

SISTEMI INFORMATICI

2.0.1 Operazioni tra numeri binari

Nelle macchine digitali i valori numerici sono rappresentati da un *numero fisso di cifre*. Si parla di **numeri in lunghezza fissa**. Così, ad esempio, il valore tre, in una macchina che elabora numeri a quattro cifre binarie è rappresentato da **0011**, in una macchina che elabora numeri ad otto cifre è rappresentato da **00000011**.

Si ricorda che questi valori corrispondono, all'interno dei circuiti in cui sono rappresentati, a due insiemi ordinati di tensioni elettriche, ad esempio del seguente tipo:

0 V 0 V 5 V 5 V → tre rappresentato in quattro cifre

0 V 0 V 0 V 0 V 0 V 0 V 5 V 5 V → tre rappresentato in otto cifre

dove 0V e 5V rappresentano, in logica positiva, rispettivamente, il valore basso (0 logico) ed il valore alto (1 logico) di una tensione binaria. Nel seguito i due livelli di tensione binaria saranno rappresentati dai corrispondenti livelli logici

L'operazione di somma di due numeri binari, di due numeri a quattro cifre si ottiene con un circuito del tipo rappresentato in **Fig.2.0.1**.

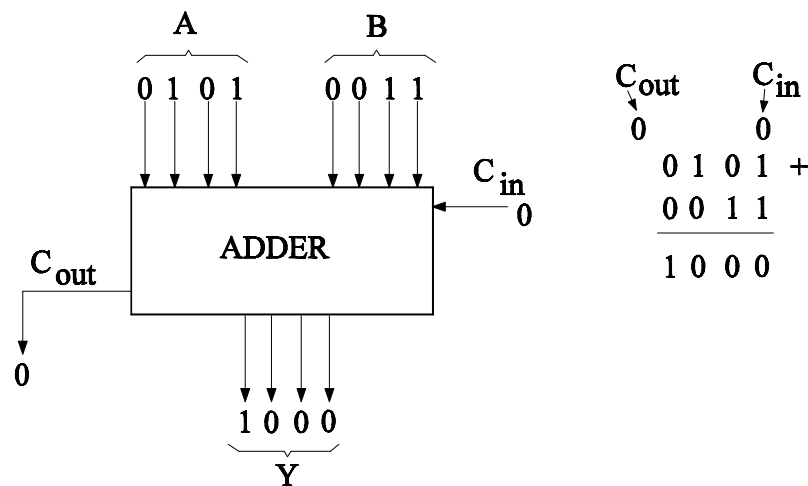


Fig.2.0.1. Circuito che esegue la somma binaria $Y=A+B$ di due numeri a 4 cifre.

Si verifica facilmente che i valori A e B rappresentano i valori 5 e 3 rispettivamente. Il risultato Y rappresenta correttamente il valore 8. C_{out} e C_{in} rappresentano, rispettivamente, il riporto sulla somma delle cifre più significative ed il riporto iniziale prodotto dalla somma precedente di cifre meno significative. Un circuito come quello rappresentato può essere realizzato con quattro **Full Adder** con collegamento **ripple carry**.

Il vincolo di avere dati in lunghezza fissa ha come conseguenza che, in alcuni casi, pur eseguendo la somma in modo corretto, il risultato può essere errato. Ciò avviene quando il risultato supera il massimo valore rappresentabile nella lunghezza prevista per i numeri elaborati. Nel caso mostrato in **Fig.2.0.1**, quando il risultato della somma supera il massimo valore rappresentabile in quattro cifre, $1111_{\text{binario}} = 15_{\text{decimale}}$, si genera un riporto nella somma delle cifre più significative. Il risultato corretto avrebbe bisogno di cinque cifre, evidentemente la lettura di sole quattro cifre fornisce un valore errato. Si parla in questo caso di **errore di superamento o di overflow**.

Se la cifra di riporto C_{out} viene memorizzata in una **cella di memoria** si può utilizzare il contenuto di questa cella come segnalatore di overflow detta **carry flag** o più sinteticamente **C-Flag**.

2.0.2 Codice in complemento a due

Per fare la differenza di due numeri binari si può ancora usare un Adder. Infatti, per numeri binari naturali (senza segno) in lunghezza quattro cifre vale sempre la relazione:

$$B + \bar{B} + 1 = 10000$$

(il numero 10000 va letto nel codice binario, per cui rappresenta sedici decimale).

Esempio (sottintendendo numeri in binario):

$$0011 + 1100 + 1 = 10000$$

Si comprende facilmente che si arriva alla stessa conclusione qualunque sia il numero di cifre di A.

Il valore $\bar{B}$, ottenuto da B scrivendo la cifra 1 dove la cifra di B è zero e scrivendo 0 dove la cifra di B è 1, è detto **complemento a uno** di B.

La somma $\bar{B} + 1$ è detta **complemento a due** di B.

Ricavando $-B$ dalla prima relazione si ha:

$$(2.0.1)$$

quindi la differenza tra il valore A ed il valore B si può scrivere:

$$-B = \bar{B} + 1 - 10000$$

$$A - B = A + \bar{B} + 1 - 10000$$

ovvero:

$$(10000 + A) - B = A + \bar{B} + 1 \quad (2.0.2)$$

La differenza tra il valore a cinque cifre ($10000 + A$) con il valore B può sempre essere eseguita, anche quando $A < B$, dato che il minuendo, a cinque cifre significative, è sempre maggiore del sottraendo.

Si comprende che nel caso sia $A \geq B$ il risultato, a cinque cifre, ha la cifra più significativa pari a 1, mentre nel caso sia $A < B$ la cifra più significativa del risultato è 0. Ad esempio, posto $A=1011$ $B=0110$, si ha: $(10000+1011)-0110 = 10101$; con $A=0111$, $B=1100$, si ha: $(10000+0111)-1100 = 01011$.

Nel primo caso il risultato, in quattro cifre ($11-6=5$), è esatto. Ciò, correttamente segnalato dal fatto che la quinta cifra del risultato è 1, indica che non è stato necessario alcun **prestito** sulla quarta cifra del minuendo per completare la differenza. Nel secondo caso il risultato, in quattro cifre, è errato ($7-12 \neq 11$), come segnalato dal valore 0 della quinta cifra del risultato. La necessità di un **prestito** alla quarta cifra del minuendo, per completare la differenza, spiega questa seconda conclusione.

La relazione (2.0.2) mostra che si ha lo stesso risultato con sommando ad A il complemento a due di B. Nel primo caso $1011+1001+1 = 10101$; nel secondo caso $0111+0011+1 = 01011$.

In conclusione si può affermare che, adottando il codice in complemento a due del sottraendo (B), **l'operazione di differenza può essere ricondotta ancora ad una somma**.

Ne deriva che la macchina digitale ADDER, descritta nel paragrafo precedente, può essere usata tanto per fare somme quanto per fare differenze.

2.0.3 Macchina digitale programmabile che esegue somme e differenze

Si è vista che con un ADDER si possono fare sia somme che differenze purché in quest'ultimo caso il sottraendo venga sostituito con il suo complemento a due. La macchina che si può utilizzare è quella di **Fig.2.0.2**.

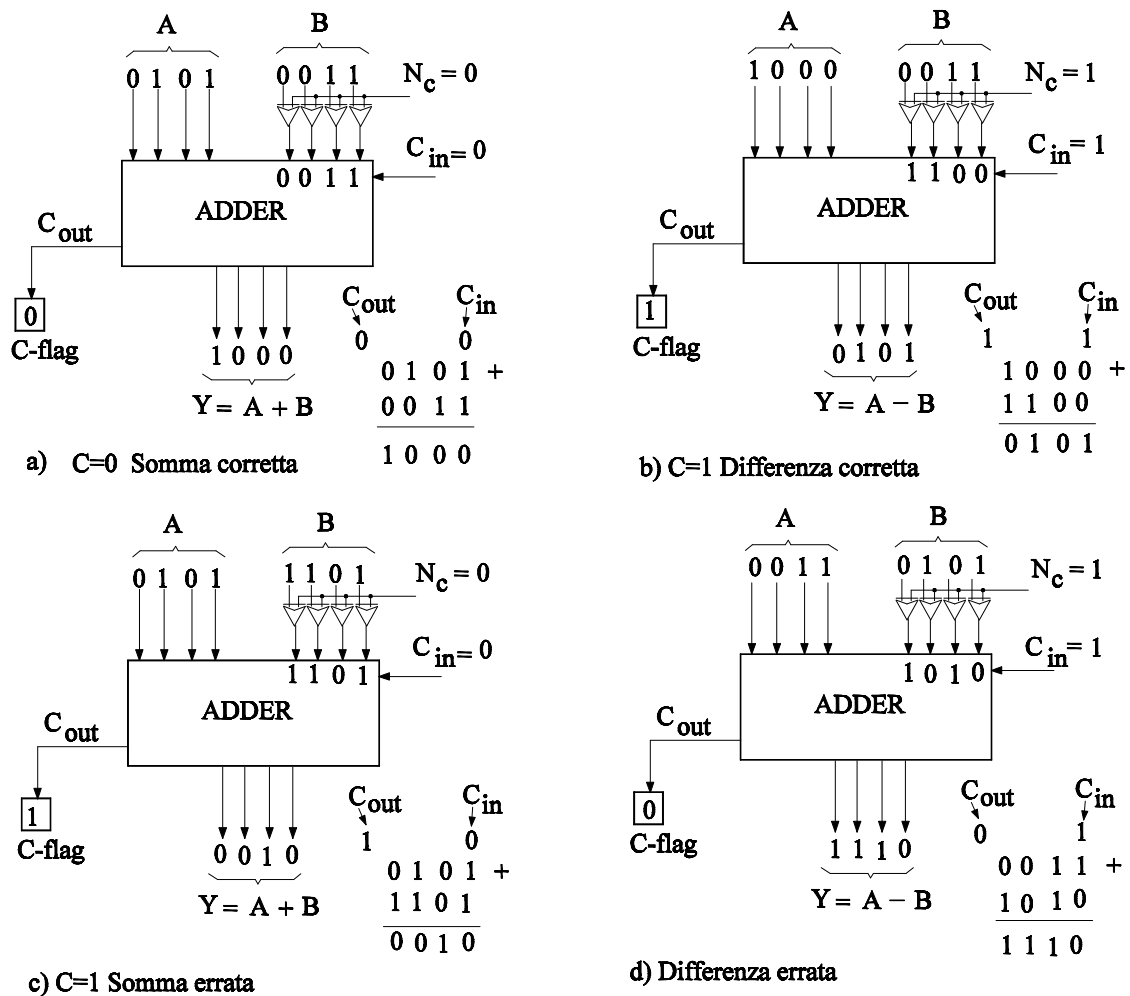


Fig.2.0.2

La **Fig.2.02** mostra come il complemento a 1 di B si ottenga ponendo a 1 il bit N_c , che agisce su un ingresso comune di 4 porte OR esclusivo. Il complemento a due richiede la somma di una unità al complemento a uno per cui anche il valore di C_{in} è posto a 1.

Volendo $C\text{-flag}=0$ con somma o differenza corretta e $C\text{-flag}=0$ con somma e differenza errate, il circuito dell'Adder deve essere modificato nel modo mostrato in **Fig.2.0.3**.

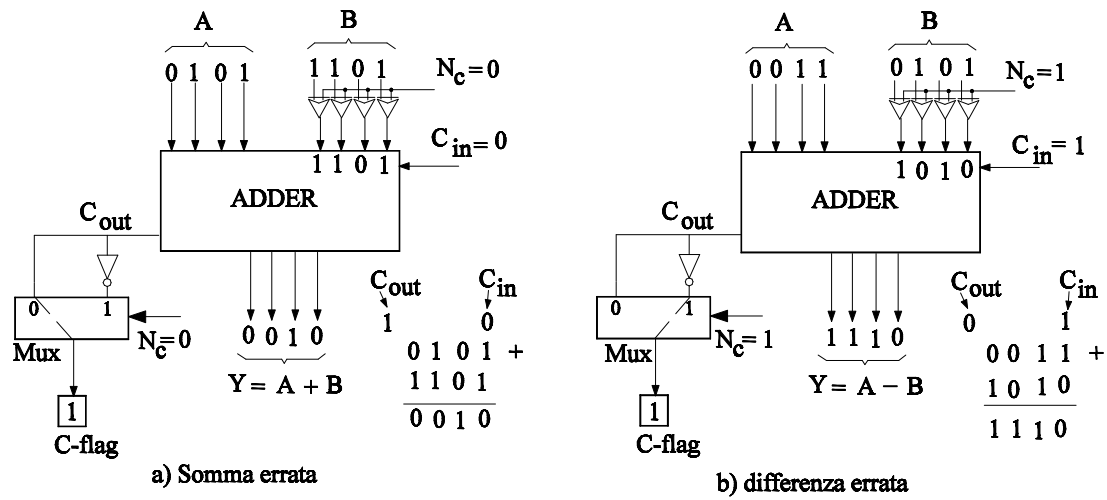


Fig.2.0.3

Gli ultimi due circuiti (**Fig.2.0.2** e **2.0.3**) hanno la proprietà di essere *programmabili*. Agendo sui bit N_c e C_{in} è possibile programmare l'operazione di somma ($N_c = C_{in} = 0$) o di differenza ($N_c = C_{in} = 1$). I due bit, N e C_{in} , sono chiamati **bit di controllo**.

2.0. 4 Rappresentazione dei numeri con segno (numeri relativi)

Si possono rappresentare numeri con segno usando il **codice in modulo e segno**.

Con questo codice si “sacrifica” la cifra più significativa per rappresentare il segno piuttosto che una quantità numerica.

Senza perdita di generalità, si può far riferimento a un numero a quattro cifre binarie. La cifra più significativa, invece che rappresentare un valore di peso $2^3=8$, rappresenta il segno “+”, quando vale 0, il segno “-“, quando vale 1.

In virtù di questa convenzione con numeri a quattro bit si possono rappresentare i seguenti valori:

<i>decimale</i>	<i>modulo e segno</i>	<i>complemento a due</i>
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
-0	1000	-----
-1	1001	1111
-2	1010	1110
-3	1011	1101
-4	1100	1100
-5	1101	1011
-6	1110	1010
-7	1111	1001
-8	-----	1000

La rappresentazione in modulo e segno ha il pregio di essere leggibile in modo semplice una volta che si è capito che i simboli 0 e 1 in posizione più a sinistra non rappresentano un valore ma solamente il segno.

Peraltro questa rappresentazione risulta poco conveniente dal punto di vista della sua utilizzazione nelle operazioni di somma algebrica. Infatti la somma algebrica si traduce in una somma o in una differenza a seconda che gli operandi siano concordi o discordi. Il circuito che effettua questa operazione dovrebbe discriminare le due situazioni, esaminando preliminarmente i segni (con circuiti logici aggiuntivi) e poi programmando l'operazione. Infine, anche l'assegnazione del segno al risultato crea problemi, dato che, nel caso di numeri discordi, il segno del risultato è uguale a quello con modulo maggiore. Servono altri circuiti capaci di esaminare i moduli per completare l'operazione di somma algebrica.

Questa operazione può essere notevolmente semplificata, con notevole economia e semplicità circuitale, se si adotta la rappresentazione dei numeri negativi in complemento a due con la regola indicata dalla relazione (2.0.1). Infatti la somma algebrica dei due valori $(+A) + (-B)$, eseguita in algebra come una differenza tra i moduli, rimane una somma, $(+A) + (B_{\text{comp}2})$, se si accetta di rappresentare $-B$ nel suo complemento a due.

La codificazione di numeri relativi in complemento a due per numeri a quattro bit prevede di dividere tutte le combinazioni di quattro bit ($2^4=16$) in due gruppi di otto numeri: il primo gruppo, formato da tutti i numeri che hanno la cifra più significativa a zero, rappresenta i numeri positivi da 0 a +7, il secondo gruppo, formato da tutti i numeri che hanno la cifra più significativa a uno, rappresenta i numeri negativi da -1 a -8.

I numeri positivi vengono scritti esattamente come i numeri senza segno (numeri naturali), i numeri negativi vengono scritti con la regola indicata dalla (2.0.1). Così, se si vuol rappresentare il numero -3 , si scrive prima il valore $B = +3_{\text{dec}} = 0011_{\text{bin}}$ dopodiché si valuta

$$-B = B_{\text{comp}2} = \bar{B} + 1 = 1100 + 1 = 1101$$

Tutti i 16 valori di questo codice sono scritti nell'ultima colonna della tabella riportata in precedenza.

Se si utilizza questo codice le somme algebriche possono essere realizzate con lo stesso circuito utilizzato nel caso precedente, ponendo semplicemente agli ingressi A e B la rappresentazione dei numeri relativi in complemento a due.

Leggendo con questo codice le operazioni indicate in **Fig.2.0.3**, si trae la conclusione che: nel caso a) gli ingressi sono $A=+5$, $B=-3$, nel caso b) $A=+3$, $B=+5$. Si deduce inoltre che i risultati delle due operazioni programmate, $Y=A+B=(+5)+(-3)=+2$ nel caso a) e $Y=A-B=(+3)-(5)=-2$ nel caso b), sono entrambe corrette. Le stesse rappresentazioni binarie rappresentano dati diversi, a seconda del codice di lettura. Inoltre il **C-flag** che segnala risultati errati, non è più attendibile. Come mai? Si ricorda che la rappresentazione di numeri con segno nel codice in complemento a due prevede di usare le cifre più significative per rappresentare il segno (0 positivo, 1 negativo) mentre le cifre rimanenti rappresentano il modulo. In questo contesto lo scopo di **C-flag** di segnalare eventuali overflow perde di significato.

E' necessario prevedere un nuovo segnalatore che indichi la correttezza o meno del risultato, quando questo supera i massimi valori rappresentabili in una certa lunghezza del dato. Per numeri a quattro cifre si ha **errore di overflow** quando il risultato supera $+7$, **errore di underflow** quando il risultato è inferiore a -8 . Queste condizioni di errore danno un risultato che, letto in codice complemento a due, appare immediatamente assurdo, e corrisponde alle seguenti due possibilità (le "x" indicano "qualunque cifra"):

0 x x x +	1 x x x +
0 x x x	1 x x x +
1 x x x	0 x x x

Si osserva, infatti, che la somma di numeri concordi (entrambi positivi o entrambi negativi) con risultato discorde (rispettivamente negativo e positivo), è una situazione assurda nel campo dell'algebra dei numeri relativi (il risultato deve essere concorde con gli operandi). Per tutte le altre combinazioni delle cifre più significative dei tre valori coinvolti nella somma, il risultato è sempre corretto.

Si può prevedere la costruzione di un circuito che, essendo sensibile ai tre bit considerati, dia risultato 1 solo nei casi rappresentati, segnalando la presenza dell'errore di overflow.

La **Fig.2.0.4** mostra questo il circuito dell'Adder precedente con l'aggiunta della segnalazione di overflow nel caso gli operandi siano letti come numeri con segno codificati in complemento a due. In questa figura è mostrato che gli elementi circuitali che immagazzinano i segnalatori aritmetici sono dei flip flop (celle di memoria). La memorizzazione in ogni flip flop del relativo segnalatore è controllata dal bit **af** (attiva flag). **af**=1 abilita la memorizzazione, **af**=0 lascia nei flip flop i valori memorizzati nell'operazione precedente

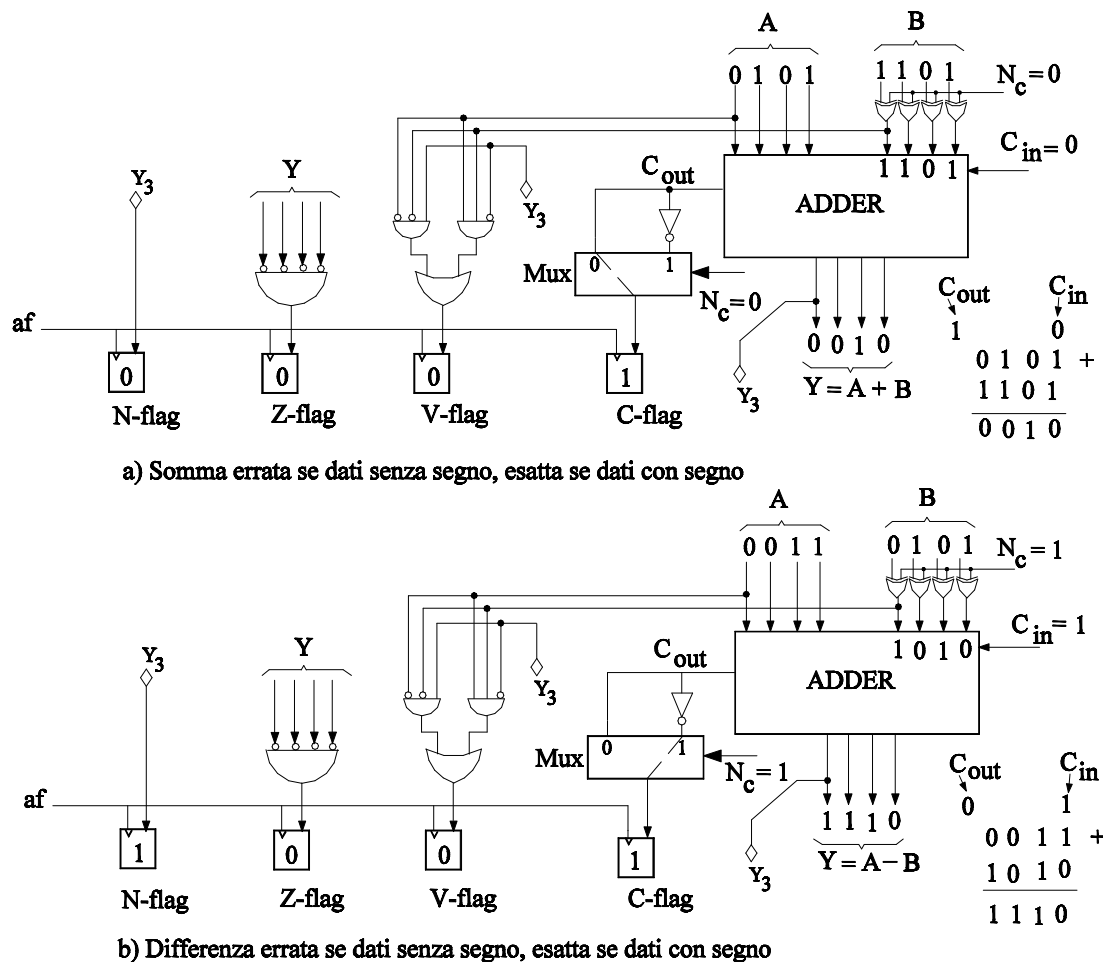


Fig.2.0.4

Nel circuito di **Fig.2.0.4** si sono aggiunti altri due segnalatori: **Z-flag** (segnalatore di zero) che assume il valore uno, se il risultato **Y** vale zero, ed **N-flag** che ricopia il bit del segno (**Y₃**). Dopo un'operazione di somma algebrica è utile, innanzitutto, sapere se il risultato è corretto, sia per numeri senza segno (**C-flag**) che per numeri con segno (**V-flag**). In caso di risultato corretto, può essere utile sapere se il risultato è zero o meno (**Z-flag**), e, per numeri con segno, può essere utile sapere se il risultato è positivo o negativo (**N-flag**). Questi quattro valori vengono indicati globalmente con il termine di *segnalatori aritmetici*.

Si osserva che la macchina di **Fig.2.0.4** elabora i dati di ingresso senza dare ad essi né l'interpretazione di numeri con segno né di numeri senza segno, in quanto il meccanismo con cui vengono elaborati i dati è sempre lo stesso. A seconda di come si vogliono interpretare i dati e i risultati cambia il sottoinsieme di segnalatori aritmetici che si debbono considerare. Per numeri senza segno, **Z** e **C** flag danno utili informazioni sul risultato. Per numeri con segno, sono invece i segnalatori **N**, **Z** e **V** flag a dover essere considerati per avere notizie sul risultato.

Infine si osserva che, nel seguito lo schema della macchina di Fig.2.0.4 viene semplificato in quello di **Fig.2.0.5**.

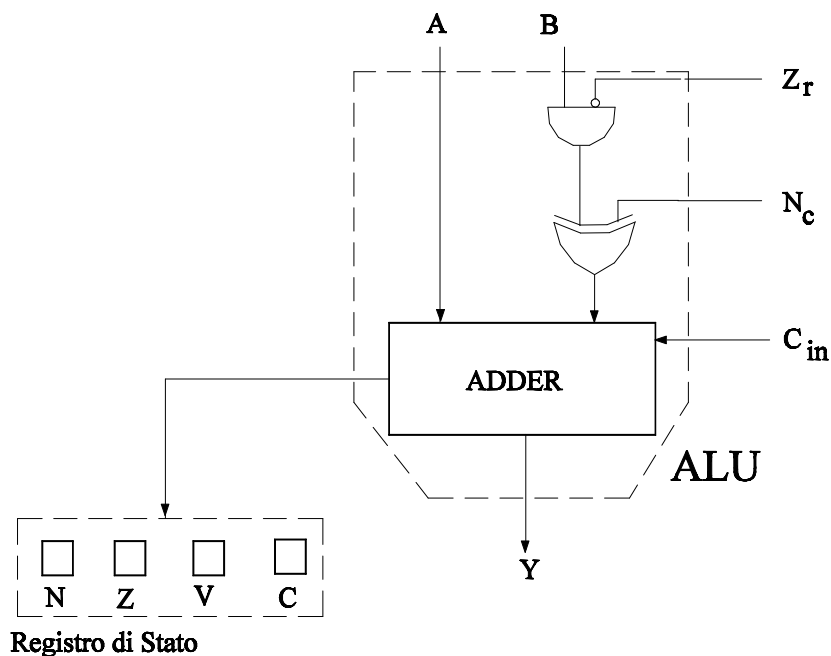


Fig.2.0.5

La logica circuitale di attivazione dei segnalatori è stata inglobata all'interno dell'integrato contenente l'Adder e lo schema degli ingressi e uscita multiterminati è stato sostituito da uno schema di tipo unificare. In questo schema si sono simbolicamente rappresentate le quattro porte EOR con una sola porta. All'ingresso B dell'Adder sono state aggiunte quattro porte AND, con un ingresso in comune pilotato dal bit di controllo Z_r , per consentire l'azzeramento del dato B (con $Z_r=1$). Questo ulteriore circuito consente di programmare anche le operazioni $A+1$ ($Z_r=N_c=0$, $C_{in}=1$) e $A-1$ ($Z_r=0$, $N_c=1$, $C_{in}=0$), necessarie in certi tipi di elaborazioni.

Dato che circuiti di questo tipo possono produrre anche operazioni di tipo logico, con AND, OR e EOR bit a bit, mediante l'aggiunta di ulteriori bit di controllo (non mostrati in figura), il nome dato complessivamente a questo dispositivo è quello di **ALU** (Aritmetico Logico Unità: Unità Logico Aritmetica).

I quattro segnalatori aritmetici formano complessivamente il **registro di stato**.

2.0.5 Macchina che consente elaborazioni di un insieme di dati (Data Path)

Per effettuare operazioni aritmetiche su un insieme di dati, in pratica per risolvere una espressione algebrica su un certo insieme di dati, del tipo: $D3 = 2 \cdot D0 + D1 - D2$, con $D0$, $D1$, $D2$, $D3$ simboli letterali che rappresentano quattro valori numerici, è necessario risolvere i seguenti problemi pratici:

- 1) predisporre all'interno della macchina digitale tanti dispositivi atti a contenere tutti i dati coinvolti nella elaborazione. Nel caso esemplificato sono necessari almeno quattro di questi "contenitori di dati";
- 2) utilizzare una unità in grado di fare elaborazioni algebriche **su coppie di dati**, del tipo di quella descritta nel paragrafo precedente;
- 3) **scomporre** l'operazione descritta in una successione di operazioni ognuna coinvolgente uno o due soli dati, in modo che, dopo un certo numero di passi intermedi, il risultato finale sia quello corrispondente all'intera espressione;
- 4) prevedere, se necessario altri contenitori di dati utili a mantenere temporaneamente i risultati intermedi dei singoli passi previsti al punto 3.

punto 1)

I dispositivi atti a contenere dati sono i **registri** (o **celle di memoria**) dispositivi formati da tanti di **flip flop**, (o **celle elementari di memoria**) quanti sono i bit che compongono i dati da elaborare. In base alle considerazioni fatte precedentemente, questi registri sono formati dallo stesso numero di celle elementari.

La **Fig.2.0.5.1** mostra la struttura di questi componenti e le modalità con cui memorizzare i dati. In particolare in **Fig.2.0.5.1 a)** si osserva l'*operazione di mantenimento* che, con i **bit di controllo** **Read=0** e **Write=0**, impedisce sia la memorizzazione del dato posto agli ingressi **D** (0101), sia la lettura del dato memorizzato nei quattro flip flop e leggibile sulle uscite **Y** (1000). La **Fig.2.0.5.1 b)** mostra l'operazione di scrittura che, con **Write=1**, memorizza l'ingresso **D** (0101) nei flip flop (**Q=D**), senza peraltro rendere possibile la lettura essendo **Read=0**. Infine in **Fig.2.0.5.1 c)** con **Read=1** e **Write=0**, è mostrata l'operazione di lettura (**Y=Q**). La **Fig.2.0.5.1 d)** mostra il simbolo sintetico che sarà utilizzato nel seguito.

punto 2

L'ALU descritta nei paragrafi precedente è in grado di effettuare una sola delle due operazioni di somma e di differenza sui due dati posti ai suoi ingressi **A** e **B**. Per questo motivo l'operazione proposta all'inizio del paragrafo dovrà essere scomposta in una **successione di operazioni**, ognuna della quali dovrà coinvolgere una sola coppia di dati.

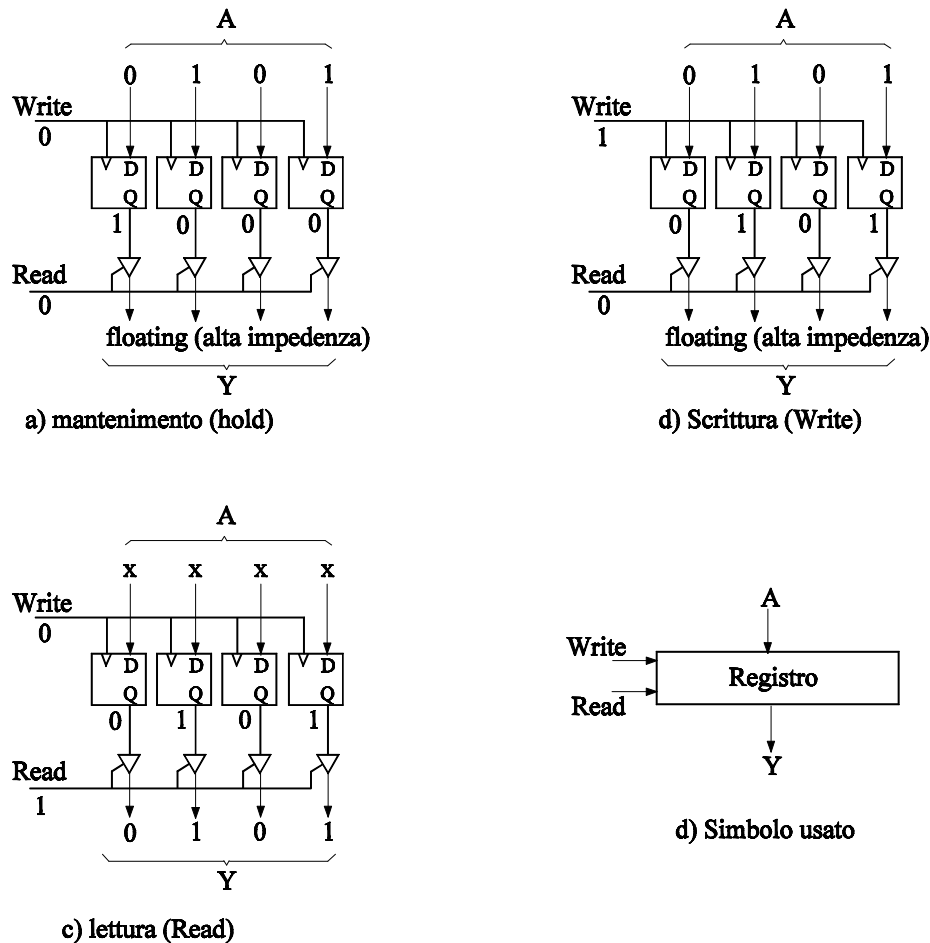


Fig.2.0.5.1 Operazioni du un registro. Le uscite sono pilotate da porte tree state il cui ingresso di controllo è **Read**. **Read=0** produce un uscita in alta impedenza per cui i livelli logici di Y dipendono dai circuiti a cui queste linee si collegano (non mostrate in figura). Nel caso c) gli ingressi “x” indicano qualunque valore (don’t care).

punto 3

l’operazione iniziale può essere riscritta:

$$D3 = 2 \cdot D0 + D1 - D2 = (((D0 + D0) + D1) - D2)$$

in modo da risolvere prima la parentesi più interna ($D0 + D0$) e porre il risultato parziale in un registro disponibile, per esempio $D3$ (lo stesso che alla fine dovrà contenere il risultato finale). Per fare questo bisogna inviare ai due ingressi dell’ALU, programmata per fare la somma, il contenuto di $D0$ e memorizzare il risultato in $D3$. Successivamente agli ingressi all’ALU, programmata ancora per fare la somma, possono essere inviati i contenuti di $D3$ e di $D1$ e memorizzare il secondo risultato parziale (corrispondente al secondo livello di parentesi) in $D3$. Infine, inviando agli ingressi dell’ALU, programmata per fare la differenza, il valore di $D3$ ed il valore di $D2$ si ottiene il risultato finale da memorizzare definitivamente in $D3$.

La successione delle tre operazioni di due dati, prodotte dall'ALU, è quindi:

$$\begin{aligned}
 D3 &= 2 \cdot D0 + D1 - D2 & \text{realizzata: } D3 &= D0 + D0 \\
 D3 &= D3 + D1 \\
 D3 &= D3 - D2
 \end{aligned}
 \tag{2.0.2}$$

Il registro **D3** che nella seconda e terza operazione aggiunge o sottrae il valore contenuto in un altro registro svolge la funzione di **registro accumulatore**.

Per attuare questo meccanismo di elaborazione *è necessario creare dei collegamenti opportuni tra l'ALU ed i registri, in modo che in ogni fase operativa i dati di due soli registri, tra tutti quelli disponibili, siano trasmessi ai due ingressi dell'ALU, la cui uscita dovrà essere collegata, successivamente, ad uno qualsiasi degli stessi registri*. Dovendo cambiare, in fasi successive, le sorgenti e la destinazione dei dati elaborati dall'ALU, è necessario inserire nella struttura dei circuiti di selezione (*struttura di interconnessione*), deputati alla predisposizione dei giusti collegamenti per ogni operazione da produrre.

La soluzione tecnologica a questa richiesta può essere quella mostrata in **Fig.2.0.5.2**.

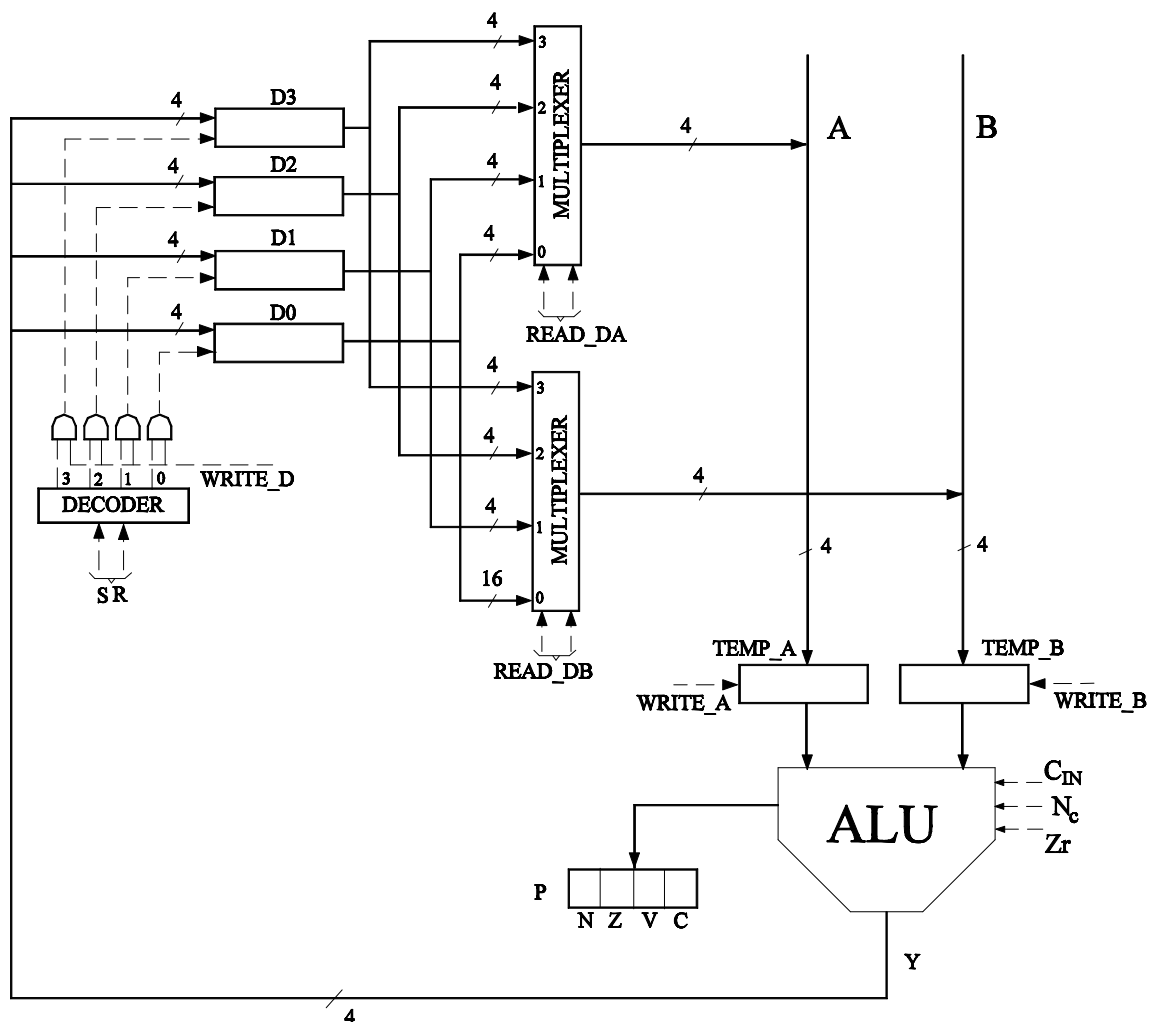


Fig.2.0.5.2 Data Path

In questa struttura, oltre ai quattro registri necessari per contenere i dati sono stati aggiunti altri due registri temporanei aventi lo scopo di mantenere i due dati A e B in ingresso all'ALU.

Oltre ai registri e all'ALU, il sistema di **Fig.2.0.5.2**, ha una rete di interconnessione formata da tre bus: **A**, **B**, **Y**, e da due multiplexer che selezionano i due registri che debbono inviare il loro contenuto ai registri di ingresso dell'ALU.

Per effettuare la prima delle tre operazioni richieste, $D3 = D0 + D0$, è necessario, *in una prima fase*, collegare l'uscita di **D0** sia con gli ingressi di **Temp_A** che **Temp_B**, abilitandone la scrittura. Ciò si ottiene ponendo i due bit di controllo dei due multiplexer **Read_A** e **Read_B** entrambi a zero e ponendo a uno i due bit di controllo **Write_A** e **Write_B**. L'ALU deve essere programmata per fare la somma per cui i suoi bit di controllo devono essere $C_{in}=N_c=Z_r=0$. Dopo pochi nanosecondi di tempo la somma $D0 + D0 = 2 \cdot D0$, è disponibile su tutti gli ingressi dei quattro registri, dato che il bus **Y** si dirama su ognuno di questi componenti. Questo risultato non si memorizza in nessuno dei registri fintanto che **Write_D** = 0. Volendo, come richiesto, memorizzare il risultato della somma in **D3**, *in una seconda fase*, è necessario porre i due bit di controllo **SR** pari a 11 (abilita l'uscita 3 della decodifica) ed il bit di controllo **Write_D** = 1.

L'operazione descritta è divisa in *due fasi temporali*: la prima seleziona due dati memorizzati in due registri e li invia ai due ingressi dell'ALU, memorizzandoli in due registri temporanei; la seconda fase memorizza il risultato su due registri di destinazione. La durata di ognuna delle suddette fasi dovrà essere adeguata alla velocità dei circuiti utilizzati, dipendente dalla tecnologia costruttiva dei singoli componenti (tecnologia MOS o tecnologia CMOS).

Con lo stesso meccanismo si possono realizzare, ordinatamente, le altre due operazioni indicate in (2.0.2).

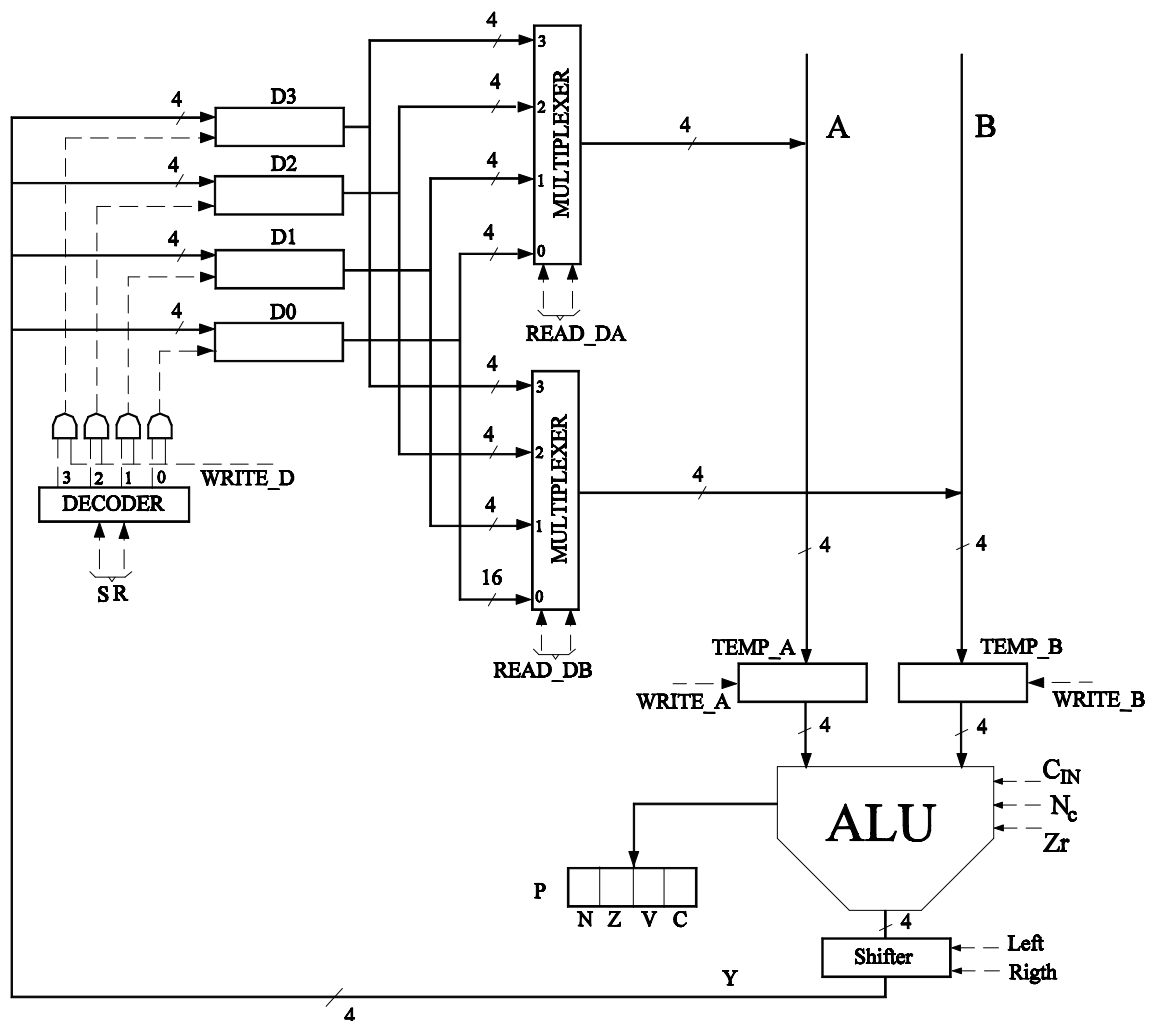


Fig. 2.0.5.3 Data path con shifter

E' utile aggiungere in uscita dell'ALU un circuito shifter, in grado di consentire lo scorrimento a sinistra (moltiplicazione per due) o a destra (divisione per due) del risultato. La Fig.2.0.5.3 mostra questa modifica.

Lo shifter può essere formato sia da quattro latch sia con semplici porte logiche. In questo caso si parla di **barrel shifter** uno schema del quale è mostrato in Fig.2.0.5.4.

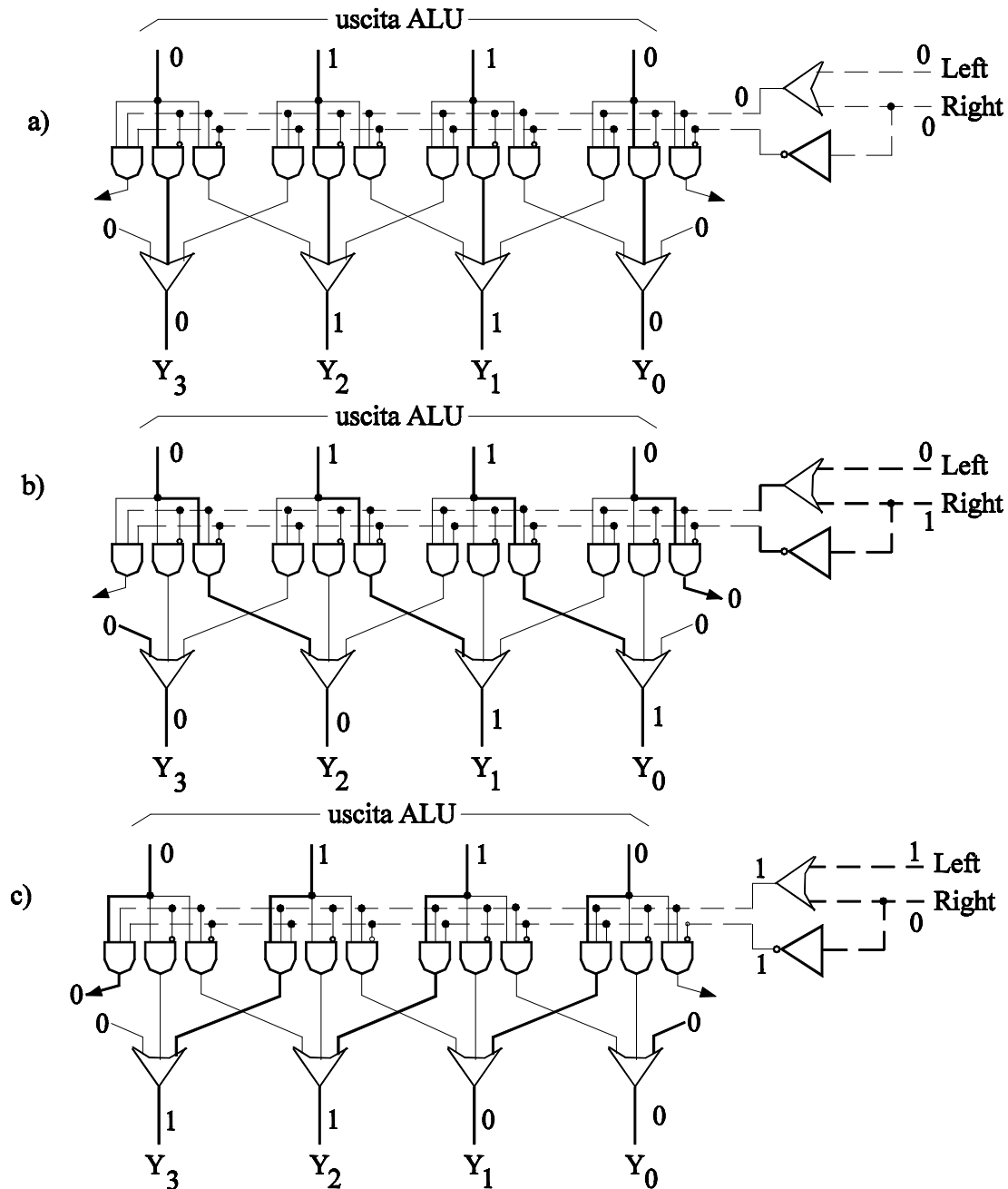


Fig.2.0.5.4 Barrel Shifter. a) Trasferimento in uscita dell'ingresso. b) Scorrimento a destra di un bit. c) Scorrimento a sinistra di un bit.

L'inserimento dello shifter, come in Fig.2.0.5.3, modifica la sequenza (2.0.2) nella:

$$\begin{aligned}
 D3 = 2 \cdot D0 + D1 - D2 \quad \text{realizzata:} \quad & D3 = 2 \cdot D0 \quad (\text{shift a sinistra}) \\
 & D3 = D3 + D1 \\
 & D3 = D3 - D2
 \end{aligned}
 \tag{2.0.2}$$

2.0.6 Unità di controllo – Processore

L'esecuzione della sequenza mostrata in (2.0.2), con la macchina descritta in Fig.2.0.5.3, richiede l'applicazione di una adeguata sequenza dei bit di controllo, ad intervalli di tempo regolari, tale da produrre la successione delle singole fasi della operazioni descritte nel paragrafo precedente. In particolare bisogna:

$D3 = 2 \cdot D0$

- trasferire il contenuto di **D0** nel registro **Temp_A** attraverso il bus A, ponendo i bit di controllo **Read_DA=00**, **Write_A=1**, **Cin=Nc=0**, **Zr=1**, **Left=1** (i bit di controllo non menzionati adesso e nel seguito si sottintendono a zero). In questo modo in uscita all'ALU vi è il valore **D0+0=D0**, ed in uscita allo shifter il valore **2·D0**;
- trasferire l'uscita dello shifter, tramite il bus Y, in **D3**, ponendo **SR=11**, **Write_D=1**. In questo modo si memorizza il risultato dell'elaborazione in **D3**.

$D3 = D3 + D1$

- trasferire il contenuto di **D3** in **Temp_A** ed il contenuto di **D1** in **Temp_B** ponendo **Read_DA=11**, **Read_DB=01**, **Write_A= Write_B=1**, **Cin= Zr=Nc=0**. In questo modo all'uscita dell'ALU e dello shifter vi è **D3 + D1**;
- trasferire l'uscita dello shifter in **D3**.

$D3 = D3 - D2$

- trasferire il contenuto di **D3** in **Temp_A** ed il contenuto di **D2** in **Temp_B** ponendo **Read_DA=11**, **Read_DB=10**, **Write_A= Write_B=1**, **Cin=Nc=1**, **Zr=0**. In questo modo all'uscita dell'ALU e dello shifter vi è **D3 - D2**;
- trasferire l'uscita dello shifter in **D3**.

Ognuna delle tre operazioni sopra indicate è stata suddivisa in *due fasi*: la prima è di *lettura* e trasferisce gli operandi, contenuti nei registri, all'ALU, la seconda è di *scrittura* e trasferisce il risultato nel registro di destinazione (**D3** in questo caso). Come già detto, queste fasi debbono durare un tempo sufficiente a consentire ai dati di ingresso dei singoli dispositivi (ALU, shifter registri) di propagarsi fino all'uscita, per produrre valori corretti. Questi tempi sono piccoli (alcuni nanosecondi) ma non nulli.

Per realizzare tutto questo, si può memorizzare l'insieme dei bit di controllo, che esegue ognuna delle fasi descritte (i bit indicati con “-” possono valere indifferentemente 0 o 1, in quanto non influenzano l'operazione):

	Read_DA	Write_A	Read_DB	Write_B	Zr	Nc	Cin	Rigth	Left	Sel_Write	Write_D	
<u>fase 1</u>	00	1	--	-	1	0	0	0	1	--	0	2D0
<u>fase 2</u>	--	0	--	-	1	0	0	0	1	11	1	D3=2D0
<u>fase 1</u>	11	1	01	1	0	0	0	0	0	--	0	D3+D1
<u>fase 2</u>	--	0	--	0	0	0	0	0	0	11	1	D3=D3+D1
<u>fase 1</u>	11	1	10	1	0	1	1	0	0	--	0	D3-D2
<u>fase 2</u>	--	0	--	0	0	0	0	0	0	11	1	D3=D3-D2

all'interno di registri, organizzati in una struttura del tipo di quella descritta in Fig.2.0.6.1.

Ponendo all'ingresso A.B. il codice 00, viene abilitato il solo tree state collegato all'uscita 0 del decoder (**Read=1**) ed il corrispondente registro è l'unico dei quattro ad inviare in uscita (D.B.) il suo contenuto. Analogamente per le altre tre combinazioni applicabili all'ingresso A.B.

Ogni registro di questa struttura è detto **locazione di memoria** ed il codice posto in A.B., che lo seleziona, è detto **indirizzo**. Il dispositivo, nel suo complesso è detto **memoria a sola lettura** (ROM). Le singole locazioni di memoria, quattro nel caso esemplificato, sono contraddistinte dall'indirizzo che le seleziona. La *locazione 0* della ROM è quella selezionata da A.B.=00, la *locazione 1* è quella selezionata da A.B.=01, ecc.

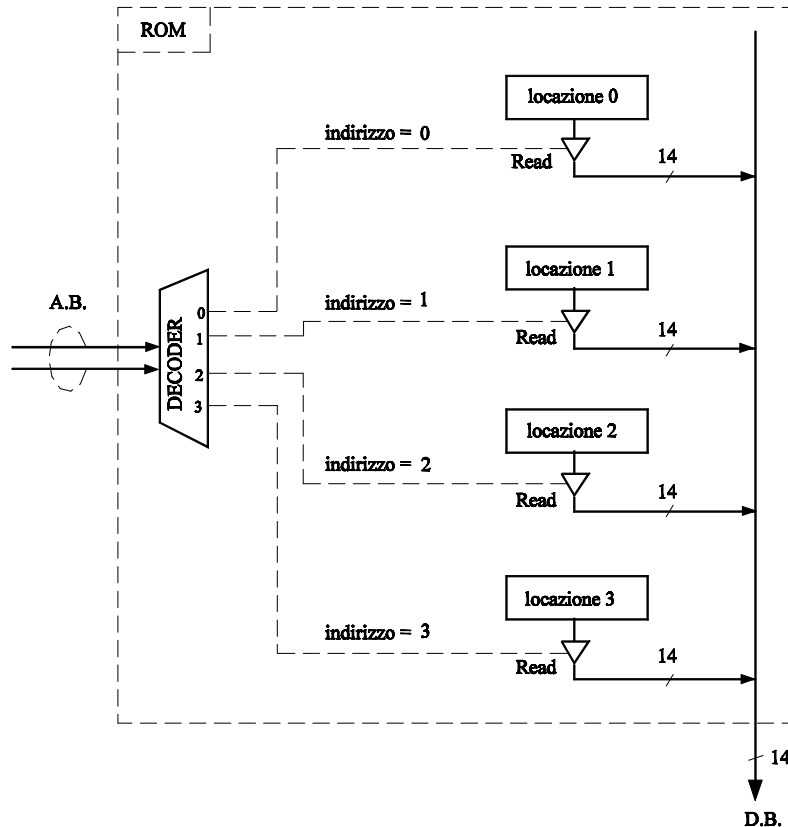


Fig.2.0.6.1 Schema di principio di una memoria a sola lettura (ROM).

Per dare una dimensione alla ROM da usare nel data path di **Fig.2.0.5.3** è necessario contare il numero dei bit di controllo necessari per pilotarlo. Si hanno complessivamente 14 bit. Ogni operazione descritta sul data path si sviluppa in due fasi, ognuna con una configurazione di bit particolare, quindi sono necessari due locazioni di ROM per ogni operazione eseguita. In **(2.0.2)** sono mostrate le tre espressioni che risolvono la relazione proposta. Serve quindi una ROM con $2 \times 3 = 6$ locazioni, ognuna contenente 14 bit.

Per eseguire le tre operazioni indicate in **(2.0.2)** bisogna memorizzare ordinatamente nelle sei locazioni 0, 1, 2, 3, 4, 5, della ROM, le sei configurazioni di bit di controllo valutate precedentemente. Bisogna inoltre pilotare l'ingresso A.B. della ROM con un contatore up, inizializzato a 0, che, alla cadenza di un clock adeguato, generi in successione gli indirizzi delle sei locazioni. In questo modo, all'uscita D.B. della ROM, appariranno, in sequenza temporale, le sei configurazioni di 14 bit di controllo capaci di produrre singolarmente le operazioni richieste.

Aggiungendo questi dispositivi al data path di **Fig.2.0.5.3** si ottiene il sistema di **Fig.2.0.6.2**.

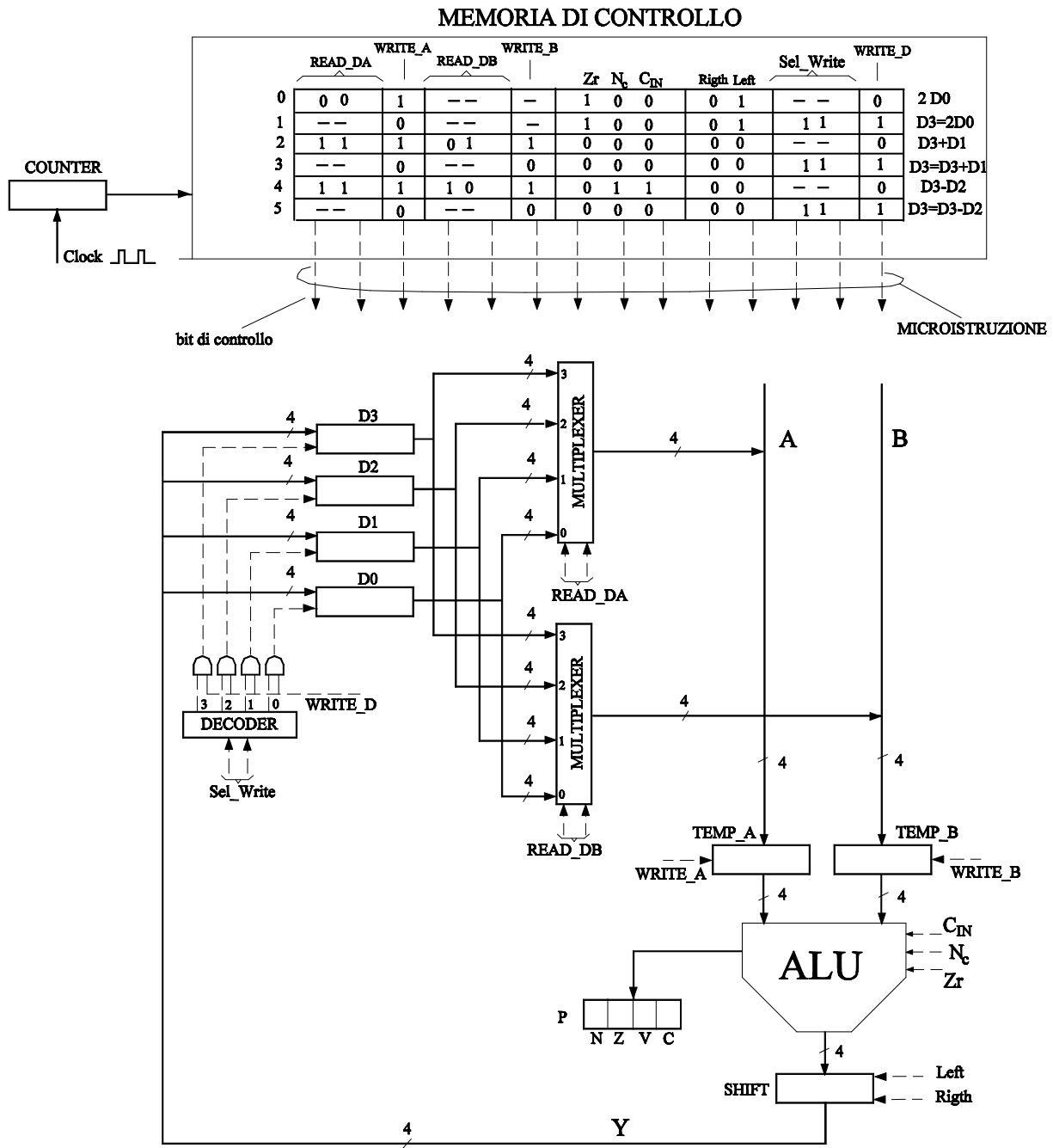


Fig.2.0.6.2 Schema di principio di un semplice processore.

La memoria di controllo, anche se rappresentata in modo più sintetico, ha lo schema di principio di **Fig.2.0.6.2**. Il contenuto di ogni registro è stato rappresentato in modo dettagliato ripartendo in base alla funzione svolta, in *campi* diversi, i singoli bit di controllo.

Il sistema di **Fig.2.0.6.2** si compone di due parti:

- il **data path**, avente la funzione descritta precedentemente;
- l'**unità di controllo**, formata dalla **ROM di controllo**, contenente la sequenza di bit di controllo, e dal **generatore di sequenza** di indirizzi, costituito da un semplice contatore. L'insieme dei bit fornito in uscita della ROM è detta **microistruzione**.

Questo sistema è chiamato **processore**.

2.0.7 Processore per impieghi generali

Il processore descritto in precedenza ha solo scopo esemplificativo in quanto esegue sempre la stessa operazione, peraltro alquanto banale, con un investimento in hardware del tutto ingiustificato.

Si può cercare di costruire una macchina digitale che, basata sull'idea di scomposizione descritta con la (2.0.2), sappia risolvere relazioni più complesse e modificabili sulla base di un programma che determina la sequenza con cui eseguire la sequenza di operazioni elementari. In pratica è necessario:

- costruire dei circuiti (*hardware*) che sappiano eseguire un insieme di *operazioni elementari*, dette **operazioni primitive**;
- aggiungere una struttura di controllo che sappia comandare l'esecuzione di una sequenza qualsiasi delle suddette operazioni primitive, sulla base di un prefissato **programma**.

La soluzione a queste richieste può essere rappresentata dal sistema di **Fig.2.0.7.1**.

Si nota una sostanziale modifica del data path, anche se la struttura generale rimane la stessa. La funzione svolta dai precedenti quattro registri **D0, D1, D2, D3**, consistente nel contenere i dati e i risultati dalle elaborazioni, è adesso delegata al solo registro **D0**. I nuovi registri, **Operando** e **PC**, hanno nomi legati alla particolare funzione cui sono destinati.

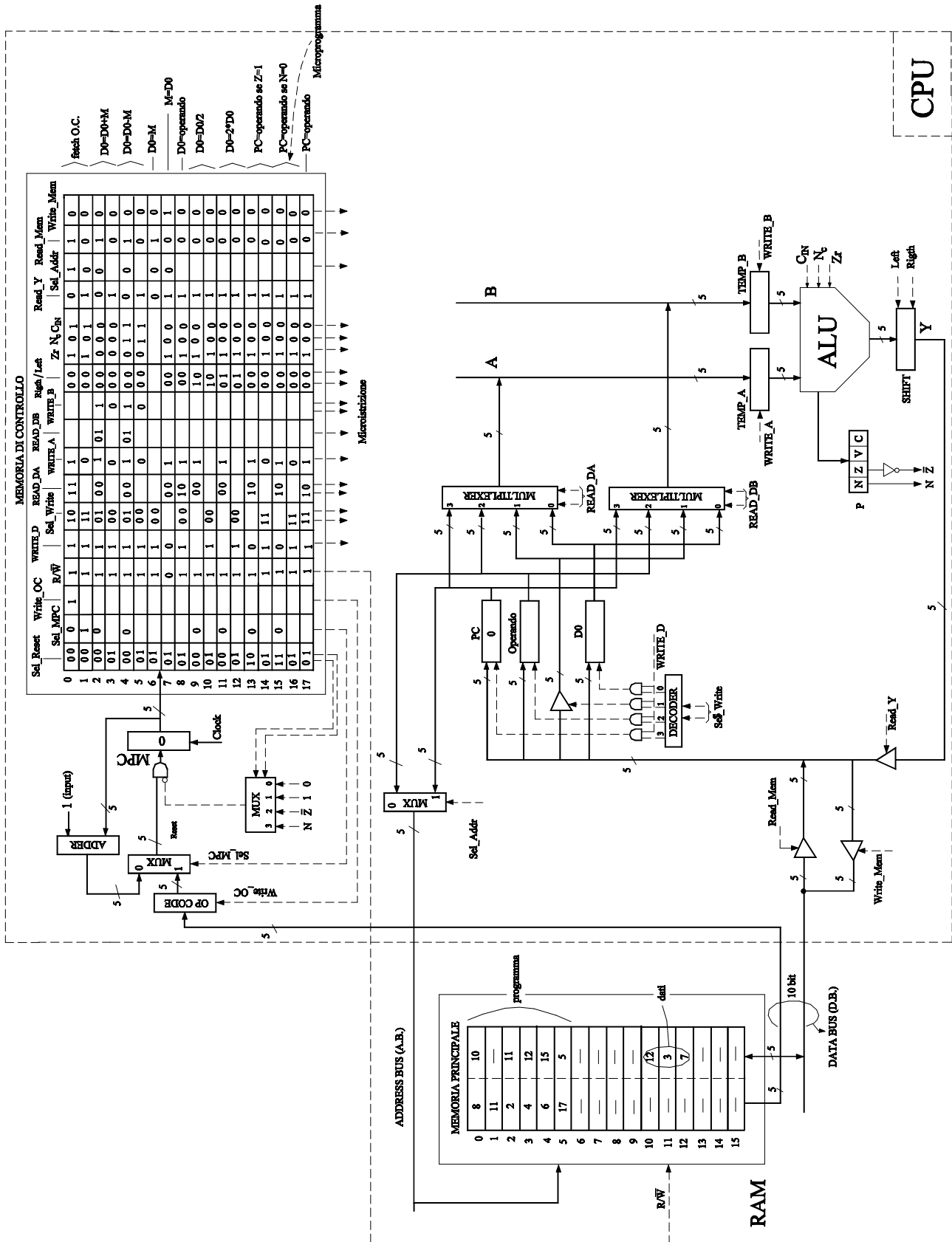
La riduzione dei registri contenenti dati è ampiamente compensata dall'utilizzazione di una **memoria a lettura e scrittura (RAM)**, esterna al processore (**Memoria principale**) e collegata a questo da due canali di comunicazione, *A.B.* e *D.B.*, e dal bit di controllo $R/\bar{W}$. La struttura interna della **RAM** è simile a quella della **ROM**, con la differenza che, ogni locazione, come uno dei quattro registri **D0, D1, D2, D3** di **Fig.2.0.7.1**, oltre ad essere letta, può essere scritta con un nuovo valore. La **Fig.2.0.7.2** mostra questo nuovo importante componente. Le locazioni di questa **RAM** contengono dati da 10 bit: i cinque più significativi sono collegati al registro **OP_CODE** ed i cinque meno significativi con il bus **Y** del data path.

Nel sistema esemplificato le locazioni dall'indirizzo 10 all'indirizzo 15 sono usate per svolgere la funzione di registri contenenti dati (in sostituzione di **D1, D2** e **D3** dello schema precedente).

L'indirizzo inviato alla **RAM** può essere scelto tra due fonti di provenienza:

- il registro **PC (Program Counter)**
- il registro **Operando**.

Anche l'unità di controllo è ampiamente modificata, rispetto a quella di **Fig.2.0.7.1**. Il contatore di indirizzamento della **ROM** è stato sostituito da una logica circuitale che consente una più ampia selezione dell'indirizzo da inviare alla memoria di controllo, come sarà spiegato nel seguito. La stessa **ROM** di controllo di sole 6 locazioni, in **Fig.2.0.7.1**, è stata sostituita da una memoria di 18 locazioni, necessaria a contenere tutte le microistruzioni di questa nuova macchina.



Per comprendere il funzionamento dell'intero sistema si fa riferimento alla **Fig.2.0.7.1**, intendendo che i contenuti della memoria, dei registri interni alla **CPU** e della memoria principale, rappresentano lo stato del sistema subito dopo l'accensione.

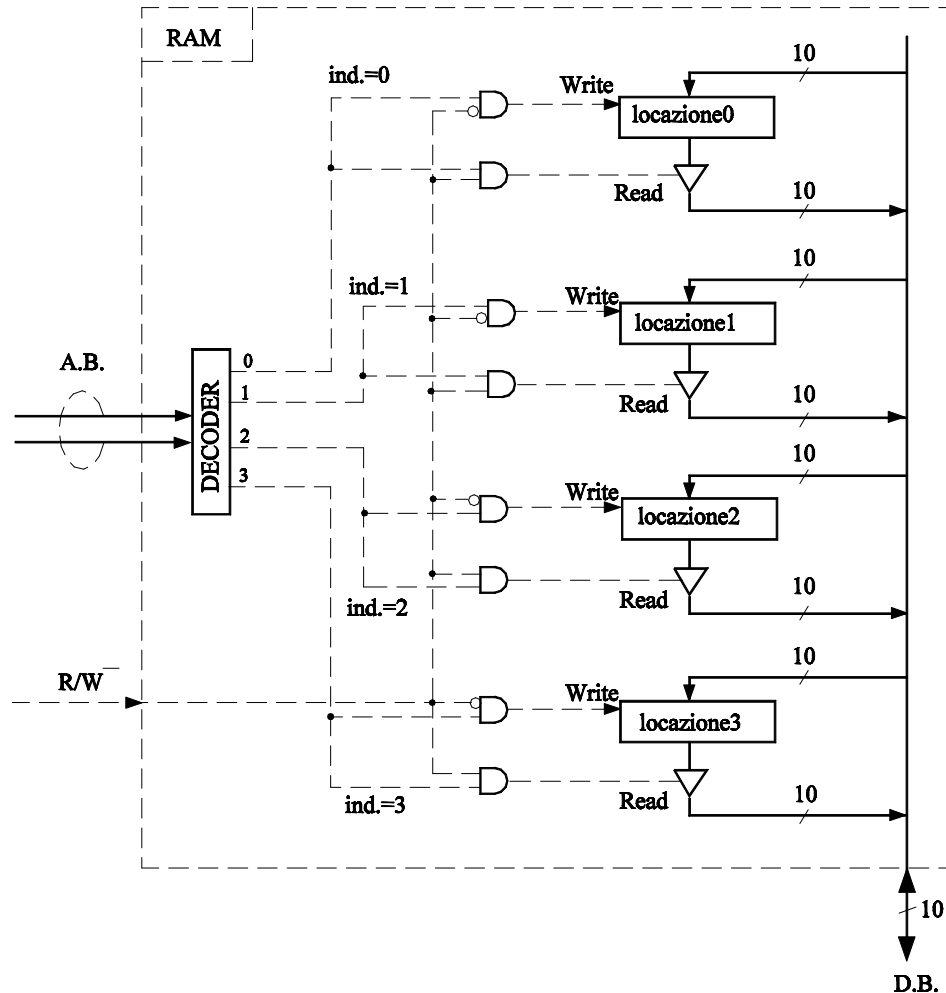


Fig.2.0.7.2 Schema di principio di una memoria a lettura e scrittura (**RAM**) con 4 locazioni.

Lettura $R/\bar{W} = 0$. Scrittura $R/\bar{W} = 1$.

Ricordando che le attività che si svolgono sul data path dipendono dai bit di controllo e quindi dalla microistruzione fornita dall'unità di controllo (unità di controllo: **ROM** di controllo più circuito generatore dei suoi indirizzi), l'evoluzione del sistema è:

accensione: il registro **MPC (microprogramcounter)** indirizza la locazione 0 della **ROM**. Cinque bit della microistruzione agiscono sul generatore di indirizzi mentre i rimanenti 18 agiscono sul data path. Questi 18 bit, come si può facilmente controllare, producono l'invio del contenuto del registro **PC (Program Counter)** sul **Address Bus** (con **Sel_Addr=1**), consentendo di indirizzare la locazione 0 della RAM e di inviare lo stesso contenuto del **PC** all'**ALU** in modo che sul bus **Y** vi sia il valore **PC+1** (**Read_DA=11**, **Write_A=1**, **ZrNcCin=101**, **Rigth/Left=00**). Infine con la stessa

microistruzione viene letta ($R/W=1$) la locazione 0 della **RAM** inviando, tramite il **Data Bus**, i 5 bit più significativi (nel caso il valore 8) al registro **OP_CODE** ($Write_OC=1$) e i 5 bit meno significativi (nel caso il valore 10) al registro **Operando** ($Read_Mem=1$, $Sel_Write=10$, $Write_D=1$).

producendo le seguenti alternative sul prossimo indirizzo generato dal **MPC** all'arrivo dell'impulso di **Clock**:

- il prossimo indirizzo è quello attuale più uno se i bit $Sel_Reset=00$, $Sel_MPC=0$. Si ha il funzionamento come contatore up analogo a quello di **Fig.2.0.6.2**;
- il prossimo indirizzo è zero se $Sel_Reset=01$. Il reset del **MPC** lo riporta allo stato iniziale
- il prossimo indirizzo è quello contenuto nel registro **OP_CODE** se $Sel_Reset=00$ e $Sel_MPC=1$.

La microistruzione di indirizzo zero, con $Sel_Reset=00$, $Sel_MPC=0$, impone la prima delle tre scelte.