

## Capitolo 1 - Costituenti dei sistemi

### Introduzione.

Il problema di trasferire un livello di tensione che codifica uno dei due elementi della logica booleana da una flip-flop dove è memorizzato ad una funzione logica che lo deve utilizzare o ad un altro flip-flop, risponde ad un criterio molto semplice che trova applicazioni anche in altri settori della Fisica, forse più familiari dell'Elettronica.

Se un serbatoio di acqua ha un rubinetto di uscita dal quale parte un tubo che è collegato ad uno o più serbatoi, ed ognuno di questi ha un rubinetto di ingresso, se si apre il rubinetto del serbatoio di partenza e questo è pieno, l'acqua passa nel tubo ed arriva, con un certo ritardo, ai serbatoi a valle ma entra solo in quelli che hanno il loro rubinetto d'ingresso aperto. Dopo un certo tempo se il serbatoio di partenza è grande e quelli di arrivo molto piccoli, l'acqua raggiunge in tutti i serbatoi aperti lo stesso livello, poco diverso da quello iniziale nel serbatoio grande. E' ovvio che il rubinetto d'ingresso di un serbatoio secondario deve essere chiuso prima che il serbatoio grande cambi il suo "livello".

Come dire che, se il serbatoio grande era vuoto (livello logico "basso"), tutti i serbatoi aperti restano vuoti, se il serbatoio era pieno tutti i serbatoi con il rubinetto di "abilitazione" aperto si riempiono e raggiungono il livello "alto".

In termini booleani un "1" (o uno "0") in uscita da un elemento di memoria si propaga e viene memorizzato in tutti gli elementi abilitati a farlo. Se si tratta di un flip-flop di tipo D con abilitazione a livello, questa va rimossa prima che il flip-flop di partenza cambi stato.

In un sistema in cui tutti gli elementi di memoria abbiano una abilitazione all'uscita ed una all'ingresso, qualsiasi trasferimento di informazioni tra flip-flop collegati fra loro, diventa un gioco temporale di abilitazioni che, come già detto in precedenza, possono essere con funzionamento "a livello" od "a transizione", ciascuna con una doppia possibilità, l'elemento può essere abilitato da un livello alto o basso o da una transizione da basso ad alto o viceversa.

Il discorso è semplice, ma non tanto quanto può sembrare a chi si avvicina al problema con eccessiva superficialità. Come si fa, infatti, a capire se un serbatoio è vuoto perché il rubinetto non è stato mai aperto o non è stato aperto quando c'era acqua nel tubo, oppure perché il livello da trasferire era "basso"?

E' chiaro che perché il livello sia significativo bisogna avere la certezza che le abilitazioni di uscita degli elementi da cui le informazioni provengono siano stati dati con un congruo anticipo rispetto a quelle in ingresso agli elementi che tali informazioni devono ricevere.

Un altro elemento che merita attenzione è che un tubo impegnato a trasferire un livello da un serbatoio A ad uno B non può essere usato anche per trasferire il livello del serbatoio C a quello D anche se esso è collegato in uscita sia ad A che a C ed in ingresso sia a B che a D. Perché se uno dei serbatoi di partenza (grandi) fosse vuoto e l'altro pieno avremmo un unico livello in tutti i serbatoi né alto né basso e, quindi, certamente un errore.

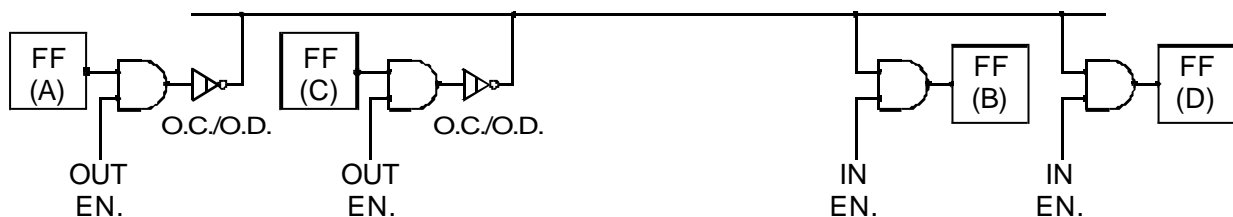


Fig. 1

In termini booleani questa situazione corrisponde allo schema circuitale di figura 1, in cui A, B, C e D sono 4 flip-flop con abilitazione di ingresso e di uscita, collegati ad una stessa linea. A e C hanno

le uscite sulla linea e, pertanto debbono avere uscite Open Collector/Open Drain in *logica negata* od essere dotati di uscite con driver “tristate” per evitare il conflitto elettrico tra due stati logici opposti.

Ovviamente non ci sono problemi a trasferire il contenuto di uno dei due flip-flop, per esempio A, a B e D contemporaneamente.

Nella struttura interna di un sistema complesso, però, il problema ricorrente non è il trasferimento di un singolo bit, ma di un dato numerico od un codice (genericamente una “parola”) costituito da molti bit, di solito, 16, 32, o 64.

Il problema, quindi, riguarda l’interconnessione non di singoli flip-flop ma di quelle strutture multiple che abbiamo chiamato registri.

### Interconnessione tra registri.

Esistono molte soluzioni per trasferire informazioni tra i registri interni di un sistema, o tra registri di sistemi diversi ma tutte le strutture si possono considerare intermedie fra le due scelte estreme, da una parte la connessione indipendente di ciascun registro a qualsiasi altro registro (collegamenti punto a punto) con possibilità quindi di attivare contemporaneamente tutti i trasferimenti che si desiderino, purché non interessino in maniera incompatibile uno stesso registro (parallelismo assoluto), dall’altro una unica interfaccia universale (bus) che consenta la connessione di qualsiasi registro a qualsiasi altro registro (o ad un gruppo di registri) ma con un solo trasferimento per volta (soluzione puramente seriale). La prima corrisponde alla massima complessità hardware, la seconda è improntata ad una grande economia hardware e duttilità ma condiziona fortemente il funzionamento della rete

Nel primo caso la struttura logica che si utilizza intensivamente è il *multiplexer*. La rete si dice di tipo “*mesh*” e si possono effettuare allo stesso tempo tutti i collegamenti. Si può anche leggere e scrivere nello stesso registro, se è del tipo master/slave, così come, ovviamente, si può trasferire lo stesso contenuto da un registro in più registri.

Per rendersi conto del funzionamento di un trasferimento dati da un registro ad un altro basta considerare che, a parte i bit di selezione di eventuali multiplexer di interconnessione, i livelli logici dei registri si propagano lungo le linee di collegamento solo quando è attiva la linea di abilitazione in uscita (spesso del tipo “*tristate*”), mentre un nuovo contenuto può essere registrato solo quando è attiva l’abilitazione in ingresso. Alcuni registri possono avere contemporaneamente attive sia l’abilitazione in ingresso che quella in uscita e possono, quindi, diventare “*trasparenti*”.

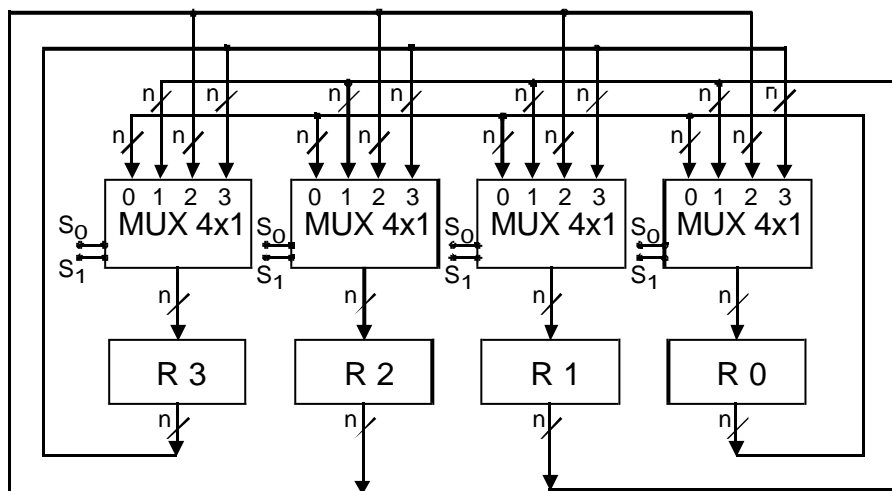


Fig. 2

Se supponiamo di avere quattro registri da  $n$  bit e gli ingressi di ciascuno di essi sono collegati a tutte le uscite da multiplexer “quattro in uno” ad  $n$  bit (figura 2). Questa rete, detta di tipo “*mesh*”, consente di trasferire il contenuto di ciascun registro in qualunque altro, senza interferire con un altro contemporaneo trasferimento che non collida col primo alla destinazione.

Il difetto sostanziale della struttura, che si manifesta sempre di più quando il numero dei registri aumenta, è il numero di funzioni logiche necessarie per realizzarla.

Una soluzione più economica ma che limita il numero di trasferimenti effettuabili allo stesso tempo è mostrato in figura 3.

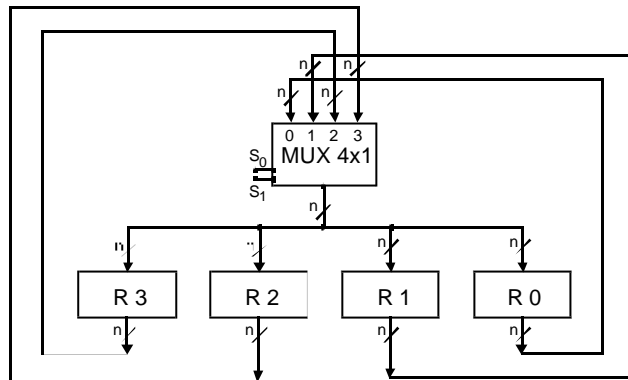


Fig. 3

Una volta che il trasferimento in corso (per esempio,  $R1 \leftarrow R0$ , che si effettua ponendo  $S_0=0$ ,  $S_1=0$  ed abilitando in registrazione  $R1$ ) ha impegnato il multiplexer non è possibile effettuare altre operazioni che coinvolgano la rete a valle del multiplexer ma rimane possibile utilizzare il contenuto dei registri, anche di quello che è la destinazione del trasferimento in corso, se i registri sono del tipo Master/Slave, per eventuali trasferimenti verso registri che fanno parte di altri gruppi (per esempio  $R7 \leftarrow R2$ ).

Questa possibilità è completamente esclusa quando si adotta la modalità di interconnessione più economica che è quella detta a “*bus*” mostrata in figura 4.

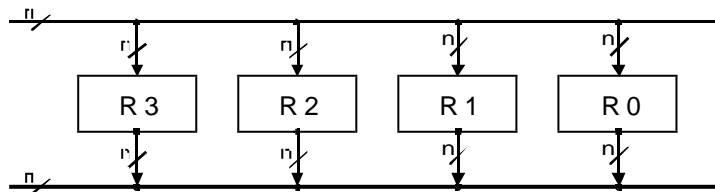


Fig. 4

Un “bus dati, formato da tante linee quanti sono i bit dei registri, collega tra loro ordinatamente sia gli ingressi che le uscite dei registri. Le linee sono bidirezionali e le uscite debbono necessariamente essere del tipo “tristate” oppure “Open Collector” / “Open Drain”.

Nel caso si usino buffer tristate è necessario che la logica di gestione garantisca che non verranno attivate contemporaneamente le uscite di due registri, perché questo riprodurrebbe il conflitto elettrico tra i livelli logici opposti.

L’analisi delle strutture di interconnessione tra registri è solo uno degli aspetti del problema reale perché abbiamo finora ragionato su elementi di memoria strutturalmente asincroni ma dotati di abilitazione di ingresso e/o di uscita e, quindi, sincronizzabili, ma come abbiamo più volte ricordato la stragrande maggioranza delle strutture logiche complesse sono costituite da elementi di memoria sincroni, generalmente flip-flop di tipo D e, di solito, la sincronizzazione è a transizione. Ogni

trasferimento di informazioni avviene, quindi, sotto il controllo di un segnale di clock e tutti gli intervalli di tempo sono misurati in periodi di clock.

In figura 5 è rappresentata la struttura hardware per il trasferimento di una “parola” da un registro “A” “di tipo D “master/slave” sincrono (a transizione da *basso* ad *alto*) ad un registro “B”, anch’esso di tipo D sincrono a transizione.

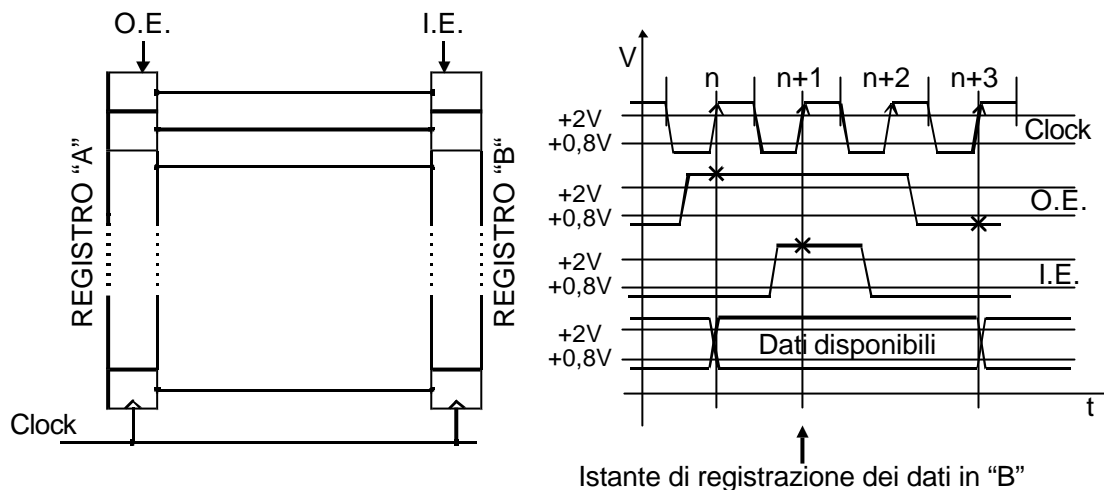


Fig. 5

Come si può rilevare i dati vengono “messi” sulle linee di collegamento quando è attivo il segnale di Output Enable del registro A e contemporaneamente si ha la transizione del clock per cui i flip-flop di A manifestano in uscita il loro contenuto. I dati restano sulle linee finché si ha la prima transizione del clock successiva ad un livello basso di O.E.. La registrazione avviene quando si ha una transizione del clock in corrispondenza dello stato attivo del segnale di I.E. del registro B. I dati sono disponibili dall’istante  $t_n$  a quello  $t_{n+3}$  e la registrazione avviene a  $t_{n+1}$ .

Naturalmente le stesse linee possono proseguire per collegare i bit del registro A ad altri registri senza che questo produca modifiche al “*protocollo*” di trasferimento.

Con la parola “*protocollo*” si intende l’insieme delle *regole logico-temporali* che vanno rispettate perché una certa configurazione hardware effettui correttamente un trasferimento.

Una operazione come quella illustrata in figura 5 può essere efficacemente espressa con una notazione del tipo:

$$t_{n+1}: RB \leftarrow RA$$

Questa notazione è presa da un vero e proprio “linguaggio” che è stato messo a punto negli anni per descrivere operazioni all’interno dei sistemi di elaborazione.

Questo linguaggio che in inglese è detto: *Register Transfer Language (RTL)* ha delle regole sintattiche molto precise, alcune delle quali sono riportate nella seguente tabella

Simboli	Descrizione	Esempi
Lettere (associate a numeri)	Indicano un registro	$PC, R2, ACC, IR$
Lettere o numeri tra parentesi	Indicano parti di registri	$R2(1), R2(7:0), AR(L)$
Freccia verso sinistra	Indica un trasferimento dati	$R1 \rightarrow R2$
Virgola	Separa operazioni contemp.	$R1 \rightarrow R2, R2 \rightarrow R1$
Parentesi quadre	Indicano un indirizzo di memoria	$DR \rightarrow M[AR]$

Tab. 1

Gli esempi di sigle letterali usate nella tabella per indicare alcuni registri non sono scelte a caso, perché **PC** sta per “*Program Counter*”, **ACC** sta per “*Accumulatore*” che, come vedremo in seguito sono registri importanti di un sistema di elaborazione.

Molto importanti sono anche le notazioni usate per indicare parti (1 byte) di registri a molti più bit, infatti, come si vede in tabella la parte che contiene il byte meno significativo di un registro a 16 o a 32 bit può essere indicata con un indice (1) o facendo esplicito riferimento ai bit (0:7), oppure usando (L) che sta appunto per “*Lower*” che in inglese significa “il più basso”.

La freccia verso sinistra, indica in maniera trasparente il trasferimento dati.

In tabella 2 sono indicate le notazioni per le operazioni logiche ed aritmetiche possibili tra i contenuti di due registri.

Simbolo	Descrizione
$R3 \leftarrow R1 + R2$	Trasferimento in R3 della somma del contenuto di R1 ed R2;
$R3 \leftarrow R1 - R2$	Trasferimento in R3 della differenza tra il contenuto di R1 e quello di R2;
$R2 \leftarrow R2^*$	Complementazione del contenuto di R2 (complemento ad 1);
$R2 \leftarrow R2^{*+1}$	Negazione del contenuto di R2 (complemento a 2);
$R3 \leftarrow R1 + R2^{*+1}$	In R3 va la differenza fra R1 ed R2 (sottrazione);
$R1 \leftarrow R1 + 1$	Incremento di una unità del contenuto di R1;
$R1 \leftarrow R1 - 1$	Decremento di una unità del contenuto di R1;

Tab. 2

Una rete logica in grado di effettuare l'operazione aritmetica descritta in RTL con la notazione:

Simbolo	Descrizione
$R3 \leftarrow R1 + R2$	Trasferimento in R3 della somma del contenuto di R1 ed R2

è riportata in figura 6.

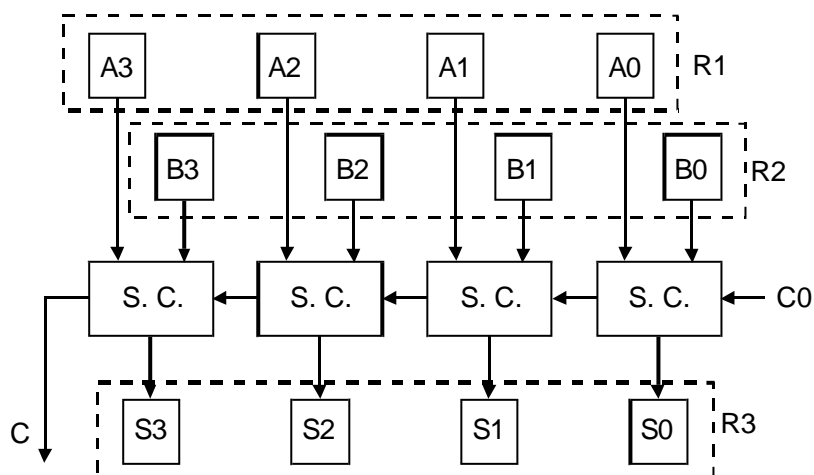


Fig. 6

Si tratta di un sommatore binario “ripple clock” a 4 bit collegato a 2 registri a 4 bit. La struttura modulare consente di costruire un sommatore ad un numero di bit multiplo di quattro utilizzandone in cascata più esemplari.

Analogamente l'operazione RTL seguente,

Simbolo	Descrizione
$R3 \leftarrow R2 - R1$	Trasferimento in R3 della differenza tra il contenuto di R2 e quello di R1

Può essere realizzata dalla rete booleana riportata in figura 7.

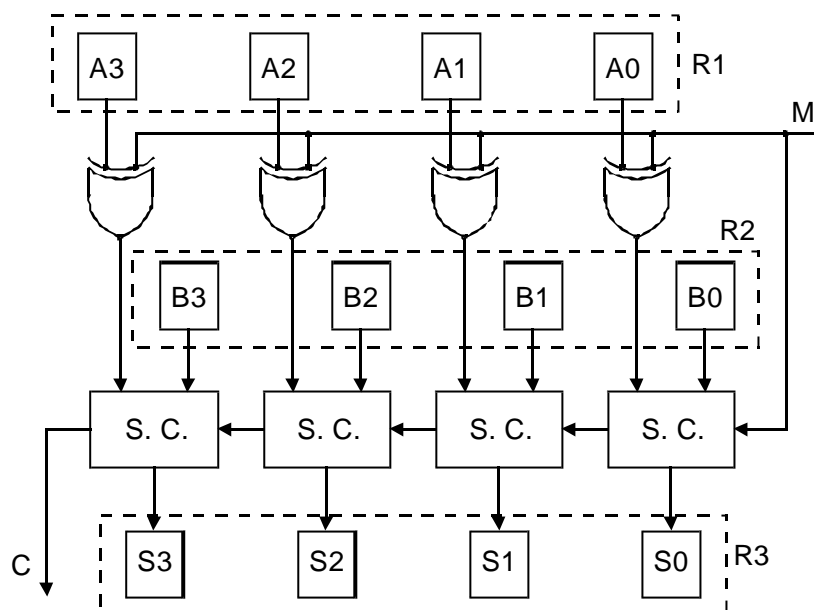


Fig. 7

Basta considerare che ponendo  $M=1$  si ottiene che attraverso gli XOR arriva alla batteria di sommatori completi il “complemento ad 1” del contenuto di R1 e l’inserimento di un riporto “1” nel sommatore dei bit meno significativi lo trasforma nel “complemento a 2”.

Analogamente l’operazione RTL,

**Simbolo**

$R1 \leftarrow R1 + 1$

**Descrizione**

Incremento di una unità del contenuto di R1;

è realizzata dalla rete di figura 8.

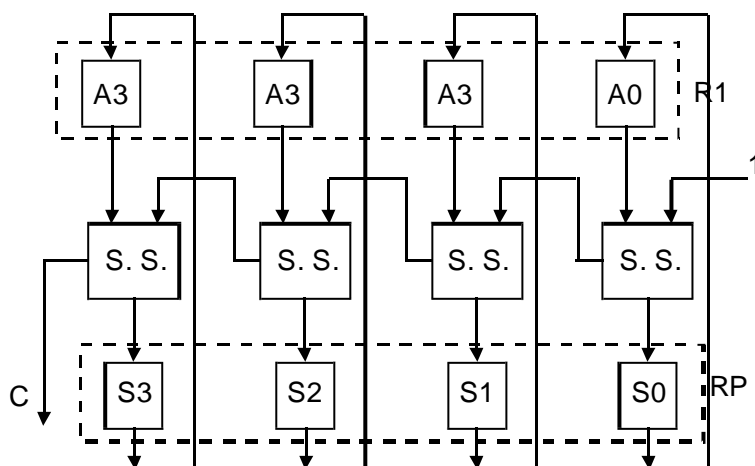


Fig. 8

È piuttosto evidente che tutte le reti che possono eseguire le varie operazioni aritmetiche presentate attraverso la loro notazione RTL sono impostate sull’uso di sommatori completi, è naturale, quindi chiedersi se non è possibile costruire un’unica rete in grado di eseguire tutte le operazioni.

La rete che soddisfa a questa richiesta è riportata in figura 9.

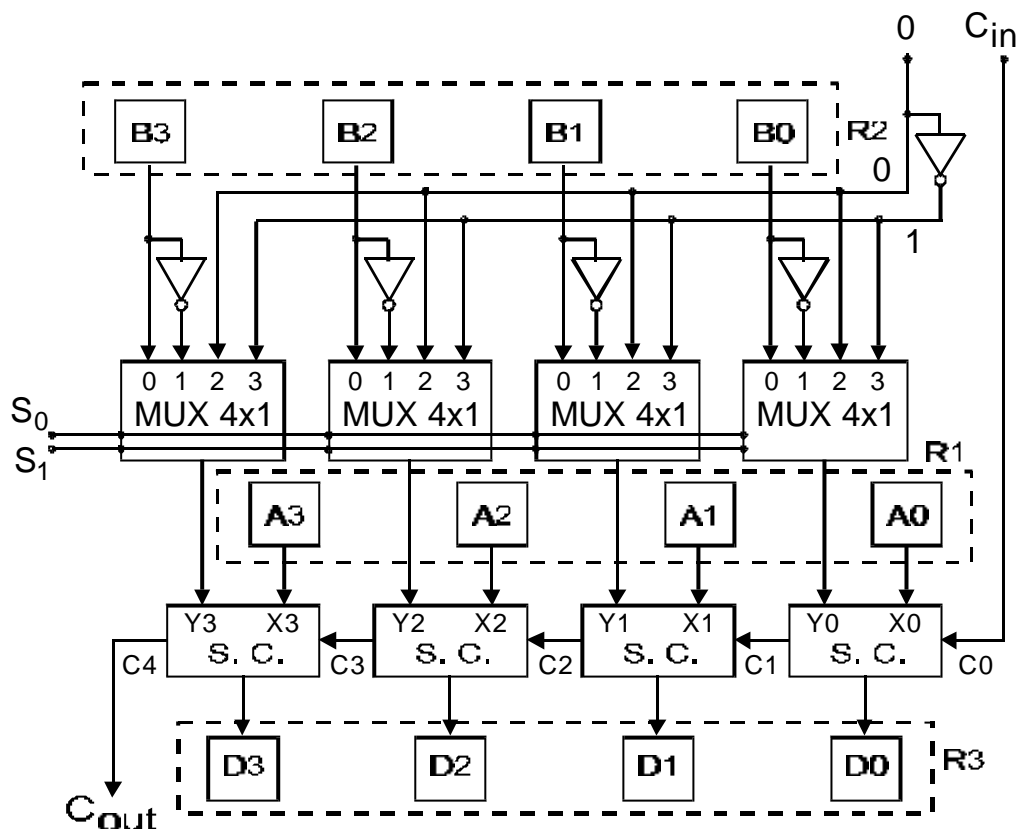


Fig. 9

La tabella dei bit di selezione (o di controllo) che corrispondono alle varie operazioni sono riassunte in tabella 3.

Selezione			Ingresso Y	Uscita $D = A + Y + C_{in}$	Microoperazioni
S1	S0	Cin			
0	0	0	B	$D = A + B$	Somma
0	0	1	B	$D = A + B + 1$	Somma con riporto
0	1	0	$B^*$	$D = A + B^*$	Sottrazione con "prestito"
0	1	1	$B^*$	$D = A + B^* + 1$	Sottrazione
1	0	0	0	$D = A$	Trasferimento di A
1	0	1	0	$D = A + 1$	Incremento di A
1	1	0	1	$D = A - 1$	Decremento di A
1	1	1	1	$D = A$	Trasferimento di A

Tab. 3

La combinazione di valori logici dei tre bit S0, S1 e Cin determina automaticamente l'operazione che la rete esegue, prelevando i due operandi dal registro A e da quello B, per le operazioni a due operandi e solo da quello A, per le operazioni ad un solo operando.

Il circuito aritmetico a 4 bit di figura 9 può essere immaginato costituito da 4 elementi modulari singoli con i quali, in linea di principio si possono costruire funzioni aritmetiche ad un qualsiasi numero di bit.

In figura 9 bis è stato appunto isolato un elemento identico a quelli usati nella figura precedente. E' stato soltanto adattato graficamente. Esso ha due ingressi di controllo S1 ed S2, tre ingressi Ai, Bi e Cin e due uscite, Di e Cout.

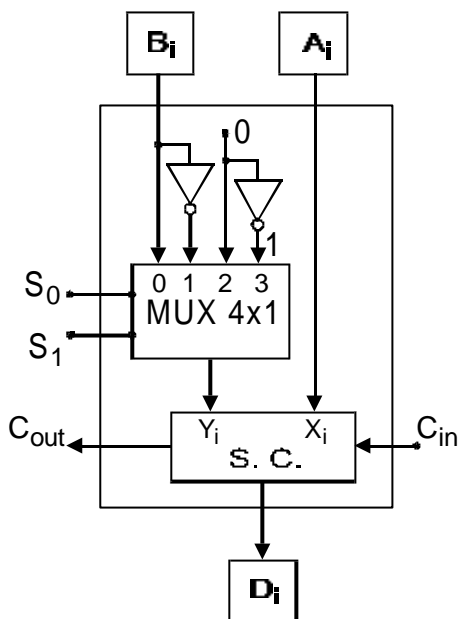


Figura 9 bis

Un sistema di elaborazione che possa fare qualunque operazione definibile matematicamente deve poter effettuare anche tutte le operazioni logiche a due operandi.

In figura 10 sono riportate le 16 combinazioni d'uscita delle funzioni booleane a due bit. Un rapido esame ci consente di concludere che eliminate le funzioni degeneri e tenuto conto della simmetria tra le variabili che può sempre essere garantita dallo scambio delle combinazioni logiche delle due variabili d'ingresso le funzioni veramente distinte sono solo 4, AND, OR, XOR e NOT.

NOME	FUNZIONE	Cost=0	AND	A and B*	F = A	A*and B	F = B	A xor B	A or B	(A or B)*	(A xor B)*	F = B*	A or B*	F = A*	A*or B	(A and B)*	Cost=1
X	Y	F <sub>0</sub>	F <sub>1</sub>	F <sub>2</sub>	F <sub>3</sub>	F <sub>4</sub>	F <sub>5</sub>	F <sub>6</sub>	F <sub>7</sub>	F <sub>8</sub>	F <sub>9</sub>	F <sub>10</sub>	F <sub>11</sub>	F <sub>12</sub>	F <sub>13</sub>	F <sub>14</sub>	F <sub>15</sub>
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Fig. 10

Un elemento modulare di rete in grado di effettuare nelle ipotesi introdotte, qualunque operazione logica è mostrata in figura 11.

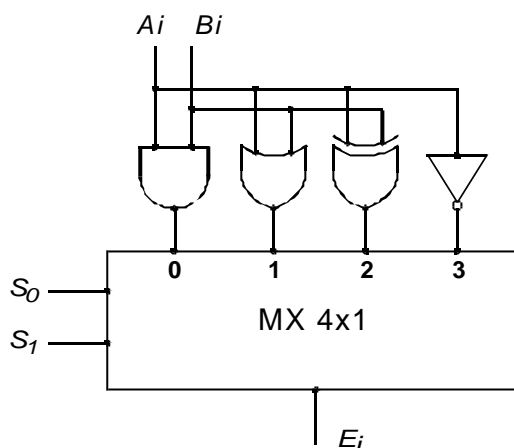


Fig. 11

Bisognerà, ovviamente, usare tanti elementi quanti sono i bit della parola del sistema. Gli ovvii accoppiamenti tra i codici di selezione del multiplexer e l'operazione effettuata sono nella tabella 4.

$S_0$	$S_1$	Uscita	Operazione
0	0	$E = A \cdot B$	AND
0	1	$E = A + B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A \cdot B}$	NOT

Tab. 4

In realtà il quadro delle operazioni che un sistema generale deve essere in grado di effettuare non è completo perché mancano le operazioni di "scorrimento" (o shift), sulla cui indispensabilità non ci possono essere dubbi, basta pensare che senza scorrimento non si possono effettuare operazioni di moltiplicazione e divisione.

I simboli RTL delle operazioni di scorrimento sono in tabella 5.

Simbolo	Descrizione
$R \leftarrow \text{shl } R$	Scorrimento a sinistra
$R \leftarrow \text{shr } R$	Scorrimento a destra
$R \leftarrow \text{cil } R$	Scorrimento circolare a sinistra
$R \leftarrow \text{cir } R$	Scorrimento circolare a destra
$R \leftarrow \text{ashl } R$	Scorrimento aritmetico a sinistra
$R \leftarrow \text{ashr } R$	Scorrimento aritmetico a destra

Tab. 5

Il problema hardware è facilmente risolvibile con un circuito, anch'esso basato su multiplexer che è rappresentato in versione modulare a 4 bit in figura 12.

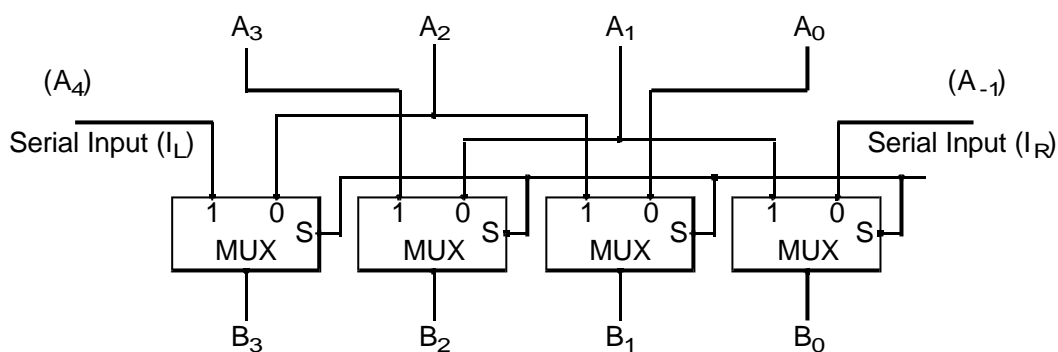


Fig. 12

La tabella 6 associa i due codici di selezione alle due possibili operazioni e da le corrispondenti uscite.

Selezione	Uscite				
	S	B0	B1	B2	B3
0	$I_R$	$A_0$	$A_1$	$A_2$	
1	$A_1$	$A_2$	$A_3$	$I_L$	

Tab. 6

Le tre strutture booleane sintetizzate risolvono ciascuna uno dei compiti di una cosiddetta Unità Aritmetica e Logica (ALU), cioè della parte di un sistema di elaborazione che materialmente esegue le operazioni logiche ed aritmetiche. Perché tutte e tre le funzionalità siano presenti in un'unica struttura bisogna combinarle in maniera efficiente. La struttura di figura 13 rappresenta un elemento modulare di ALU che incorpora un elemento aritmetico come quello di figura 9 bis, un elemento logico come quello di figura 11 e la tecnica di collegamento utilizzata in figura 12.

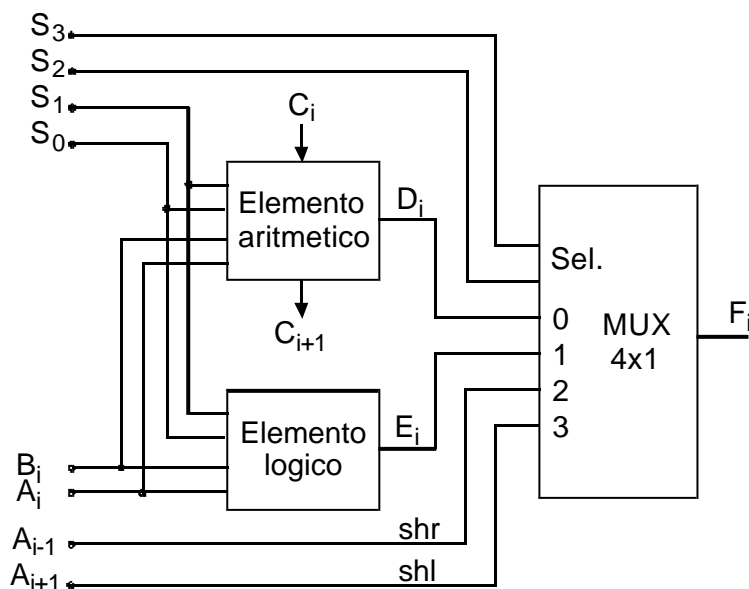


Fig. 13

Il risultato è una rete che attraverso un opportuno uso dei codici operativi a quattro bit  $S_0$ - $S_3$  determina l'operazione logica o aritmetica che viene effettuata sugli ingressi.

### Generalità sulla struttura di un sistema di elaborazione.

Un “*sistema di elaborazione*”, è una struttura logica in grado di eseguire su dei “*dati*” in ingresso operazioni logiche od aritmetiche ottenendo dei “*risultati*”.

Le operazioni da effettuare sono individuate attraverso codici numerici detti “*istruzioni*”.

Un sistema di questo tipo si può facilmente immaginare scomposto in due strutture, una parte “*operativa*” che esegue le sequenze di operazioni previste per trasformare i “*dati*”,  $X$  nei “*risultati*”,  $Z$  ed una parte di “*controllo*” che interpreta le “*istruzioni*” trasformandole in “*comandi*”,  $a$  che sono funzione anche delle “*condizioni*”  $b$  attraverso le quali la rete operativa comunica a quella di controllo lo stato di avanzamento della elaborazione.

Entrambe le parti del sistema sono reti sequenziali sincrone il cui funzionamento è cadenzato dallo stesso “*clock*”.

Una “*istruzione*” è una operazione del sistema esprimibile in una sequenza di **microoperazioni** (***μ*operazioni**) che per questo motivo viene detta “***μ*sequenza**”

Ogni ***μ*operazione** può impegnare diverse parti del sistema ed essere quindi costituita di un insieme di operazioni singole da eseguire in un certo intervallo di tempo.

In linea di massima una ***μ*sequenza** esprime la successione delle azioni elementari della parte “*operativa*” del sistema in cui la parte di “*controllo*” traduce ciascuna istruzione.

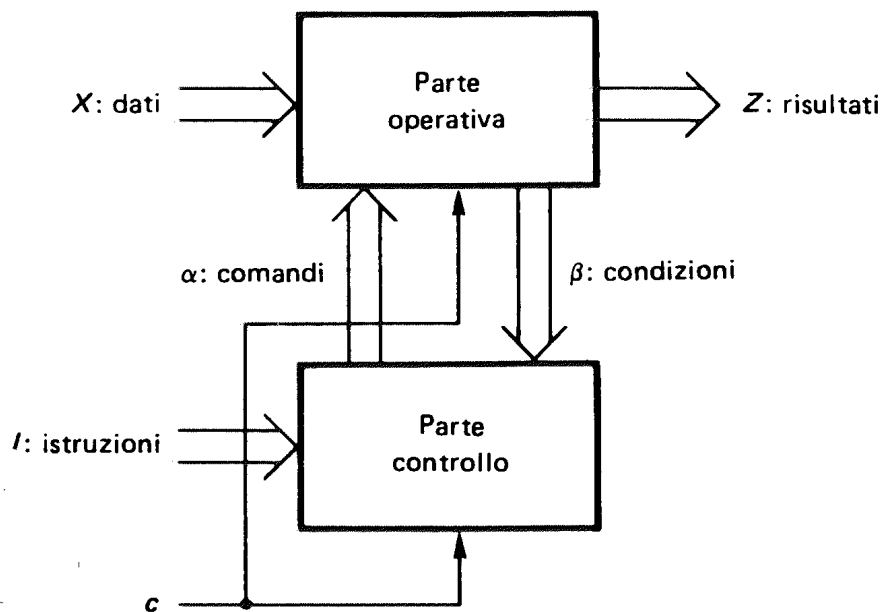


Fig. 14

La parte “*operativa*” del sistema è composta da moduli combinatori, ciascuno in grado di eseguire un certo repertorio di  $\mu$ operazioni logiche, aritmetiche e/o di trasferimento. L’esecuzione di una  $\mu$ operazione da parte dell’unità operativa del sistema richiede in generale alla parte di **controllo** la generazione di una sequenza di segnali di **comando** sul cui sviluppo agiscono le “**condizioni**” che ad ogni periodo di clock vengono trasmesse dalla parte operativa. Tutte le operazioni sono abilitate da appositi bit ma sincronizzate dal clock.

L’insieme dei comandi necessari per eseguire una ***μ*operazione** è detto “***μ*istruzione**”.

La successione di comandi costituenti una **istruzione** gestiranno il funzionamento dei moduli combinatori della parte operativa coinvolti nella **operazione**, come ALU, multiplexer, registri a scorrimento, ecc..., abilitando i dati in uscita dai registri sorgente, selezionando l'operazione da effettuare ed abilitando in lettura il registro di destinazione, il tutto scandito dal periodo di clock.

I “**comandi**” non sono altro che i codici numerici che abbiamo visto associati a ciascuna operazione logica, aritmetica e di trasferimento nei vari esempi di reti di interconnessione tra registri e blocchi logici combinatori, considerate per introdurre il linguaggio RTL.

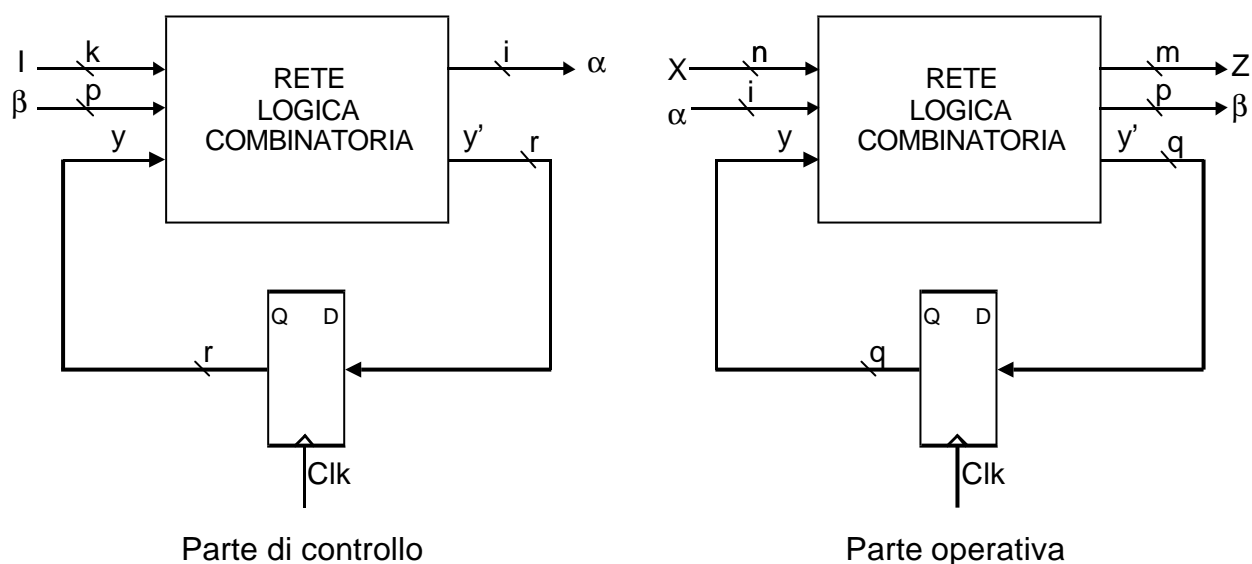


Fig. 15

Abbiamo già detto che in generale sia la **parte operativa** che quella di **controllo** sono reti sequenziali. Entrambe, quindi, se realizzate secondo il modello degli automi a stati finiti di tipo sincrono avranno una struttura simile. Le due varianti che il modello assumerà nei due casi sono in figura 15:

Il prossimo ciclo di lezioni sarà dedicato allo sviluppo di esempi di sistemi di elaborazione con parte operativa e parte di controllo di complessità crescente.

Il ciclo successivo, nel prossimo semestre, sarà dedicato allo sviluppo di una struttura di elaborazione semplice ma completa.