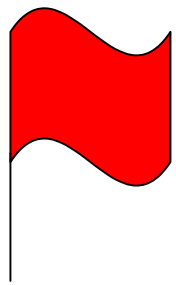
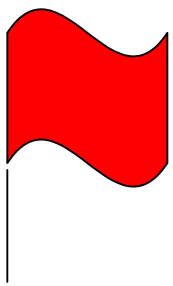
Rappresentazione dei numeri

- Intervallo numerico
- Base di numerazione e numero di cifre
- Approssimazione
- Condizione di overflow
- Condizione di underflow

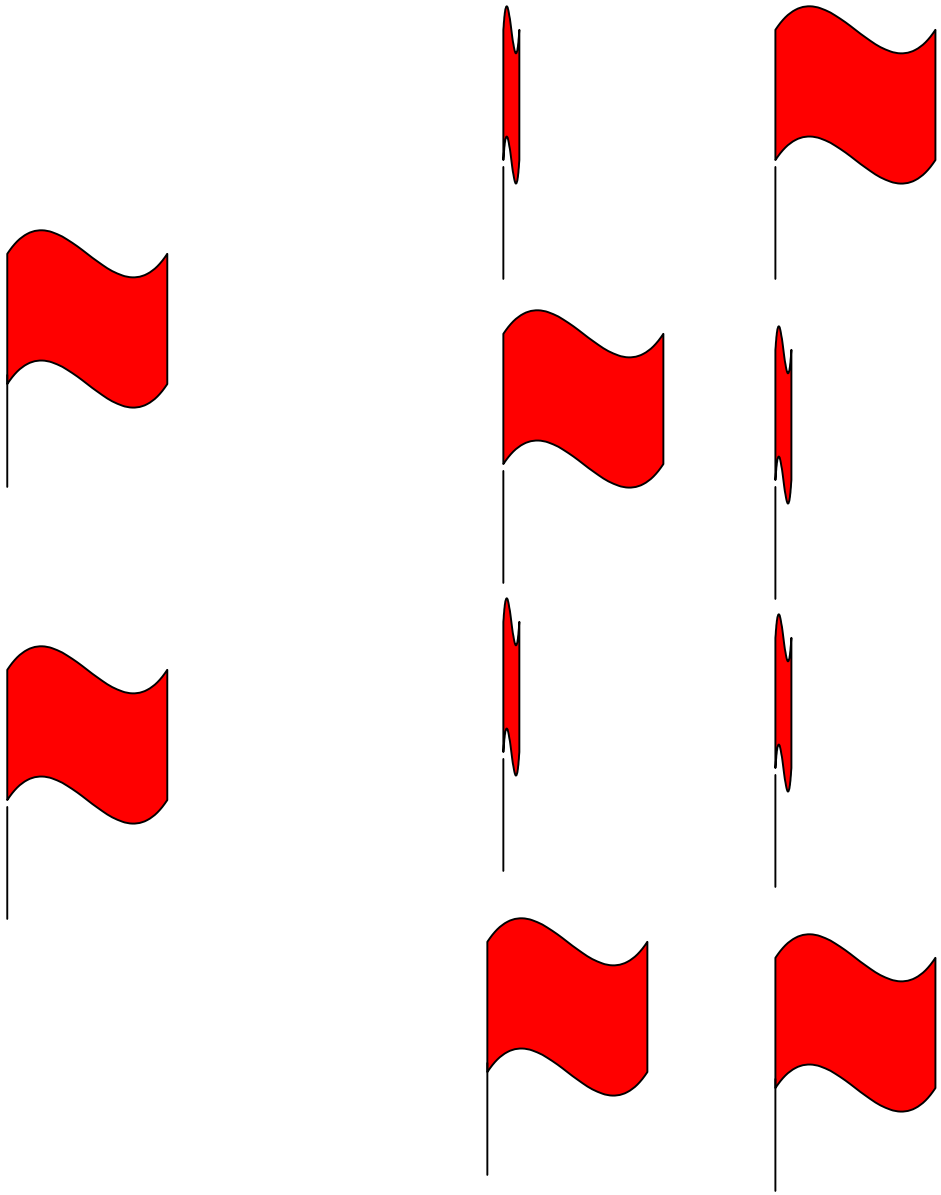
La Rappresentazione dell'informazione

Bandiera= strumento di rappresentazione





1 bandiera → 2 messaggi diversi



Messaggio n°1

Messaggio n°2

Messaggio n°3

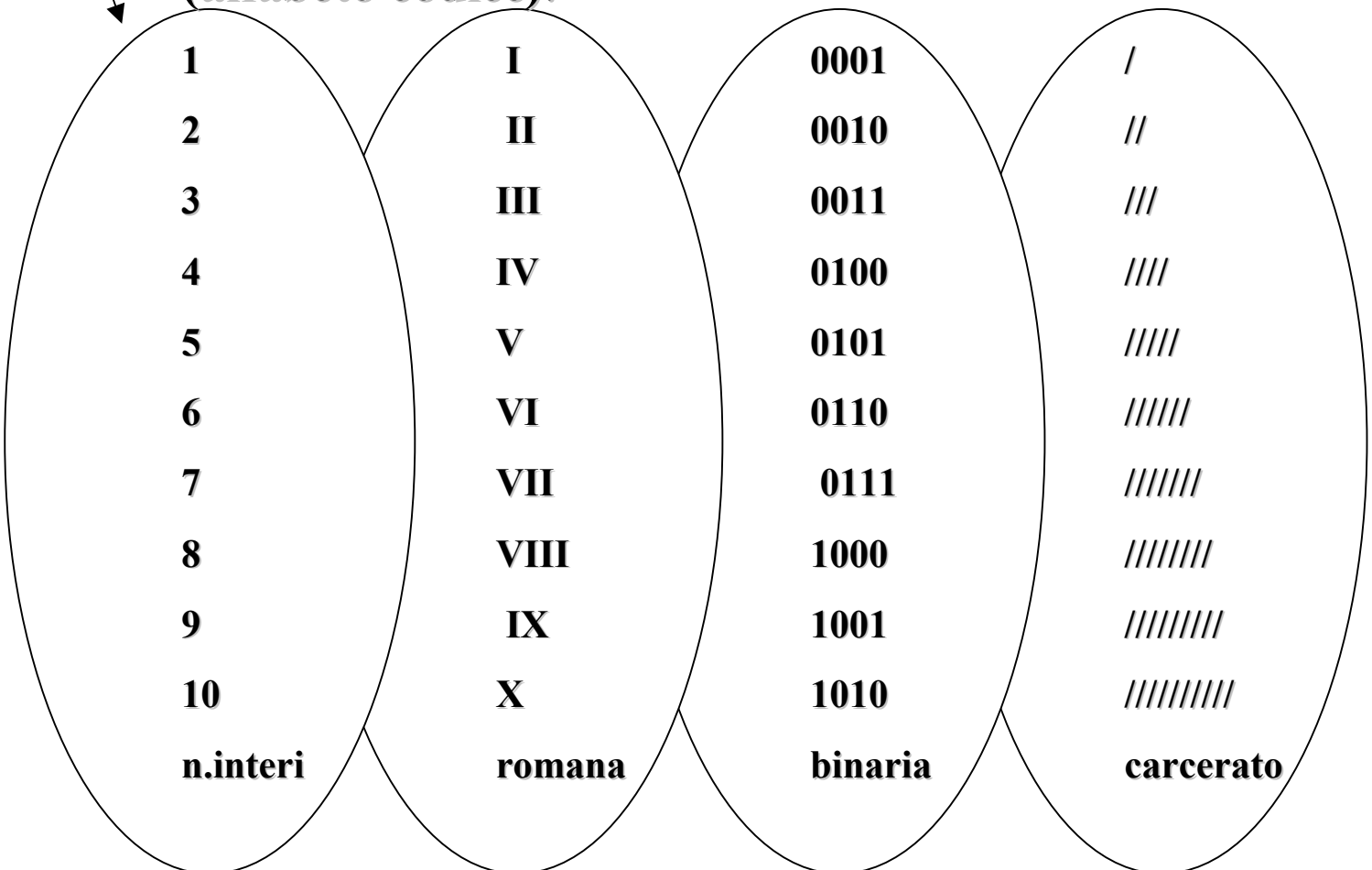
Messaggio n°4

2 bandiere → 4 messaggi diversi

Problema della codifica

alfabeto sorgente

Il problema è di associare a ciascun elemento di un insieme di informazioni **(alfabeto sorgente**)una sequenza di simboli **(alfabeto codice)**:



L’alfabeto codice

Insieme di simboli introdotto per rappresentare le informazioni

a b d e f g h……..

0 1 2 3 4 5 6 7 8 9

- .

0 1

I V X C M D

In genere la cardinalità dell’alfabeto codice è inferiore al numero dei valori diversi dell’informazione da rappresentare (cardinalità dell’alfabeto sorgente)

Codifica come applicazione

Applicazione biunivoca che associa ad un valore dell'informazione (alfabeto sorgente) una ed una sola sequenza di simboli dell'alfabeto codice *(parola codice)*

Se:

- **a** è la cardinalità dell'alfabeto
- **r** è la lunghezza della parola

allora $\mathbf{n = a^r}$

è il numero di parole diverse.

Esempio di codifica

Alfabeto: **/ . –**

Lunghezza parola: 2

nove parole diverse:

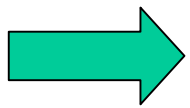
/. ./ /- -/ .- -. .. -- //

tutte le possibili disposizioni con ripetizione

Caratteristiche della codifica

A lunghezza fissa:

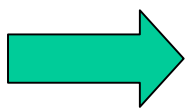
le parole hanno la medesima dimensione



$A^r > n$

A lunghezza variabile:

negli altri casi

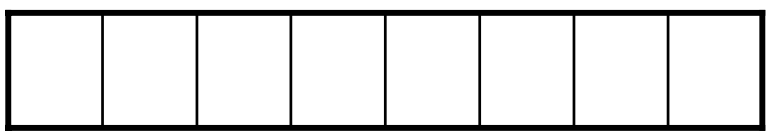


scelta opportuna della lunghezza della parola codice in base all'occorrenza

I registri di memoria

E’ un contenitore costituito da una sequenza di celle.

Ciascuna cella (**il bit)** è in grado di memorizzare solo due simboli diversi (0 o 1)



2^8 combinazioni

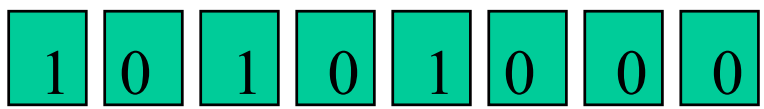
Registro: Ad es. casella di 8 bit (8 flip-flop)

MEMORIE

La capacità delle memorie (*di massa e centrali*) viene espressa in multipli del byte

1 KB	(**Kilobyte**)	$=2^{10}$	=1024 byte	10^3 byte
1 MB	(**Megabyte**)	$=2^{20}$	=1.049.776	10^6 byte
1 GB	(**Gigabyte**)	$=2^{30}$		10^9 byte
1 TB	(**Terabyte**)	$=2^{40}$		10^{12} byte
1 PB	(**Petabyte**)	$=2^{50}$		10^{15} byte

byte una stringa di 8 bit



Sistemi di numerazione

Nel **sistema romano** ciascuna cifra esprime una quantità indipendente dalla propria posizione:

MML (2050)

MDCCCLXXXVIII (1888)

Nel **sistema decimale** ogni cifra ha un'importanza *(peso)* variabile a seconda della sua posizione nel numero (codice): ad esempio, la prima cifra a destra indica le unità, la seconda indica le decine, la terza le centinaia, ecc.

$84079 = 8*10^4 + 4*10^3 + 0*10^2 + 7*10^1 + 9*10^0 =$

80000 + 4000 + 0 + 70 + 9

Sistemi di numerazione posizionali

Fissati k simboli si può definire un sistema posizionale in base k

La base di rappresentazione viene specificata in pedice:

$(4791)_{10}$ $(1431)_5$ $(10101)_2$

Conversione da una base k qualsiasi a quella decimale:

è sufficiente considerare la successione di cifre con il peso nella base k assegnata ed eseguire la somma

Esempio numerazione binaria:

$(10101)_2=1*2^4+0*2^3+1*2^2+0*2^1+1*2^0=21$

Conversioni di base $2^k \longrightarrow$ 2

Conversione diretta:

• Ogni cifra deve essere sostituita con la corrispondente parola codice di k bit

Conversione inversa:

• Si raggruppano i bit in gruppi di k a partire dai bit meno significativi (cioè da sinistra a destra)

• Ogni gruppo deve essere sostituito con la corrispondente cifra.

Esempio conversione esadecimale-binario:

A78F $\longrightarrow$ 1010 0111 1000 1111

Sistemi di numerazione

Nel **sistema romano** ciascuna cifra esprime una quantità indipendente dalla propria posizione:

MML (2050)

MDCCCLXXXVIII (1888)

Nel **sistema decimale** ogni cifra ha un'importanza *(peso)* variabile a seconda della sua posizione nel numero (codice): ad esempio, la prima cifra a destra indica le unità, la seconda indica le decine, la terza le centinaia, ecc.

$84079 = 8*10^4 + 4*10^3 + 0*10^2 + 7*10^1 + 9*10^0 =$

80000 + 4000 + 0 + 70 + 9

Sistemi di numerazione posizionali

Fissati k simboli si può definire un sistema posizionale in base k

La base di rappresentazione viene specificata in pedice:

$(4791)_{10}$ $(1431)_5$ $(10101)_2$

Conversione da una base k qualsiasi a quella decimale:

è sufficiente considerare la successione di cifre con il peso nella base k assegnata ed eseguire la somma

Esempio numerazione binaria:

$(10101)_2=1*2^4+0*2^3+1*2^2+0*2^1+1*2^0=21$

Conversioni di base $2^k \longrightarrow$ 2

Conversione diretta:

• Ogni cifra deve essere sostituita con la corrispondente parola codice di k bit

Conversione inversa:

• Si raggruppano i bit in gruppi di k a partire dai bit meno significativi (cioè da sinistra a destra)

• Ogni gruppo deve essere sostituito con la corrispondente cifra.

Esempio conversione esadecimale-binario:

A78F $\longrightarrow$ 1010 0111 1000 1111

Rappresentazione di Interi Positivi

- Base=2
- Numero di bit=n
- Numero rappresentazioni: 2^n
- Intervallo: $[0, 2^n[$

  n=5 → massimo rappresentabile: $2^5=32$

  5←→00101

  20←→ 10100

Proprietà: Riporto

Addizione

2+6=?
00010+
00110=

01000

1+31=?
11111+
00001=

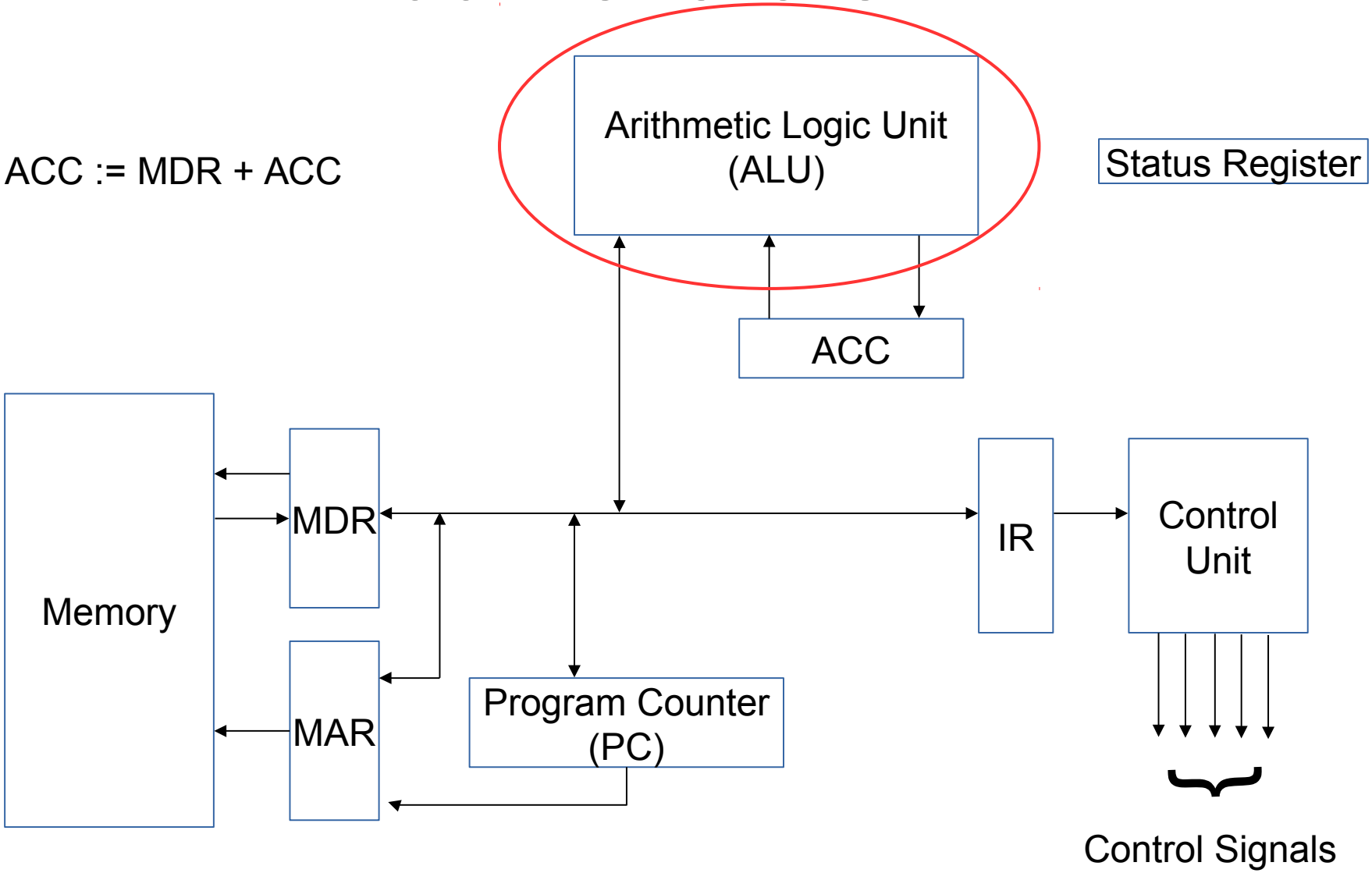
00000 Riporto=1

Le operazioni

- Effettuata la codifica ...
- Quali operazioni realizzare?
- Dove si realizzano le operazioni?
- Come si realizzano le operazioni ?

Addizionatore ?

ACC := MDR + ACC



Half Adder

$s = |a + b|_2$

a	b	S
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

R= (a+b+r)/2

a	b	R
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Half Adder

$s = |a + b|_2$

a	b	S
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

S = a xor b

R= (a+b+r)/2

a	b	R
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

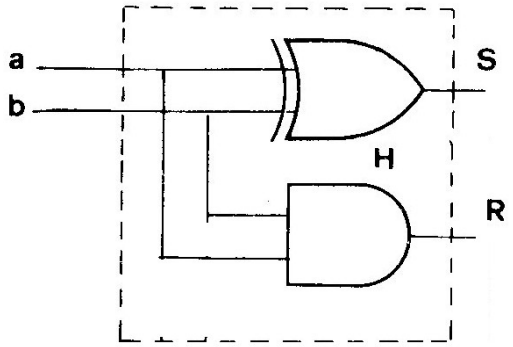
R = a and b

Half Adder

$S = |a + b|_2$

a	b	S
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

S = a xor b



R= (a+b)/2

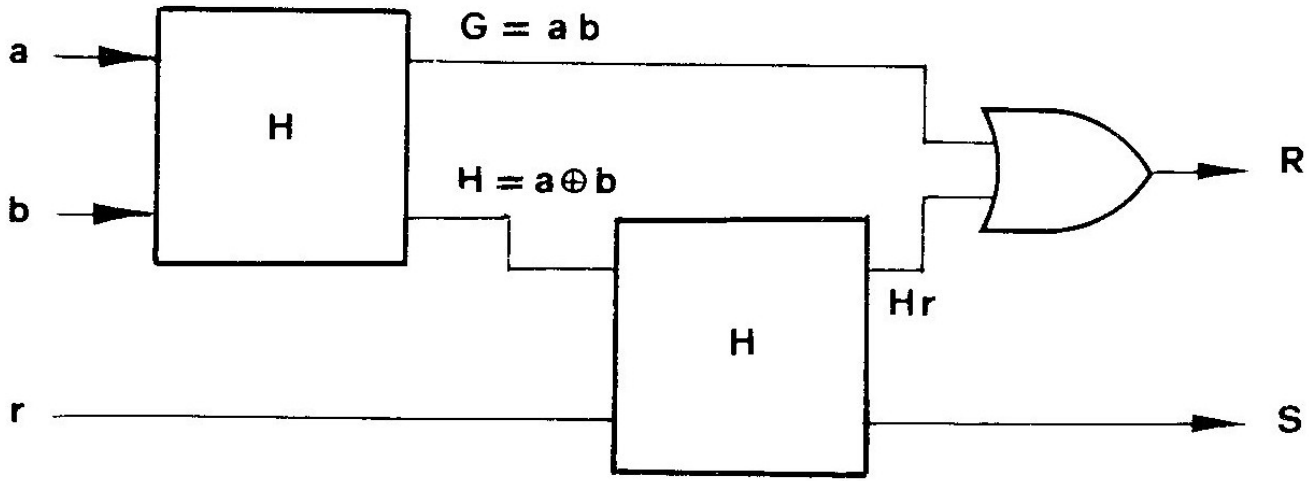
a	b	R
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

R = a and b

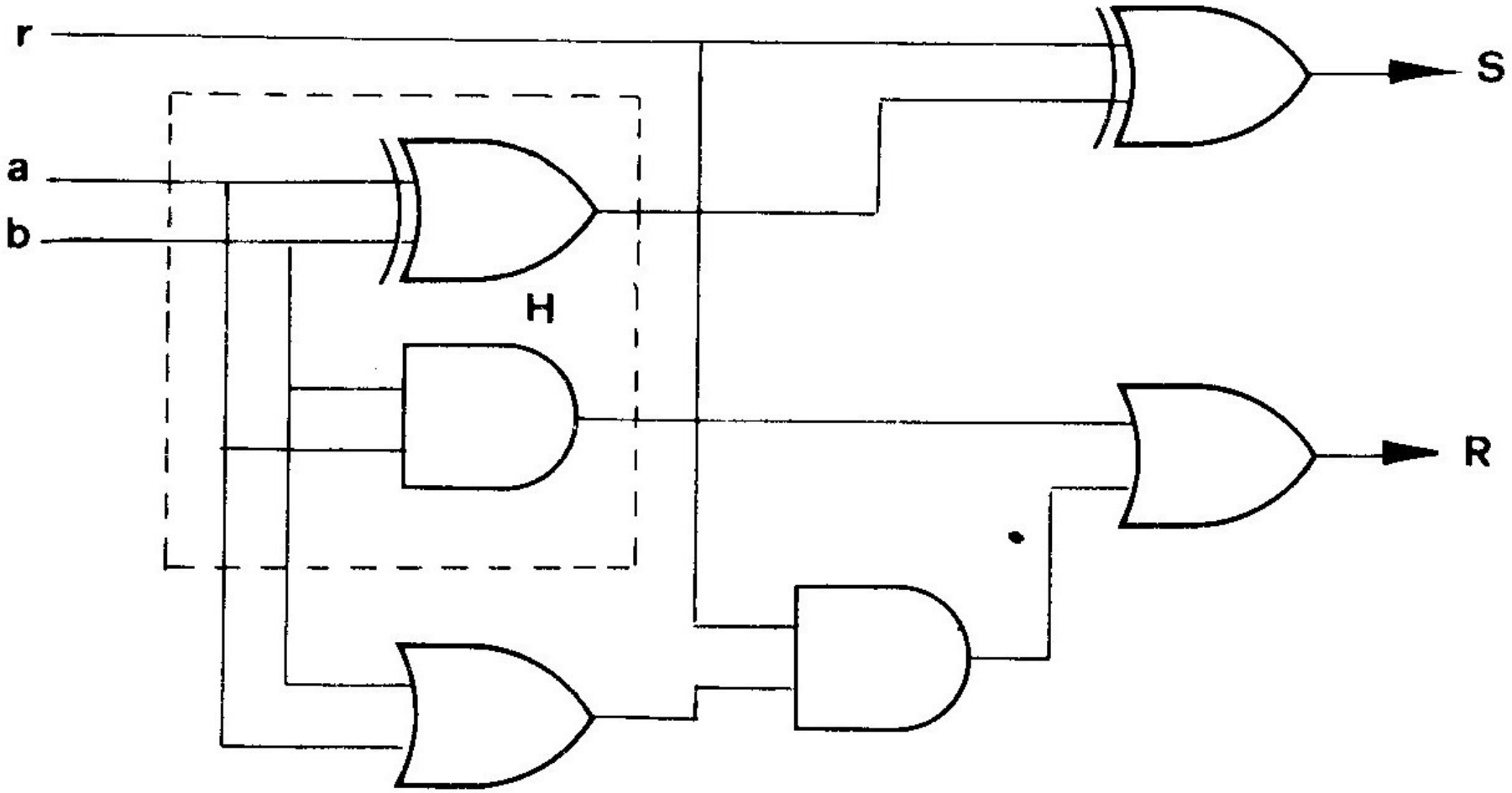
Full Adder

$S = |a + b + r|_2$

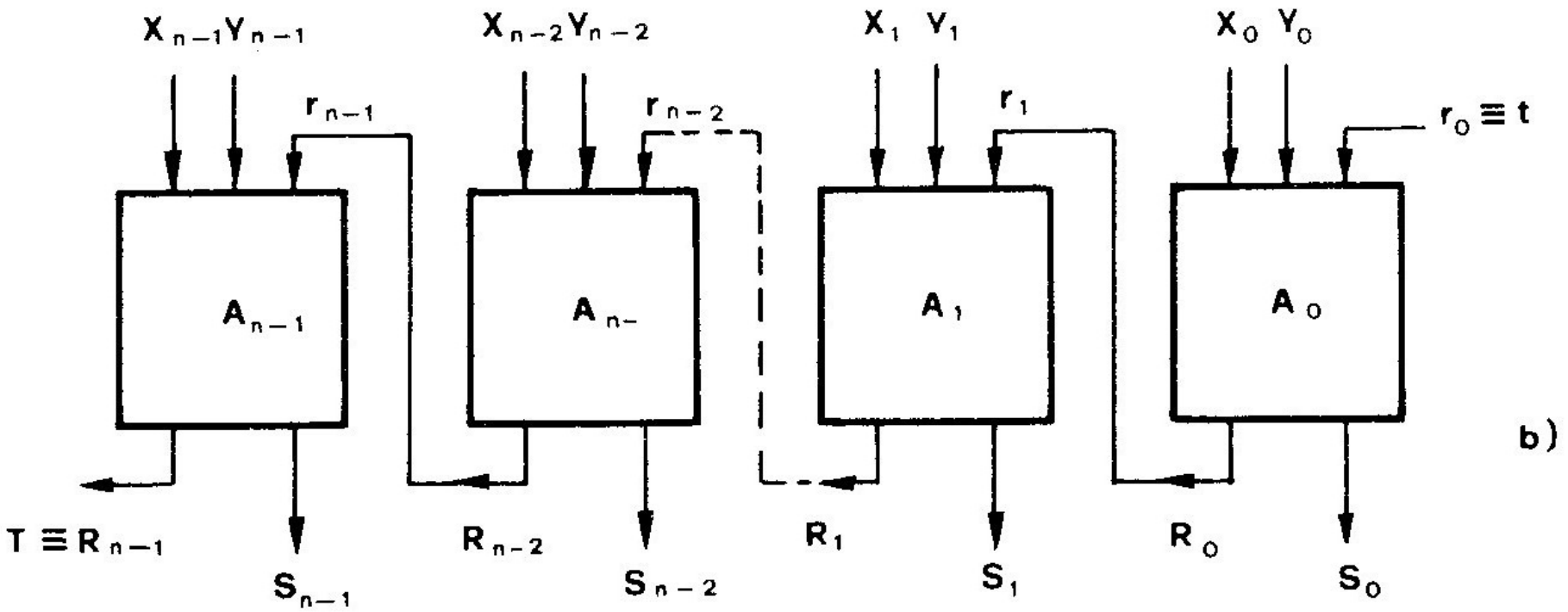
$R = (a+b)/2$



Full Adder



Adder 8 Bit


X n−1 Y n−1
X n−2 Y n−2
X 1 Y 1
X 0 Y 0
r n−1
r n−2
r 1
r 0 ≡ t
A n−1
A n−
A 1
A 0
T ≡ R n−1
S n−1
R n−2
S n−2
R 1
S 1
R 0
S 0
b)

Modulo e Segno

- Rappresentazione interi segno e modulo:
- 1° bit per il segno (1: numeri negativi; 0: numeri positivi)
- Intervallo: $]-b^n/2, b^n/2[$

n=5 ➔ [-15,15]

00000,100000 sono due zeri

15➔01111 -15➔11111

6 ➔00110 -6➔10110

Si opera separatamente su segno e modulo

Operazione Aritmetiche

Rappresentazione modulo e segno

-3-1=?
10011+
10001=

10100

Occorre trattare diversamente numero e segno

15+1=?
01111+
00001=

10000

Overflow !!!!

La migliore codifica

- Modulo e segno è la migliore codifica?
- Ne esistono altre?
- Quali sono utilizzate nella realtà?

Rappresentazione in complementi alla base

- Una seconda tecnica per la rappresentazione dei numeri relativi consiste nel associare a ciascun numero il suo resto modulo $M=2^n$, definito come:

$$|x|_M = x - [x/M] \times M$$

- Questo tipo di codifica, su n bit, è equivalente ad associare:
  - il numero stesso (cioè X=x), ai numeri positivi compresi tra 0 e $2^{n-1} – 1$;
  - il numero $X = 2^n – |x|$, ai numeri negativi compresi tra 2^{n-1} e – 1;
- I numeri rappresentati sono quelli compresi nell'intervallo

$$[–2^n/2;\ 2^{n/}\ /2[$$

Complementi alla base

- Intervallo: $[-b^n/2, b^n/2[$
- Un solo zero

n=5

X=10 Occorre codificare: 10➔01010

X=-10 Occorre codificare: b^n-10=22➔ 10110

Proprietà:

•Come complementare il numero

•Rappresentazione di $-b^k$ 1111111000

•Operazioni con le rappresentazioni

Complementi alla base: la complementazione

- In complementi alla base, a partire dalla rappresentazione di un numero, è anche particolarmente semplice ottenere la rappresentazione del suo opposto.
- È infatti sufficiente *complementare tutti i bit a partire da sinistra, tranne l'uno più a destra ed eventuali zero successivi*.
- Questa ulteriore caratteristica consente di realizzare le sottrazioni attraverso la composizione di una complementazione (nel senso suddetto) ed un'addizione.
- Nell'aritmetica in complementi alla base, di conseguenza, l'addizionatore e il complementatore rappresentano i componenti fondamentali per la realizzazione di tutte le operazioni.

Esempi di complementazione su 4 bit

- La rappresentazione di 6_{10} su 4 bit è 0110_2.
- Complementando tutti i bit tranne l'uno più a destra e gli zero successivi si ottiene: 1010_2.
- 1010_2 è la rappresentazione di –6 in complementi alla base.

- La rappresentazione di 5_{10} su 4 bit è 0101_2.
- Complementando tutti i bit tranne l'uno più a destra e gli zero successivi si ottiene: 1011_2.
- 1011_2 è la rappresentazione di –5 in complementi alla base.

- La rappresentazione di 1_{10} su 4 bit è 0001_2.
- Complementando tutti i bit tranne l'uno più a destra e gli zero successivi si ottiene: 1111_2.
- 1111_2 è la rappresentazione di –1 in complementi alla base.

Operazione per complementi

Basta sommare le rappresentazioni

-3+1=?
11101+
00001=

11110

Risultato=-2

Complementi diminuiti

- La rappresentazione in complementi diminuiti costituisce un'ulteriore alternativa per la codifica dei numeri relativi.
- Concettualmente è analoga alla rappresentazione in complementi alla base.
- La differenza rispetto ad essa è che la legge di codifica dei numeri negativi è leggermente differente:
  - $X=2^n - |x|;$ (complementi alla base)
  - $X=2^n - 1 - |x|;$ (complementi diminuiti)
- I numeri rappresentabili, se si utilizzano n bit, sono quelli compresi nell'intervallo $[-(2^{n-1} - 1); 2^{n-1} - 1]$.
- Anche in questo caso, quindi, le cifre rappresentabili sono $2^n - 1$ e ancora una volta è lo zero ad avere una doppia rappresentazione.

Complementi diminuiti

- Intervallo: $]-b^n/2, b^n/2[$
- Un solo zero

n=5

X=10 Occorre codificare: 10➔01010

X=-10 Occorre codificare: $(b^n-1)-10=21$➔ 10101

Proprietà:

•Più facile il complemento

•Rappresentazione di $-b^k$ 1111101111

•Operazioni con le rappresentazioni

Complementi diminuiti: perché?

- La rappresentazione in complementi è suggerita dalla maggiore semplicità con cui è possibile calcolare la rappresentazione dell'opposto di un numero, a partire dalla rappresentazione del numero stesso.
- A questo proposito, infatti, basta semplicemente complementare tutti i bit della rappresentazione indistintamente.

- Esempio:
  - la rappresentazione in complementi diminuiti su 4 bit di 4 è 0100;
  - complementando tutti i bit si ottiene 1011;
  - 1011 è la rappresentazione in complementi diminuiti su 4 bit di –4.

  - la rappresentazione in complementi diminuiti su 4 bit di – 6 è 1001;
  - complementando tutti i bit si ottiene 0110;
  - 0110 è la rappresentazione in complementi diminuiti su 4 bit di –6.

I componenti dell'aritmetica in complementi diminuiti.

- I componenti fondamentali per implementare un'aritmetica in complementi diminuiti sono:
  - ancora l'addizionatore modulo-2^n (e non 2^{n-1});
  - un complementatore.
- l'addizionatore modulo-2^n è infatti di più semplice realizzabilità.
- Il risultato però deve essere opportunamente "corretto" per renderlo compatibile con l'aritmetica in modulo 2^{n-1}.
- In particolare deve essere aggiunta un'unità al risultato nei seguenti casi:
  - se entrambi gli addendi sono negativi;
  - se un addendo è positivo, l'altro negativo e la somma è positiva.
- Nei casi suddetti l'aritmetica degli interi positivi darebbe overflow.
- L'overflow quindi può essere interpretato come la necessità di effettuare la correzione.

Operazione per complementi diminuiti

-3+1=?

11100+

00001=

11101

Complementi diminuiti

Risultato=-2

Esempi di somme in complementi diminuiti.

– 2+		1101+
– 3=		1100=
---	→	---
– 5		**1**1001+
	overflow	1=

		1010

Somma di due numeri negativi.
Si è generato overflow nell'aritmetica degli interi positivi.
Necessita la correzione.

5+		0101+
– 2=		1101=
---	→	---
3		**1**0010+
	overflow	1=

		0011

Somma di un numero positivo e un numero negativo.
Il risultato è positivo.
Si è generato overflow nell'aritmetica degli interi positivi
Non necessita alcuna correzione.

3+		0011+
– 4=		1011=
---	→	---
– 1		1110

Somma di un numero positivo e un numero negativo.
Il risultato è negativo.
Non si è generato overflow nell'aritmetica degli interi positivi
Non necessita alcuna correzione.

Rappresentazione in virgola fissa dei numeri reali.

- Quando di un numero frazionario si rappresentano separatamente la parte intera e la parte frazionaria si parla di rappresentazione in *virgola fissa*.
- La rappresentazione dei due contributi (che sono numeri interi) può essere realizzata secondo una delle tecniche viste in precedenza.
- In questo caso la posizione della virgola è fissa e resta sottintesa.

Rappresentazione in virgola mobile dei numeri reali

- Un numero reale x può essere rappresentato dalla coppia

$$(m,e)$$

tale che:

$$x = m \times b^e$$

dove:

  - *m* è detta *mantissa*;
  - *e* è detto *esponente*;
  - *b* è la base di numerazione adottata.
- Il metodo, in questo caso, prende il nome di *codifica in virgola mobile*.

Normalizzazione

- Per ciascun numero esistono infinite coppie che lo rappresentano.
- Esempio (b=10):
  - 346.09801 è rappresentato da
    - » (346.09801, 0) oppure
    - » (346098.01, -3) oppure
    - » (0.034609801, 4) etc…

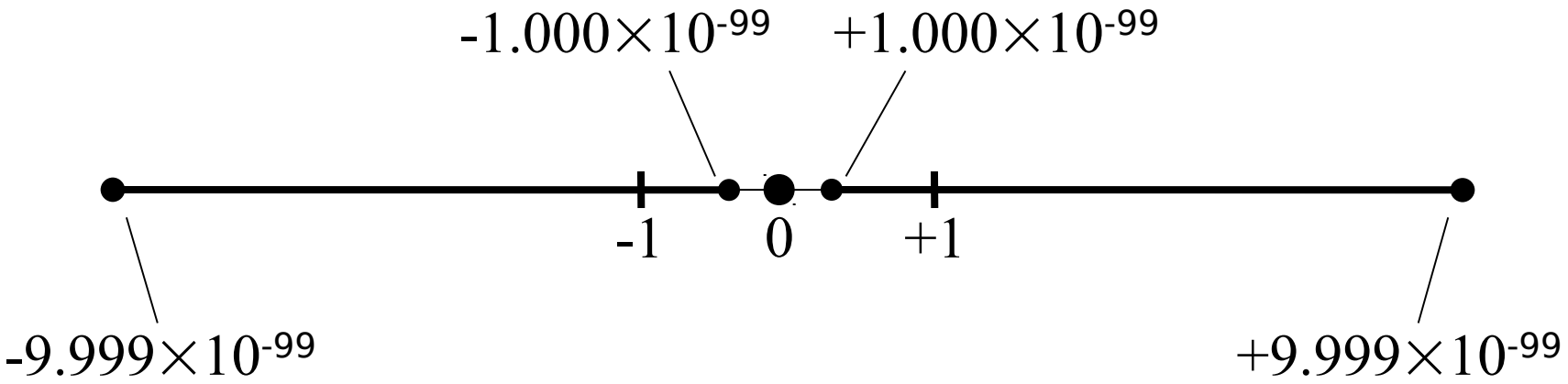
- Allo scopo di uniformare le rappresentazioni, si fissa convenzionalmente la posizione della virgola subito dopo la prima cifra significativa, ottenendo l'*unico rappresentante*:

(3.4609801, 2)

Esempio: intervallo di rappresentazione

- Con b=10, usando 4 cifre per *m* e 2 per *e* (più due bit per i relativi segni), l'insieme rappresentabile (utilizzando solo rappresentazioni normalizzate) è:

$$[-9.999\times10^{99},-1.000\times10^{-99}]\cup\{0\}\cup[+1.000\times10^{-99},+9.999\times10^{99}]$$



Approssimazione

- Come è facile verificare, in questo tipo di rappresentazione l'approssimazione non è costante.
- In particolare la precisione assoluta è molto spinta in prossimità dello zero e va diminuendo progressivamente a mano a mano che il numero aumenta (in valore assoluto).
- Ad esempio:
  - in prossimità dello zero l'errore massimo che può essere commesso è pari a $1.001 \times 10^{-99} - 1.000 \times 10^{-99} = 0.001 \times 10^{-99}$;
  - in prossimità dell'estremo superiore dell'intervallo di rappresentazione, invece, l'errore massimo che si può commettere è $9.999 \times 10^{99} - 9.998 \times 10^{99} = 0.001 \times 10^{99}$.
- Si commettono quindi "errori piccoli" su "numeri piccoli" ed "errori grandi" su "numeri grandi".
- Quello che resta inalterato è invece l'errore relativo, costante su tutto l'asse di rappresentabilità.

Overflow e Underflow

- L'errore relativo dipende dal numero di cifre della mantissa.
- Gli estremi dell'intervallo di rappresentazione dipendono dal numero di cifre dell'esponente.

- Nel caso precedente di 2 cifre per l'esponente, si ha overflow per numeri maggiori (in modulo) di 10^{99} e si ha underflow per numeri minori (in modulo) di 10^{-99}.

L’algebra di Boole - richiami

Operazioni fondamentali sui bit:

x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

Congiunzione

x AND y si indica anche con $x \cdot y$

x	y	x OR y
0	0	0
0	1	1
1	0	1
1	1	1

Disgiunzione

x OR y si indica anche con $x + y$

x	NOT x
0	1
1	0

Negazione

NOT x si indica anche con $\overline{x}$

L’algebra di Boole - alcune proprietà (1)

- **Proprietà commutativa**:
  x AND y = y AND x x OR y = y OR x

- **Proprietà associativa:**
  (x AND y) AND z = x AND (y AND z)
  (x OR y) OR z = x OR (y OR z)
  per la propr. associativa posso definire AND e OR a più di due operandi (es. x AND y AND z)

- **Proprietà di idempotenza e assorbimento:**
  x AND x = x x OR x = x
  x AND (x OR y) = x x OR (x AND y) = x

L’algebra di Boole - alcune proprietà (2)

- **Proprietà distributiva**
  x AND (y OR z) = (x AND y) OR (x AND z)
  x OR (y AND z) = (x OR y) AND (x OR z)

- **Proprietà di convoluzione**
  NOT (NOT x) = x

- **Proprietà del minimo e del massimo:**
  x AND 1 = x x AND 0 = 0
  x OR 0 = x x OR 1 = 1

- **Leggi di De Morgan:**
  NOT (x AND y) = (NOT x) OR (NOT y)
  NOT (x OR y) = (NOT x) AND (NOT y)

Funzioni booleane

$y = f(x_1, x_2, \ldots, x_n)$ è una funzione booleana se ad ogni ennupla di valori booleani $x_1,\ldots,x_n$ associa un valore booleano y

Due esempi:

x_1	x_2	$f(x_1,x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

Questa funzione è detta OR esclusivo, o XOR

x_1	x_2	$f(x_1,x_2)$
0	0	1
0	1	0
1	0	0
1	1	1

Questa funzione è detta equivalenza, o EQU

Anche AND, OR e NOT sono funzioni booleane. Esse vengono dette *funzioni fondamentali* dell'algebra

Insiemi funzionalmente completi

Si può dimostrare che qualsiasi funzione booleana può essere calcolata applicando le funzioni AND, OR, e NOT. Ad esempio:

x XOR y = (x AND NOT y) OR (y AND NOT x)

Per questo, l'insieme {AND, OR, NOT} si dice *funzionalmente completo*.

Esistono altri insiemi funzionalmente completi. Si noti che grazie alle leggi di De Morgan si può costruire la AND da {OR, NOT}, oppure la OR da {AND, NOT}. Quindi anche {AND, NOT} e {OR, NOT} sono insiemi funzionalmente completi.

Reti logiche

I valori booleani possono essere rappresentati da grandezze elettriche. Ad esempio:

0 <=> tensione di 0 Volt

1 <=> tensione di +5 Volt

In tal caso le funzioni booleane possono essere realizzate mediante circuiti elettronici detti *reti logiche*.

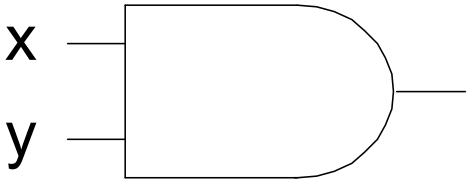
Nelle reti logiche *unilaterali*, leuscite della rete corrispondono a valori di grandezze elettriche misurate in opportuni punti del circuito; il flusso dell'elaborazione procede fisicamente in un'unica direzione, dai segnali di ingresso verso i segnali di uscita.

Nelle reti logiche *bilaterali*, invece, l'uscita della rete è determinata dalla presenza o dall'assenza di "contatto" tra due punti della rete.

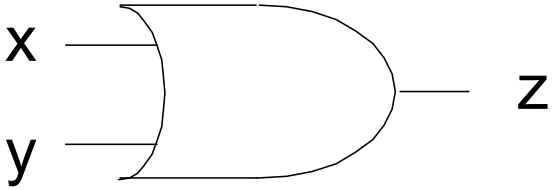
Porte logiche (*gates*)

Circuiti logici elementari che realizzano le operazioni fondamentali. Le reti logiche si costruiscono connettendo più porte logiche.

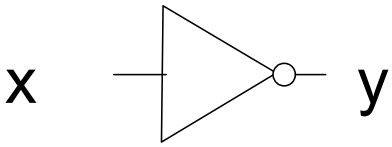
Simboli delle principali porte logiche:



z = x AND y



z = x OR y



y = NOT x

(se connesso a un'altra porta, il NOT si indica talora con un semplice pallino)

Operatori logici generalizzati (1)

Dato un vettore di variabili booleane $X = (x_1, x_2, \ldots, x_n)$, e una variabile booleana α, indicheremo con la notazione:

$Y = \alpha \; OP \; X$ (dove *OP* è un operatore booleano)

l'operazione che produce il vettore booleano Y così definito:

$Y = (y_1, y_2, \ldots, y_n)$ con

$y_1 = \alpha \; OP \; x_1$

. . . . . .

$y_n = \alpha \; OP \; x_n$

Esempio:

α AND X ha come risultato il vettore formato da:

(α AND x_1, α AND x_2, …, α AND x_n)

Operatori logici generalizzati (2)

Dati due vettori di variabili booleane $X = (x_1, x_2, \ldots, x_n)$ e $Y = (y_1, y_2, \ldots, y_n)$ indicheremo con la notazione:

$Z = X \; OP \; Y$ (dove *OP* è un operatore booleano)

l'operazione che produce il vettore booleano Z così definito:

$Z = (z_1, z_2, \ldots, z_n)$ con

$z_1 = x_1 \; OP \; y_1$

. . . . . .

$z_n = x_n \; OP \; y_n$

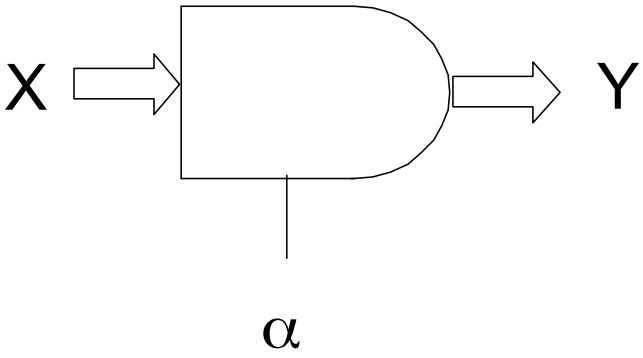
Esempio:

X OR Y ha come risultato il vettore formato da:

$(x_1 \text{ OR } y_1, x_2 \text{ OR } y_2, \ldots, x_n \text{ OR } y_n)$

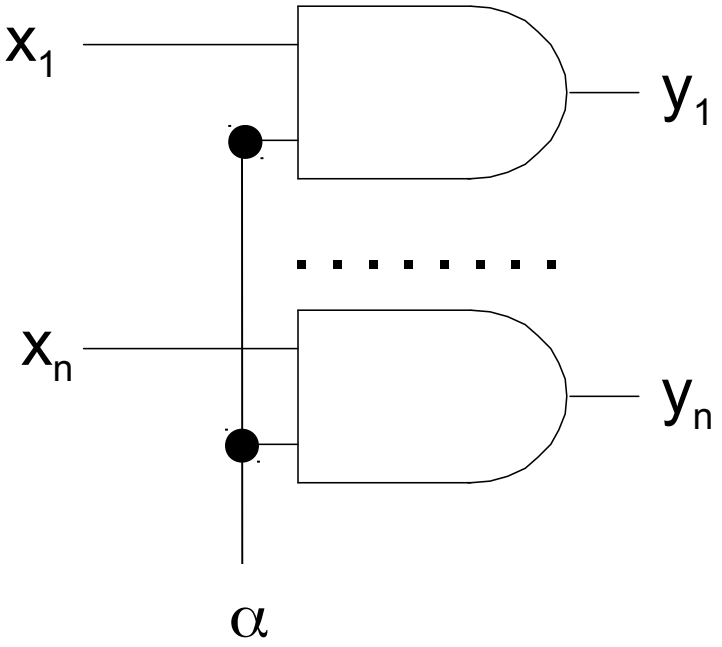
Porte logiche generalizzate (1)

Rappresentazione simbolica:



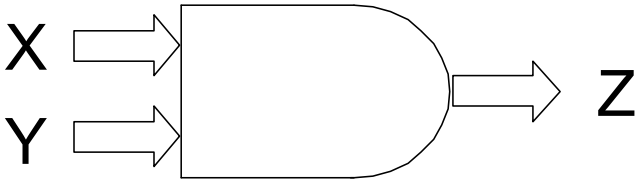
$Y = \alpha \text{ AND } X$

Circuito equivalente:



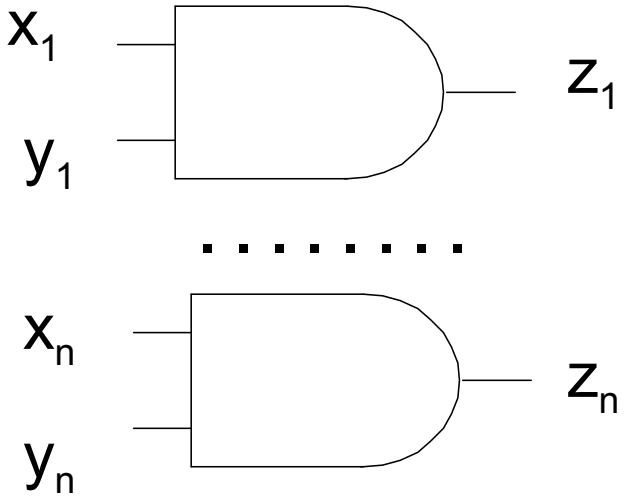
Porte logiche generalizzate (2)

Rappresentazione simbolica:



Z = X AND Y

Circuito equivalente:



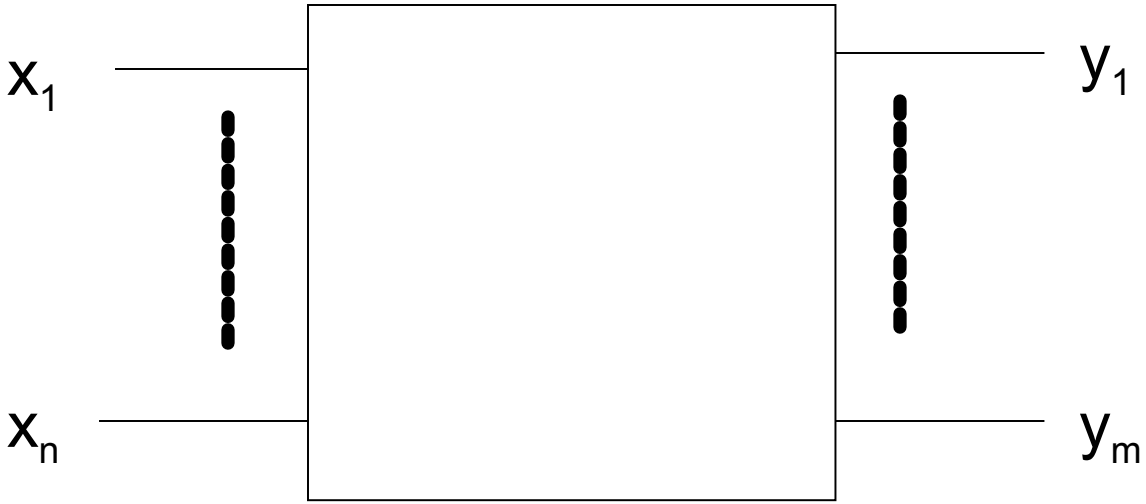
Macchine combinatorie (1)

Reti logiche con *n* ingressi $x_1, x_2, \ldots, x_n$ e *m* uscite $y_1, y_2, \ldots, y_m$ che realizzano la corrispondenza:

$$y_1 = f_1(x_1, x_2, \ldots, x_n)$$

. . . . . .

$$y_m = f_m(x_1, x_2, \ldots, x_n)$$



Macchine combinatorie (2)

In una macchina combinatoria i valori delle uscite dipendono esclusivamente dai valori degli ingressi.

In una macchina combinatoria ideale tale dipendenza è istantanea; in una macchina reale c'è sempre un ritardo tra l'istante in cui c'è una variazione in uno degli ingressi e l'istante in cui l'effetto di questa variazione si manifesta sulle uscite.