fronte in salita di α, oltre a quello iniziale in corrispondenza dello stato 1.

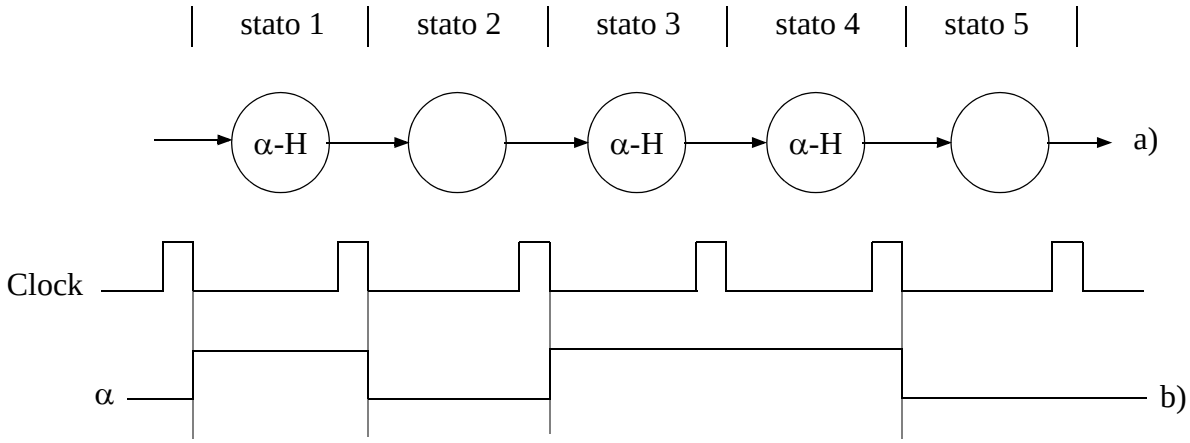


Fig. 7.20

Il grafo di stato costruito con le modalità descritte può essere utilizzato per realizzare la parte di controllo direttamente con i metodi di sintesi propri delle reti sequenziali, almeno nei casi semplici (pochi stati interni e pochi stati di ingresso), secondo lo schema generale descritto dalla Fig. 7.21. In esso è presente un registro che contiene lo stato attuale del sistema, mentre una logica di generazione dello stato successivo sceglie lo stato successivo in base allo stato attuale e allo stato di ingresso; infine una logica di uscita decodifica lo stato attuale per generare i segnali di controllo per la parte operativa.

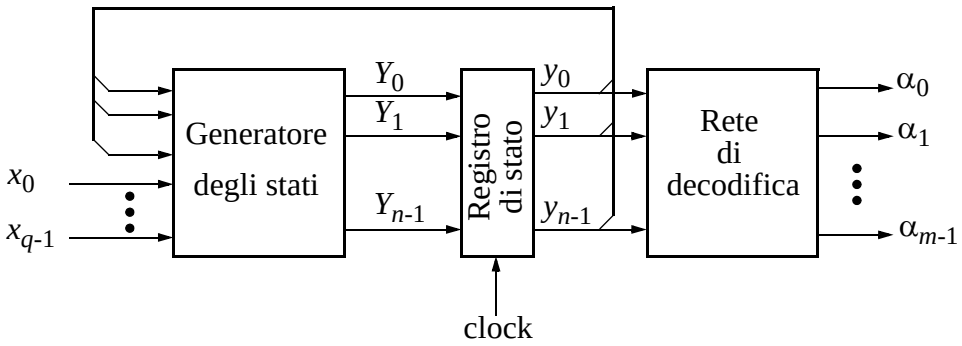


Fig. 7.21

Tuttavia una parte di controllo realizzata in questo modo non evidenzia chiaramente la corrispondenza tra struttura del controllo e struttura dell'algoritmo e qualunque modifica di questo comporta il totale rifacimento della rete. Altri approcci, utilizzando la stessa struttura per esprimere gli stati e la stessa codifica delle variabili di stato e di uscita, consentono un progetto che risponde maggiormente a criteri di chiarezza e di aderenza all'algoritmo; tra questi particolarmente interessanti sono quelli basati sull'idea di ricavare lo stato successivo leggendolo da una tabella (tecnica di **table look-up**), realizzata mediante una ROM o semplicemente mediante multiplexer, come mostra il seguente esempio.

Esempio

Applichiamo il metodo al progetto della parte di controllo del sistema descritto dal diagramma ASM di Fig. 7.17, dal quale deduciamo il grafo degli stati illustrato in Fig. 7.22:

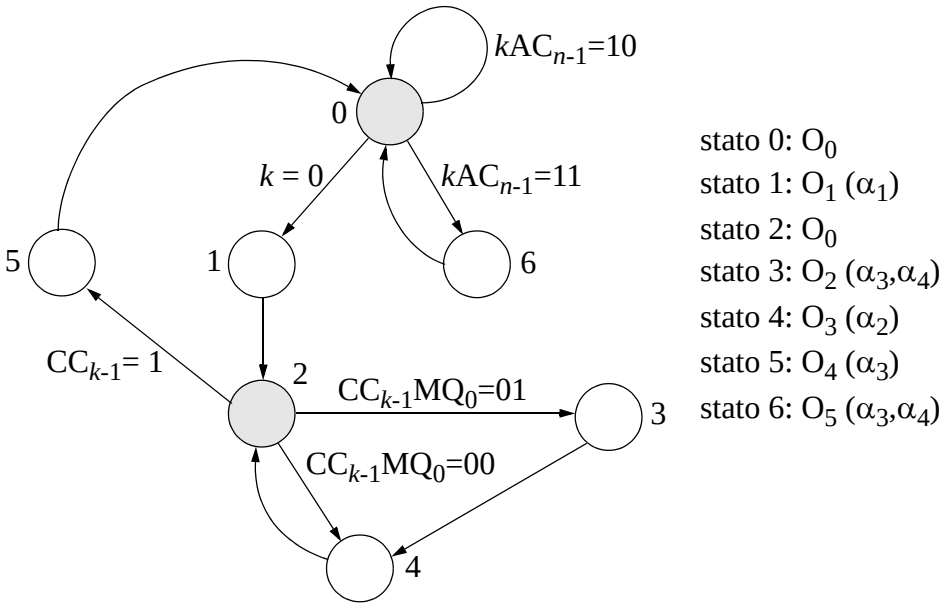


Fig. 7.22

Per quanto riguarda la rete combinatoria denominata generatore degli stati in Fig. 7.21, essa verrà realizzata con una tecnica che fa uso di un multiplexer per ciascuna variabile di stato; la selezione di tutti multiplexer è effettuata in parallelo mediante le variabili dello stato successivo, mentre gli ingressi sono determinati elaborando le variabili di condizione attraverso semplici reti combinatorie, definite dalla seguente **tabella degli stati**:

stato		MUX$_2$	MUX$_1$	MUX$_0$
0	000	k·AC$_{n-1}$	k·AC$_{n-1}$	$\overline{k}$
1	001	0	1	0
2	010	CC$_{k-1}$+$\overline{\text{MQ}_0}$	$\overline{\text{CC}}_{k-1}$·MQ$_0$	CC$_{k-1}$+MQ$_0$
3	011	1	0	0
4	100	0	1	0
5	101	0	0	0
6	110	0	0	0

Per meglio comprendere come è stata realizzata la tabella, consideriamo sul grafo lo stato 0: da esso si può passare nello stato 1 per k = 0 e qualunque valore di AC$_{n-1}$; nello stato 0 stesso per k = 1 e AC$_{n-1}$ = 0 e infine nello stato 6 per k = 1 e AC$_{n-1}$ = 1; possiamo così

costruire la seguente tavola di verità:

k	AC_{n-1}	MUX_2	MUX_1	MUX_0
0	0	0	0	1
0	1	0	0	1
1	0	0	0	0
1	1	1	1	0

da cui si ricavano le espressioni logiche riportate nella prima riga della tabella degli stati; dallo stato 1 si passa sempre allo stato 2, qualunque valore abbiano le variabili di condizione, pertanto nella seconda riga della tabella degli stati si scrive semplicemente 010. Analogamente si procede per le altre righe. La corrispondente struttura è mostrata nella Fig. 7.23.

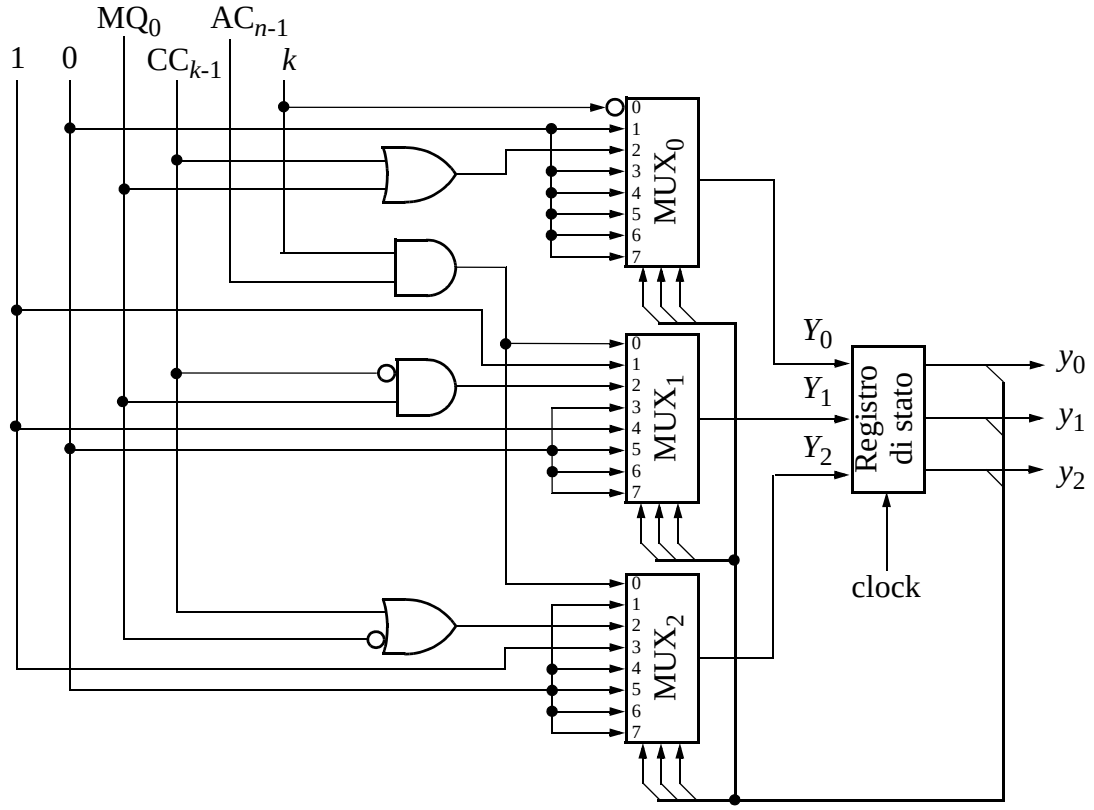


Fig. 7.23

Per quanto riguarda la generazione delle variabili di controllo, si decodificano le variabili dello stato attuale e si prelevano le variabili di controllo o direttamente dalle uscite appropriate del decodificatore, oppure da gruppi di queste riunite attraverso porte OR, a seconda che una variabile di controllo sia attiva in un solo stato oppure in più di uno; nel caso specifico la rete che produce le variabili di controllo è strutturata come in Fig. 7.24.

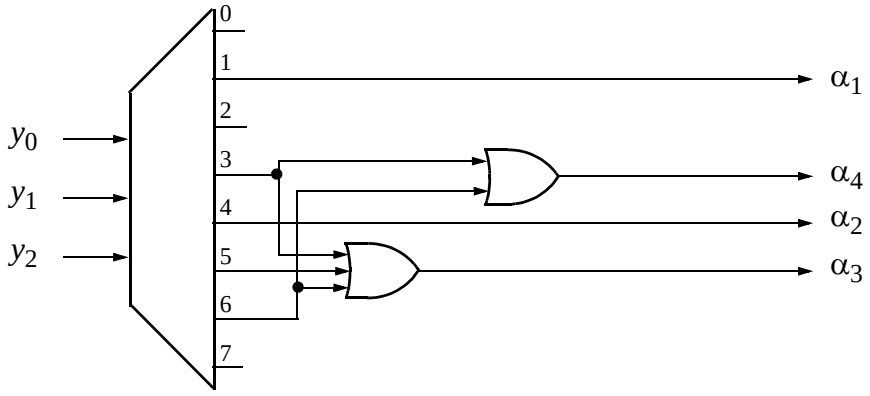


Fig. 7.24

Quando una variabile di controllo deve avere un comportamento a livelli, rimanendo ad valore fisso per più periodi di clock consecutivi, come è stato detto in altra parte può essere generata attraverso l'uscita di un latch SR, il cui set e reset sono effettuati rispettivamente dall'uscita del decodificatore associata al primo e da quella associata all'ultimo dei periodi di clock consecutivi durante i quali il segnale è attivo. Se vi sono problemi di temporizzazione (tempo di commutazione del latch paragonabile alla durata del tempo T_s), si può anticipare il set effettuandolo mediante l'uscita del decodificatore che corrisponde allo stato assunto dal controllo nel tempo di ciclo precedente quello in cui la variabile di controllo deve diventare attiva e si può anticipare il reset mediante l'uscita del decodificatore che corrisponde al penultimo dei periodi di clock in cui la variabile è attiva. In ogni caso è necessario tenere conto in modo accurato della temporizzazione, per evitare il rischio di anticipare (o di ritardare) eccessivamente il set ed il reset.

7.7.2 - Realizzazione microprogrammata (firmware)

Accanto al metodo illustrato nel paragrafo 7.7.1 per tradurre il diagramma di flusso di un sistema in una struttura hardware che implementa l'algoritmo di controllo, ne esiste un altro il quale consente al tempo stesso di descrivere il controllo di un sistema – e in questo senso ha una funzione analoga a quella del grafo di stato di una ASM – e di realizzarlo.

Questo metodo è noto come **microprogrammazione** ed è basato sulla scrittura di un particolare programma detto **microprogramma** mediante il quale viene descritto il comportamento dell'algoritmo; lo stesso microprogramma, memorizzato in una ROM, serve per controllare e dirigere le attività della parte operativa e in questa funzione costituisce l'essenza di una parte di controllo detta **microprogrammata** o **firmware**.

Per scrivere un μ-programma è necessario un linguaggio specifico, che è detto linguaggio di microprogrammazione o **μ-linguaggio**; lo *statement* fondamentale di un **μ-linguaggio** è la **microistruzione** (μ-istruzione), ovvero la descrizione delle funzioni eseguite da un sistema, inteso come insieme delle due parti PO e PC, in un passo elementare; un **microprogramma**

(μ-programma) è l'insieme delle μ-istruzioni che descrivono un'operazione OP in tutti i passi elementari che la compongono.

Una μ-istruzione è scritta secondo la seguente sintassi:

<etichetta>"|"<frase semplice>|<frase condizionata>{";"<frase condizionata>} | <μ-ordine>
<trasferimento condizionato> {";"<trasferimento condizionato>}
<etichetta>::= "μ_h"

<frase semplice>::="O_j,μ_k"

<frase condizionata>::="(C_r) O_j , μ_k"

<trasferimento condizionato>::="(C_r) μ_k"

La frase semplice viene interpretata come l'asserzione «esegui la μ-operazione O_j, quindi trasferisci il controllo alla μ-istruzione μ_k»; quella condizionata è interpretata come «se la condizione logica $C_r = f(x_1, x_2, \ldots, x_s)$ è vera esegui la μ-operazione O_j, quindi trasferisci il controllo alla μ-istruzione μ_k»; infine il trasferimento condizionato viene interpretato come «se la condizione logica $C_r = f(x_1, x_2, \ldots, x_s)$ è vera trasferisci il controllo alla μ-istruzione μ_k».

I μ-linguaggi possono essere divisi in due grandi famiglie: quelli le cui μ-istruzioni contengono solo frasi condizionate e che sono detti **a struttura di frase** (phrase structured o *ps*) e quelli caratterizzati da μ-istruzioni che contengono solo trasferimenti condizionati e che sono detti **a struttura di trasferimento** (transfer structured o *ts*).

Limitandoci per semplicità in questa trattazione solo ai linguaggi *ts*, una μ-istruzione nella sua forma più generale è scritta nel seguente modo:

$$\mu_h \mid O_j\ (C_1)\ \mu^1{}_k\ ;\ (C_2)\ \mu^2{}_k\ ; \ldots ; (C_q)\ \mu^q{}_k$$

e la sua interpretazione è:

«*esegui la μ-operazione O_j, quindi se la condizione C_1 è vera, vai alla μ-istruzione $\mu^1{}_k$, altrimenti se è vera la condizione C_2 vai alla μ-istruzione $\mu^2{}_k$, altrimenti ...*».

Nel caso in cui due o più simboli di trasferimento siano uguali, la μ-istruzione generale può essere semplificata; per esempio se $\mu^s{}_k = \mu^t{}_k = \mu^*{}_k$, i trasferimenti condizionati si riducono al seguente (C_{st}) $\mu^*{}_k$, dove C_{st} è l'unione delle condizioni C_s e C_t. Nel caso limite in cui tutte le condizioni sono uguali, la μ-istruzione si riduce alla frase semplice:

$$\mu_h \mid O_j, \mu_k$$

Esaminiamo ora in maggiore dettaglio la relazione tra le condizioni C_r , $r = 1, 2, \ldots, q$ e le variabili di condizione $x_1, x_2, \ldots, x_s$. Come è stato detto, ogni C_r è una funzione booleana di quelle variabili e dovendo essere vera quando tutte le altre condizioni sono false, una sua

possibile espressione è costituita da uno dei $q = 2^s$ prodotti completi (mintermini) associato ad una delle q combinazioni binarie distinte, o **stati**, X_r delle variabili stesse. Per esempio se $s = 3$ esistono 8 possibili funzioni distinte e una di esse, diciamo $x_1\bar{x}_2x_3$, può essere messa in corrispondenza con lo stato 101, scrivendo $x_1\bar{x}_2x_3 = 101$ con il che si fa riferimento alla condizione C_5 che è vera se $x_1 = 1$, $x_2 = 0$, $x_3 = 1$; analogamente allo stato 111 sarà associata l'espressione $x_1x_2x_3$, a formare la condizione $C_7 \equiv x_1x_2x_3 = 111$. Due o più condizioni C_1, C_2, ..., C_r possono inoltre essere sostituite da una condizione composta $C_{12\ldots r}$, ottenuta dall'unione delle singole condizioni e associabile a tutti gli stati individuati separatamente da esse. La condizione composta è interpretata affermando «se la condizione C_1 è vera o la condizione C_2 è vera , …, allora». Per esempio alle due condizioni C_5 e C_7 può essere sostituita la condizione composta $C_{57} \equiv x_1\bar{x}_2x_3+x_1x_2x_3 = 101+111$, interpretabile come «se la condizione C_5 è vera o la condizione C_7 è vera, allora …»; in alternativa si può utilizzare la forma semplificata $x_1x_3 = 11$ che significa «se $x_1 = 1$ e $x_3 = 1$, allora …». Notare che quest'ultima notazione, che corrisponde al costrutto *case* di un linguaggio ad alto livello, consente di individuare direttamente la condizione vera, senza bisogno di scandire tutte le altre.

Da ultimo verifichiamo il fatto fondamentale che un μ-programma è realmente in grado di descrivere il comportamento di un sistema, modellato come insieme delle due parti PO e PC interconnesse.

A livello di sistema, descrivere il suo comportamento significa specificare ad ogni passo lo stato successivo e lo stato di uscita di ciascuna delle due macchine sequenziali PO e PC; a livello di μ-programma significa definire ad ogni passo la μ-istruzione che deve essere eseguita. Si tratta dunque di individuare la relazione tra macchina sequenziale e μ-programma.

Supponiamo che di un sistema sia dato il μ-programma e che sia noto lo stato di ingresso e lo stato interno di PO ad ogni passo elementare; pertanto è noto anche lo stato di ingresso X_r di PC, una volta note le espressioni delle variabili di condizione, nell'ipotesi che tali variabili siano funzione solo di informazioni esterne e/o del contenuto di registri di PO, e non di variabili di controllo (ossia nell'ipotesi che PO sia modellata come macchina di Moore).

Al simbolo della generica μ-istruzione μ_h associamo uno stato di PC e supponiamo che qualunque μ-istruzione riferita in un trasferimento sia ancora una μ-istruzione del μ-programma (**μ-programma chiuso**); allora lo stato attuale sarà μ_h e i possibili stati successivi saranno μ^1_k, μ^2_k, …, μ^q_k, ciascuno relativo ad uno stato di ingresso di PC X_r , r = 1, 2, …, q. Dato uno specifico stato di ingresso X_r, è nota la condizione logica C_r ad esso associata e quindi lo stato successivo μ^r_k della macchina che modella PC. D'altra parte gli stati di uscita della parte di controllo PC sono rappresentati dai codici delle μ-operazioni O^r_j e poiché, secondo la sintassi illustrata, nelle μ-istruzioni i codici O^r_j non dipendono dalle condizioni C_r, lo stato di uscita di PC dipende solo dallo stato interno.

Risulta dunque evidente che ogni μ-istruzione definisce una riga di una tabella di flusso di

una macchina di Moore, per cui una volta definito l'insieme degli stati di ingresso X_r, degli stati di uscita O_j e degli stati interni μ_h tramite un μ-programma, è possibile definire la tabella di flusso di una macchina di Moore che descrive il comportamento della parte PC del sistema. Tale tabella ha un numero di colonne pari al numero q di stati di ingresso X_r ed un numero di righe pari al numero di μ-istruzioni del μ-programma; ogni casella riporta lo stato successivo μ^i_k.

Se il μ-linguaggio comprende anche la lista delle μ-operazioni O_j, il μ-programma descrive compiutamente anche il comportamento della PO, dal momento che essa è definita proprio da tale lista.

Possiamo concludere dicendo che un μ-programma chiuso ed una tabella di flusso sono modi equivalenti di descrivere una macchina sequenziale e anzi un μ-programma completo della lista delle μ-operazioni è al tempo stesso la descrizione del comportamento di un sistema e la componente essenziale della sua parte di controllo.

La scrittura di un μ-programma partendo dal diagramma di flusso procede secondo le seguenti regole.

a) Si considera il blocco di ingresso, o uno non ancora considerato, connesso all'uscita di uno già esaminato.

b) Se esso è un blocco di decisione si considera, se esiste, l'albero di decisione avente quel blocco come radice e in corrispondenza si scrive una μ-istruzione del tipo:

$$\mu_h \mid O_0\,(C_1)\,\mu^1_k\,;\ (C_2)\,\mu^2_k\,;\ \ldots;\,(C_n)\,\mu^n_k$$

dove il numero n di trasferimenti condizionati è pari al numero di uscite dell'albero.

c) Se si tratta di un blocco operativo, seguito da un altro blocco operativo, si scrive una μ-istruzione del tipo:

$$\mu_h \mid O_j,\,\mu_k$$

Invece se il blocco operativo è seguito da un blocco di decisione, si esamina l'albero di decisione avente come radice il blocco di decisione; se nessuna delle condizioni di tale albero (eventualmente costituito dalla sola radice) dipende da registri modificati dalla μ-operazione del blocco operativo, si considerano insieme questo blocco e l'albero e in corrispondenza si scrive una μ-istruzione del tipo:

$$\mu_h \mid O_j\,(C_1)\,\mu^1_k\,;\ (C_2)\,\mu^2_k\,;\ \ldots;\,(C_n)\,\mu^n_k$$

dove n è ancora il numero di uscite dell'albero. Viceversa se almeno in un blocco dell'albero figura una condizione dipendente dai registri modificati nel blocco operativo in esame, si considerano insieme tale blocco e la porzione di albero (eventualmente vuota) nella quale non esistono condizioni dipendenti dai registri sotto modifica e si scrive una μ-istruzione come la precedente, ovvero:

$$\mu_h \mid O_j\,(C_1)\,\mu^1{}_k\,;\ (C_2)\,\mu^2{}_k\,;\ \ldots;\ (C_i)\,\mu^i{}_k$$

salvo che il numero *i* di trasferimenti condizionati in essa presenti è limitato al numero dei nodi del sottoalbero scelto. Si considera quindi il sottoalbero rimanente e per esso si scrive una μ-istruzione del tipo:

$$\mu_h \mid O_0\,(C_1)\,\mu^{i+1}{}_k\,;\ (C_2)\,\mu^{i+2}{}_k\,;\ \ldots;\ (C_l)\,\mu^n{}_k.$$

Il procedimento è ripetuto fino ad esaurire tutti i blocchi.

Come esempio di microprogrammazione, riprendiamo il sistema descritto dal diagramma di flusso di Fig. 7.17. Con le regole enunciate, il microprogramma è il seguente:

$\mu_1 \mid O_0\,(k{=}0)\,\mu_2;\ (k\mathrm{AC}_{n-1}{=}10)\,\mu_1;\ (k\mathrm{AC}_{n-1}{=}11)\,\mu_8$
$\mu_2 \mid O_1,\ \mu_3$
$\mu_3 \mid O_0\,(\mathrm{CC}_{k-1}\mathrm{MQ}_0{=}00)\,\mu_4;\ (\mathrm{CC}_{k-1}\mathrm{MQ}_0{=}01)\,\mu_5;\ (\mathrm{CC}_{k-1}{=}1)\,\mu_6$
$\mu_4 \mid O_3,\ \mu_3$
$\mu_5 \mid O_2,\ \mu_4$
$\mu_6 \mid O_4\,(k{=}0)\,\mu_2;\ (k{=}1)\,\mu_7$
$\mu_7 \mid O_0\,(\mathrm{AC}_{n-1}{=}0)\,\mu_1;\ (\mathrm{AC}_{n-1}{=}1)\,\mu_8$
$\mu_8 \mid O_5\,(k{=}0)\,\mu_2;\ (k{=}1)\,\mu_7$

Rispetto al grafo degli stati ottenuto per la realizzazione hardware, si noti come questo stile di microprogrammazione tende ad aumentare il numero degli stati della macchina sequenziale (8 invece di 7): ciò è dovuto essenzialmente al fatto di anticipare il più possibile le decisioni prese dal controllo, a vantaggio in definitiva della velocità dell'intero sistema.

Lo schema di principio di un controllo microprogrammato in linguaggio *ts* prevede una memoria di sola lettura, una rete logica combinatoria che ha la funzione di generare gli indirizzi per la lettura della memoria ed un registro: la parte combinatoria dell'intera rete sequenziale è costituita dalla memoria e dalla rete logica di indirizzamento; si ricordi infatti che una ROM è per sua natura un circuito combinatorio.

Ogni cella della ROM contiene una μ-istruzione completa (cioè il codice della μ-operazione O_j, sotto forma di stringa binaria che codifica le variabili di controllo, gli indirizzi delle μ-istruzioni successive $\mu^r{}_k$ e le condizioni logiche associate C_r) e per questo tale schema è detto a **memorizzazione di microistruzione** (mm). Il numero delle celle è pari al numero delle μ-istruzioni ed ogni cella ha una lunghezza sufficiente ad accogliere la μ-istruzione più lunga ovvero al massimo $L_O+2^s\times(s+\lceil \log_2 N_\mu \rceil)$, dove L_O è la lunghezza del campo riservato al codice di μ-operazione, N_μ il numero di μ-istruzioni del μ-programma, *s* il numero delle variabili di condizione e 2^s il numero delle possibili condizioni logiche.

I codici delle condizioni logiche e delle μ-istruzioni $\mu^r{}_k$ associate, contenuti nella cella

relativa alla μ-istruzione corrente, vengono trasferiti dalla ROM all'ingresso della rete logica la quale, in base allo stato X_r delle variabili di condizione, pure applicato in ingresso, genera l'indirizzo della cella di memoria contenente la prossima microistruzione μ^i_k, in base alla condizione logica C_i che risulta vera; la rete logica si può dunque pensare costituita da due parti: una che determina quale delle condizioni logiche è verificata per lo stato di ingresso X_r, l'altra che genera l'indirizzo μ^i_k; l'indirizzo μ^i_k viene memorizzato nel registro R (Fig. 7.25).

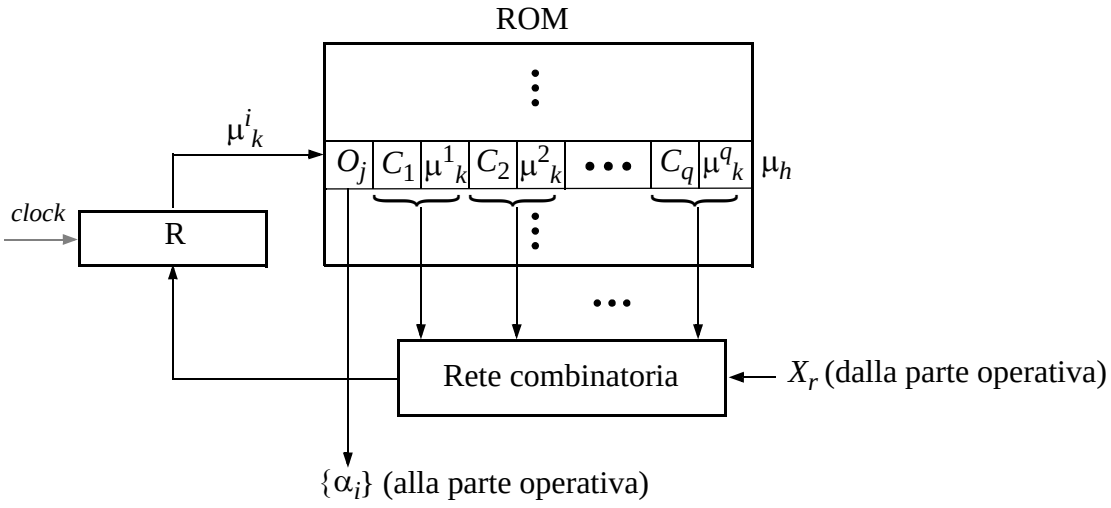


Fig. 7.25

La struttura della rete combinatoria può essere semplificata e resa più flessibile con l'utilizzo di moduli combinatori standard in base alle seguenti considerazioni.

Se nel microprogrammare il sistema non si pone alcun vincolo sul numero di trasferimenti condizionati che possono essere presenti in una μ-istruzione, si deve ammettere la possibilità di un numero di trasferimenti pari a 2^s, se s è il numero di variabili di condizione presenti nel μ-programma. Questo comporta considerare μ-istruzioni espanse nelle quali figurino tutte le 2^s condizioni ottenute dalla codifica delle variabili di condizione. Per esempio se esistono tre variabili di condizione x_1, x_2, x_3, una μ-istruzione del tipo:

$$\mu_h \mid O_j\ (x_1 = 0)\ \mu_a;\ (x_1x_2 = 10)\ \mu_b;\ \ (x_1x_2 = 11)\ \mu_c$$

deve essere espansa in

$$\mu_h \mid O_j\ (x_1x_2x_3 = 000)\ \mu_a;\ (x_1x_2x_3 = 001)\ \mu_a;\ \ldots;\ (x_1x_2x_3 = 011)\ \mu_a;\ (x_1x_2x_3 = 100)\ \mu_b;\ (x_1x_2x_3 = 101)\ \mu_b;\ \ldots;\ (x_1x_2x_3 = 110)\ \mu_c;\ (x_1x_2x_3 = 111)\ \mu_c$$

Se tutte le μ-istruzioni sono espanse in questo modo, in ogni cella della ROM i 2^s indirizzi di trasferimento specificati in ogni μ-istruzione possono essere associati implicitamente per posizione alle rispettive condizioni logiche, senza bisogno che anche queste siano

memorizzate, con risparmio sulla lunghezza di parola; inoltre la rete logica si riduce ad un semplice multiplexer (Fig. 7.26).

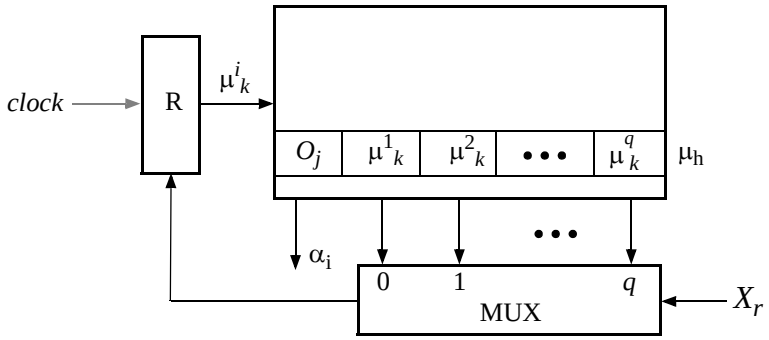


Fig. 7.26

Tuttavia questa soluzione non è praticabile non appena il numero delle variabili di condizione supera qualche unità, ed il motivo è l'aver fatto ricorso ad un μ-linguaggio verticale generale. In pratica i μ-linguaggi verticali che si considerano sono caratterizzati da un numero di trasferimenti limitato, ossia il numero effettivo *n* delle variabili di condizione è solo 2 o 3. In questi casi ogni μ-istruzione è espansa in 2^n trasferimenti condizionati, i quali vengono ordinati come nel caso generale; la differenza sta nel fatto che le variabili di condizione sono diverse da una μ-istruzione espansa all'altra, per esempio si possono avere due μ-istruzioni espanse del tipo:

$$\mu_h \mid O_j\ (x_1x_2 = 00)\ \mu_a;\ (x_1x_2 = 01)\ \mu_b;\ \ (x_1x_2 = 10)\ \mu_c;\ (x_1x_2 = 11)\ \mu_d$$
$$\mu_k \mid O_j\ (x_1x_3 = 00)\ \mu_a;\ (x_1x_3 = 01)\ \mu_a;\ \ (x_1x_3 = 10)\ \mu_c;\ (x_1x_3 = 11)\ \mu_c$$

con $x_3 \neq x_1$; questo comporta la necessità di operare una scelta tra le *s* variabili di condizione in modo da poter continuare a controllare i trasferimenti con un multiplexer. A tale scopo si inseriscono in ogni cella di memoria dei campi in numero pari alle variabili da identificare, ossia *n*, e ogni campo contiene un codice binario associato ad una delle *s* variabili di condizione, lungo $t = \lceil \log_2 s \rceil$ bit, con il quale si controlla un multiplexer con $\lceil \log_2 s \rceil$ ingressi che ha come ingressi-dati tutte le variabili di condizione, tra quali ne viene selezionata una, diversa da un multiplexer all'altro. Le *n* variabili selezionate controllano a loro volta un multiplexer con 2^n ingressi per la scelta della μ-istruzione successiva (Fig. 7.27).

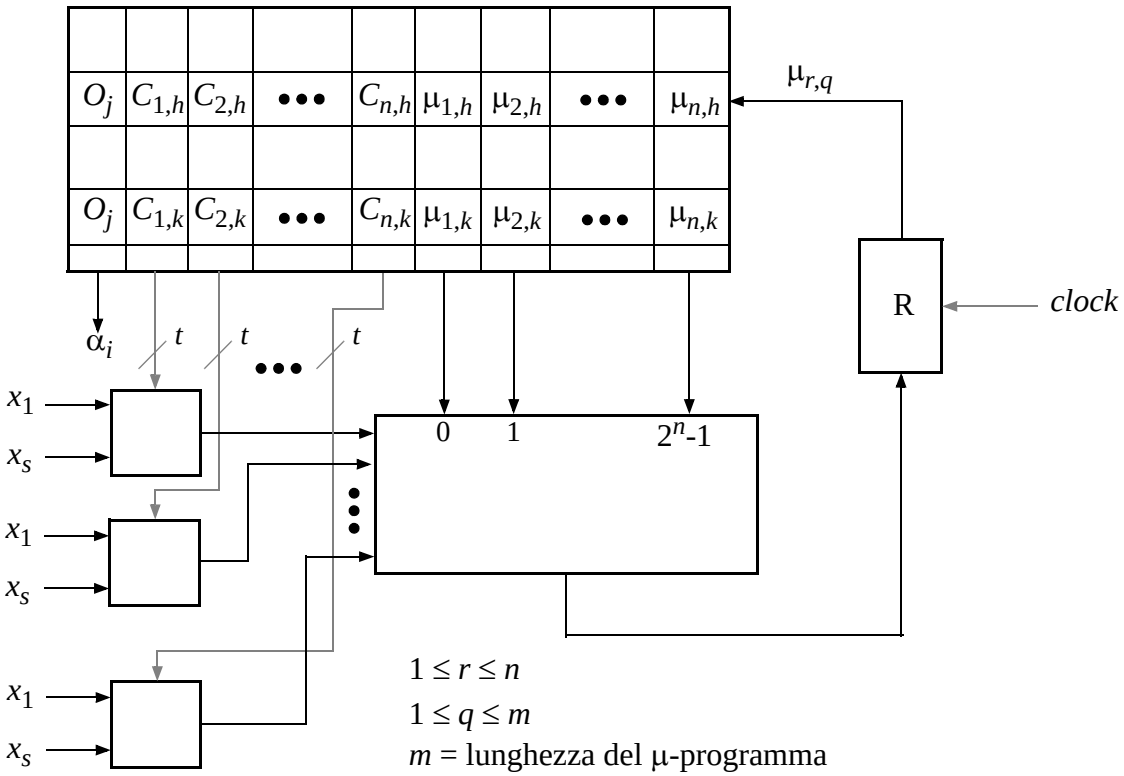


Fig. 7.27

7.8 - Velocità di esecuzione

Ricordando che T indica il tempo di ciclo del sistema, il tempo medio impiegato per compiere l'operazione i-esima del sistema è dato dalla relazione:

$$\overline{T}_{e,i} = \overline{n}_i \cdot T \tag{7.1}$$

essendo $\overline{n}_i$ il numero medio di μ-operazioni richieste per quell'operazione; di qui si può desumere il tempo di elaborazione medio del sistema, ossia il tempo che mediamente il sistema impiega per eseguire una qualunque delle sue v operazioni, nell'ipotesi che esse siano equiprobabili:

$$\overline{T}_e = \frac{\sum_{i=1}^{v} \overline{T}_{e,i}}{v} \tag{7.2}$$

L'inverso di $\overline{T}_e$ è un parametro che viene assunto come indice significativo per misurare le

prestazioni di un sistema e viene misurato in milioni di operazioni al secondo (MIPS).

Nella realtà le operazioni non hanno la stessa probabilità di occorrenza, per cui se indichiamo con p_i la probabilità dell'operazione *i*-esima (valutata su base statistica, per esempio attraverso una simulazione), avremo:

$$\bar{T}_e = \sum_{i=1}^{v} p_i \bar{T}_{e,i} = T \cdot \sum_{i=1}^{v} p_i \bar{n}_i = T \cdot \bar{N} \tag{7.3}$$

con $\bar{N}$ numero medio di passi per operazione.

Come è noto, il tempo di ciclo dipende dal tempo elementare e dalla durata dell'impulso di clock secondo la relazione $T = T_S + \Delta M$. Per calcolare il tempo elementare, ricordiamo che esso è il tempo minimo che può intercorrere tra due impulsi di clock consecutivi in un sistema costituito da due reti sequenziali connesse in ciclo. Se il sistema è del tipo Moore-Mealy o Mealy-Moore, cioè eterogeneo, per il calcolo si deve considerare solo uno spezzone del ciclo, mentre se è del tipo Moore-Moore si devono considerare entrambi gli spezzoni, a causa del ritardo delle uscite della macchina di Moore.

Sia t_{PO} il tempo massimo richiesto per propagare una variabile di controllo α_i da un ingresso della parte operativa PO ad un registro della stessa, attraverso multiplexer e moduli logici e t_x il ritardo massimo con cui viene formata una variabile di condizione, a partire dal contenuto di un registro di PO; si osservi che t_x in molti casi può essere considerato nullo, e precisamente quando una variabile di condizione è direttamente l'uscita di un registro di PO. Inoltre siano t_{ROM} e t_{RL} i tempi massimi di trasmissione nella parte di controllo, supposta microprogrammata[1]. In riferimento agli schemi Moore-Mealy e Moore-Moore di Fig. 7.28, avremo rispettivamente nei due casi:

$$T = T_S + \Delta M = \Delta^{PO}_{s,z} + \Delta^{PC}_{i,z} + \Delta^{PO}_{i,s'} + \Delta M = t_x + (t_{ROM} + t_{RL}) + t_{PO} + \Delta M \tag{7.4}$$

$$T = max(T_{S_1}, T_{S_2}) + \Delta M \tag{7.5}$$

essendo:

$$T_{S_1} = \Delta^{PO}_{s,z} + \Delta^{PC}_{i,s'} = t_x + t_{RL} \qquad T_{S_2} = \Delta^{PC}_{s,z} + \Delta^{PO}_{i,s'} = t_{ROM} + t_{PO} \tag{7.6}$$

In genere si ha $T_{S_1} << T_{S_2}$, dal momento che t_{PO} è determinato dal modulo logico più lento, di

1. Se PC è realizzata in hardware, al posto dei termini t_{ROM} e t_{RL}, che tengono conto del tempo di lettura della memoria del microprogramma e del tempo di risposta della rete combinatoria di indirizzamento, si devono considerare rispettivamente il tempo di risposta della rete di decodifica delle variabili di controllo e il tempo di risposta del generatore degli stati (Fig. 7.21).

solito un addizionatore o anche un modulo funzionalmente più complesso, quale ad esempio un moltiplicatore, per cui risulta già $t_x+t_{RL} < t_{PO}$, anche senza considerare il contributo dovuto al tempo di accesso della ROM.

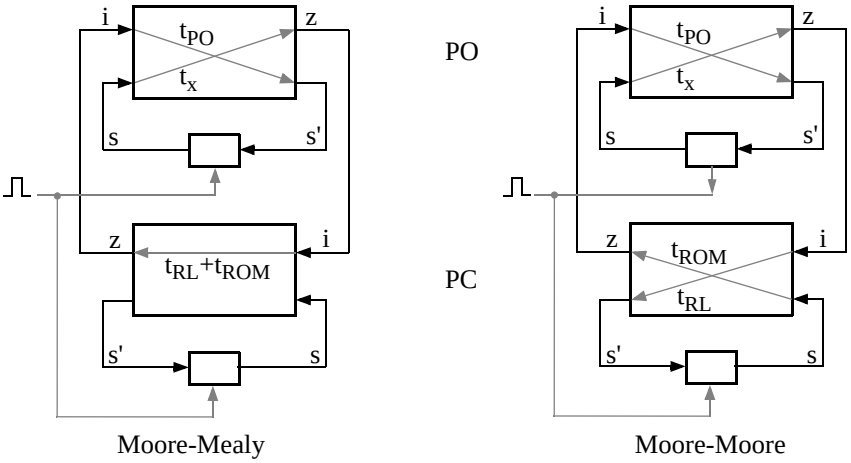


Fig. 7.28

Nel caso di sistema Mealy-Moore si ottiene (Fig. 7.29):

$$T = \Delta^{PC}_{s,z} + \Delta^{PO}_{i,z} + \Delta^{PC}_{i,s'} + \Delta M = t_{ROM} + (t_{PO} + t_x) + t_{RL} + \Delta M \qquad (7.7)$$

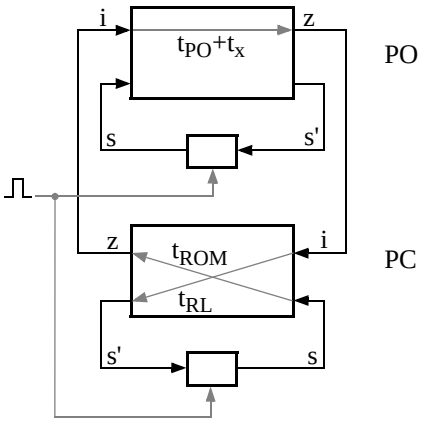


Fig. 7.29

Nell'ipotesi che i tempi t_{PO}, t_x, t_{ROM} e t_{RL} siano gli stessi per i sistemi esaminati, si nota dalle espressioni di T che il tempo di ciclo dei sistemi Moore-Mealy e Mealy-Moore è lo stesso, mentre quello del sistema Moore-Moore è inferiore della quantità t_x+t_{RL}, nell'ipotesi, precedentemente fatta, che sia $t_{ROM}+t_{PO} >> t_x+ t_{RL}$.

Consideriamo infine il caso di sistema Moore-Moore dove la parte di controllo è realizzata come macchina di Mealy ritardata, ossia avente un registro R sull'uscita. Poichè tale registro corrisponde ad una macchina di Moore inserita nel ciclo di due macchine preesistente e per la

quale è $\Delta^R_{s,z} = \Delta^R_{i,s'} = 0$, si ha (Fig. 7. 30):

$$T = max(T_{S_1}, T_{S_2}) + \Delta M$$
$$T_{S_1} = \Delta^{PO}_{s,z} + \Delta^{PC}_{i,z} + \Delta^{R}_{i,s'} = t_x + (t_{RL} + t_{ROM}) + 0 \qquad (7.8)$$
$$T_{S_2} = \Delta^{R}_{s,z} + \Delta^{PO}_{i,s'} = 0 + t_{PO}$$

Queste relazioni mostrano che il tempo di ciclo di questo sistema è ulteriormente ridotto rispetto a quello del sistema Moore-Moore, in quanto, anche nel caso in cui risultasse $T_{S_1} > T_{S_2}$ sarebbe pur sempre $t_x+t_{RL}+t_{ROM} < t_{ROM}+t_{PO}$.

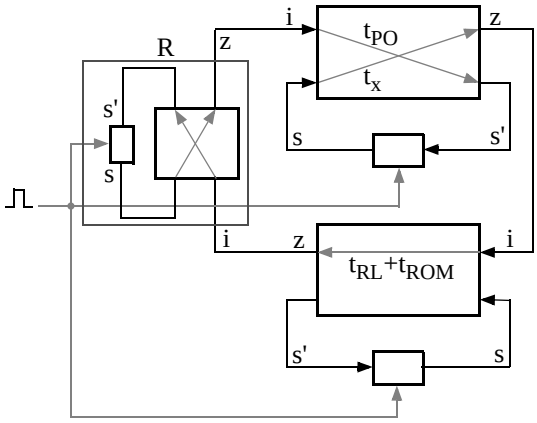


Fig. 7.30

Se due sistemi sono equivalenti, il confronto fra i tempi di latching può essere esteso ai tempi medi di elaborazione $\overline{T}_e$, ma se non lo sono è necessario tenere in conto anche il numero medio $\overline{N}$ di μ-operazioni per operazione, proprio di ciascuno di essi, oltre ai valori di *T*.

Per esempio in un sistema Moore-Moore ed in uno Moore-Mealy non equivalenti il confronto tra i tempi medi di elaborazione implica un bilancio tra il valore di $\overline{n}_i$ (e quindi di $\overline{N}$), che è più elevato nel primo sistema, ed il valore di *T* che viceversa è più grande per il secondo sistema. Lo stesso criterio può essere seguito nel confronto tra un sistema Moore-Moore ed uno Mealy-Moore, dal momento che dato un sistema Mealy-Moore, ne esiste sempre uno Moore-Mealy ad esso equivalente, quindi con gli stessi valori di $\overline{n}_i$ e di *T*.

Se $\overline{T}_{e1}$ e $\overline{T}_{e2}$ sono i tempi medi di elaborazione per un sistema Moore-Moore e per un sistema Moore-Mealy (o Mealy-Moore), il rapporto:

$$r = \frac{\overline{T}_{e1}}{\overline{T}_{e2}} = \frac{\overline{N}_1}{\overline{N}_2} \cdot \frac{T_1}{T_2} \qquad (7.9)$$

dove è $\bar{N}_k = \sum_{i=1}^{v} p_i \cdot \bar{n}_{i,k}$, $k = 1, 2$, mostra che il sistema Moore-Moore è più veloce di quello Moore-Mealy (o Mealy-Moore) per $r < 1$, più lento per $r > 1$.