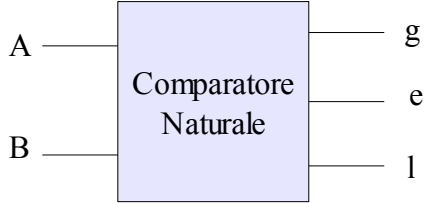
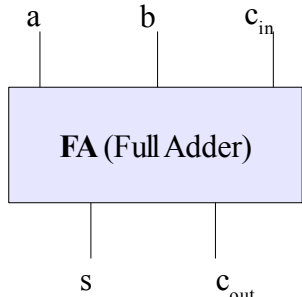
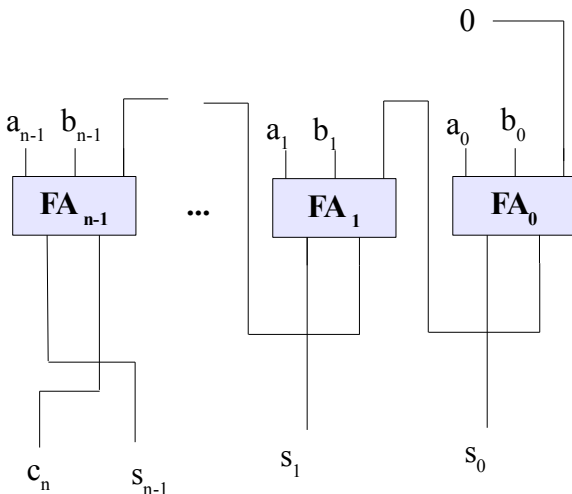
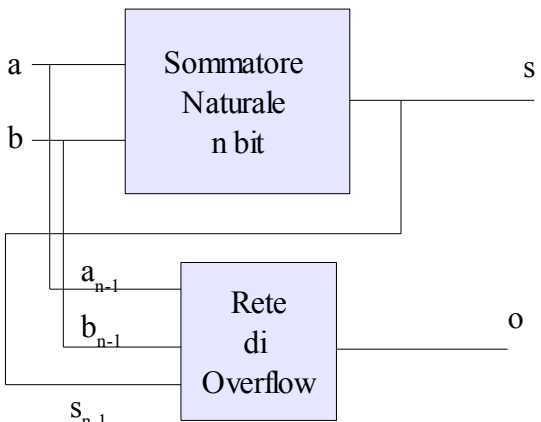
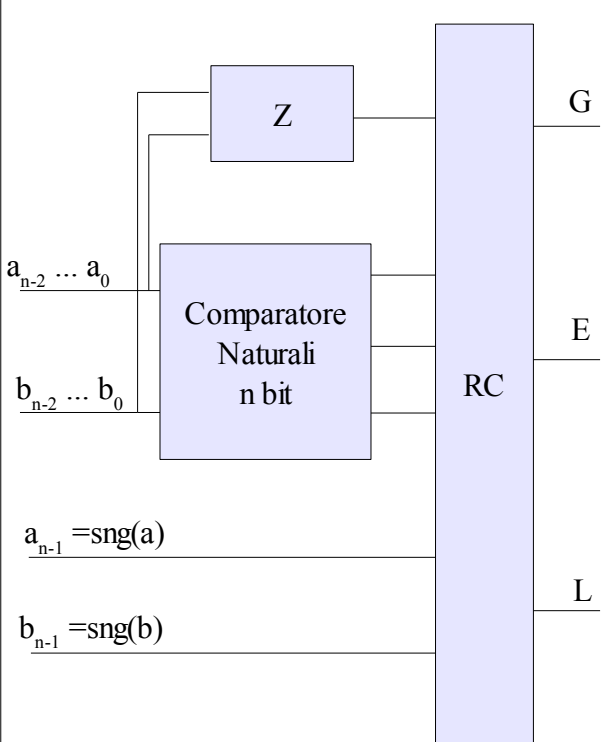
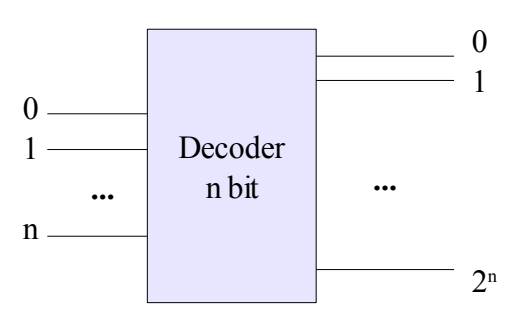


# Calcolatori

## Reti Combinatorie

<p><b>COMPARATORE NUMERI NATURALI:</b> Compara due numeri naturali A e B e restituisce tre bit (g,e,l) che sono asseriti rispettivamente se <math>A &gt; B</math>, <math>A = B</math> o se <math>A &lt; B</math>. Sono mutuamente esclusivi. <b>INPUT:</b> A e B naturali <b>OUTPUT:</b> g, e, l</p>	
<p><b>FULL ADDER (SOMMATORE AD UN BIT):</b> Somma due bit (a e b) tenendo conto del riporto precedente o <i>carry</i> (<math>c_{in}</math>) e produce in uscita il bit di somma (s) ed il bit di riporto o <i>carry</i> (<math>c_{out}</math>) <b>INPUT:</b> a, b, <math>c_{in}</math> <b>OUTPUT:</b> s, <math>c_{out}</math></p>	
<p><b>SOMMATORE AD n BIT:</b> Il sommatore per numeri naturali di grandezza arbitraria n viene realizzato mettendo in cascata n Full Adder e collegando il riporto prodotto dall' <i>i</i>-esimo Full Adder al carry d'ingresso dell' (<i>i+1</i>)-esimo Full Adder (<b>Ripple-Carry</b>) Naturalmente il primo FA dovrà avere carry d'ingresso = 0 <b>INPUT:</b> a,b di lunghezza arbitraria n <b>OUTPUT:</b> s risultato di lunghezza n, <math>c_n</math> bit di riporto della somma</p>	
<p><b>SOMMATORE IN C2:</b> Sommare due numeri a e b espressi in complemento a 2 corrisponde ad effettuare (per una proprietà del C2) una somma di due numeri naturali, non tenendo conto del riporto (carry) e tenendo di conto di una possibile situazione di overflow (che si verifica se a e b hanno segno concorde ed il risultato s ha segno discorde da questi) <b>INPUT:</b> a,b di lunghezza n in C2</p>	

<p><b>OUTPUT:</b> s risultato di lunghezza n, o bit di overflow</p>	
<p><b>COMPARATORE M&amp;S:</b>          Compara due numeri binari (a e b) di lunghezza n espressi in modulo e segno. Viene implementato comparando prima i moduli (<math>a_{n-2}, \dots, a_0</math> e <math>b_{n-2}, \dots, b_0</math>) e tramite una rete combinatoria che gestisce i risultati tenendo di conto del segno di a e del segno di b (ad esempio se <math>a &gt; b</math> in modulo ma <math>\text{sgn}(a) = -1</math> e <math>\text{sgn}(b) = 1</math> allora <math>b &gt; a</math>)</p> <p><b>PROBLEMA:</b> lo zero viene rappresentato in due modi diversi nella codifica M&amp;S (<math>+0 = 000..0</math> e <math>-0 = 100..0</math>) per gestire tale problema (e non rischiare che la rete combinatoria reputi diversi per via del segno due numeri che in realta' sono uguali) viene aggiunto un altro modulo combinatorio (<b>Z</b>) che ha l'unico compito di controllare il modulo di a ed il modulo di b ed asserire il suo bit d'uscita se <math> a  =  b  = 0</math>.</p> <p><b>INPUT:</b> a, b di lunghezza n in M&amp;S</p> <p><b>OUTPUT:</b> G, E, L asseriti se <math>a &gt; b</math>, <math>a = b</math>, <math>a &lt; b</math></p>	
<p><b>DECODER SU n BIT:</b>          Il decoder è un dispositivo ad n ingressi e <math>2^n</math> uscite. In base alla configurazione in ingresso (ad esempio 101 in un caso di decoder a 3 bit) tale rete combinatoria asserisce l'uscita corrispondente (nell'esempio la 5°) tra quelle possibili. Superfluo dire che le uscite sono mutuamente esclusive.</p> <p><b>INPUT:</b> n bit</p> <p><b>OUTPUT:</b> <math>2^n</math> bit (tutte le possibili configurazioni degli ingressi)</p>	
<p><b>ENCODER SU n BIT:</b>          Come il decoder ma il funzionamento è esattamente opposto. Deve essere asserita solo una linea d'ingresso. In</p>	<p style="text-align: center;"><b>&lt; vedi sopra &gt;</b>  <b>&lt; con ingressi ed uscite invertiti &gt;</b></p>

caso questo requisito venga a mancare si ha un uscita non specificata

**INPUT:**  $2^n$  bit

**OUTPUT:** n bit

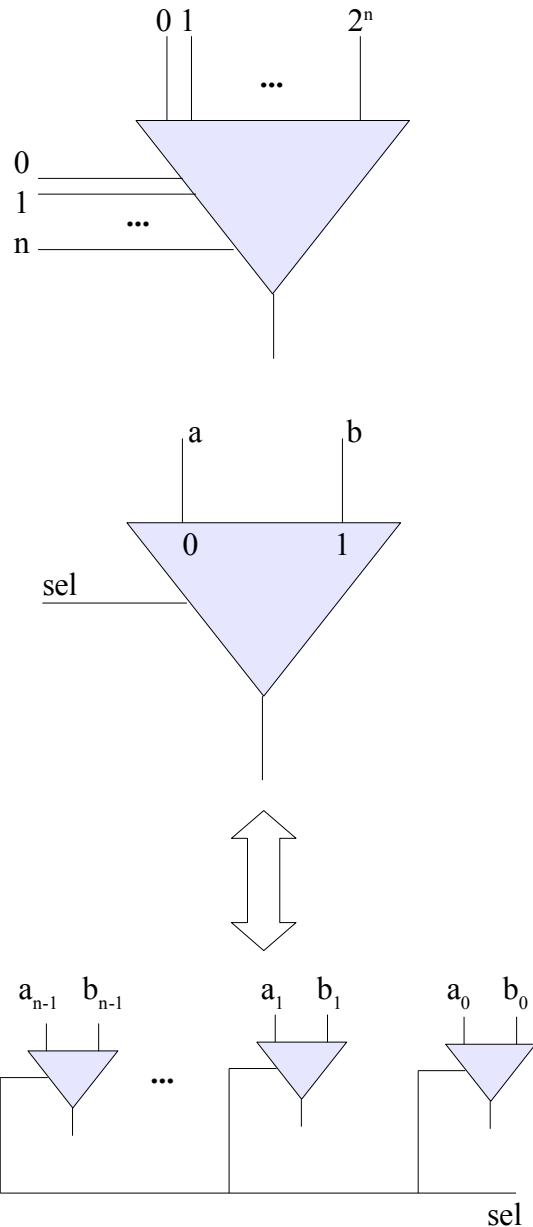
**MULTIPLEXER:**

Il multiplexer e' un selettore di ingressi. Esso ha, nel caso generale,  $2^n$  ingressi "di selezione" ed n ingressi "selettivi". Questi n ingressi codificano uno ed un solo ingresso che verra' scelto e trasferito sul bit d'uscita del componente. Facendo un esempio informatico, esso è simile ad uno *switch*. Analogamente il **demultiplexer** (poco usato in questo corso) effettua l'operazione inversa: a partire da un ingresso, in base a n bit "selettivi" viene scelta una delle  $2^n$  uscite sulle quali trasferire l'ingresso.

In alcuni casi pero', si ha il problema di dover scegliere tra piu' parole anziche' bit, ad esempio scegliere se mandare in uscita a o b parole di n bit (anche l'uscita dovra' essere di n bit chiaramente). Dovremmo usare in questo caso n multiplexer. Ma notiamo che tali selettori sono azionati dal medesimo bit selettivo, e pertanto è facilmente creabile un multiplexer con ingressi a n bit

**INPUT:**  $2^n$  bit tra cui scegliere, n bit selettivi

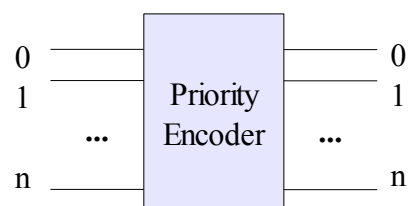
**OUTPUT:** 1 bit scelto



**PRIORITY ENCODER:**

Ad ogni ingresso è associato un valore di prioritá (generalmente decrescente dall'ingresso 0 ... n-1) ed in uscita viene asserita solo la linea corrispondente alla prioritá maggiore.

**INPUT:** n bit



**OUTPUT:** n bit (mutuamente esclusivi)

**ALU:**

La ALU (Arithmetic Logic Unit) unisce molti dei concetti precedentemente visti per creare una rete combinatoria capace, in base a dei flag d'ingresso, di eseguire molteplici operazioni tra cui:

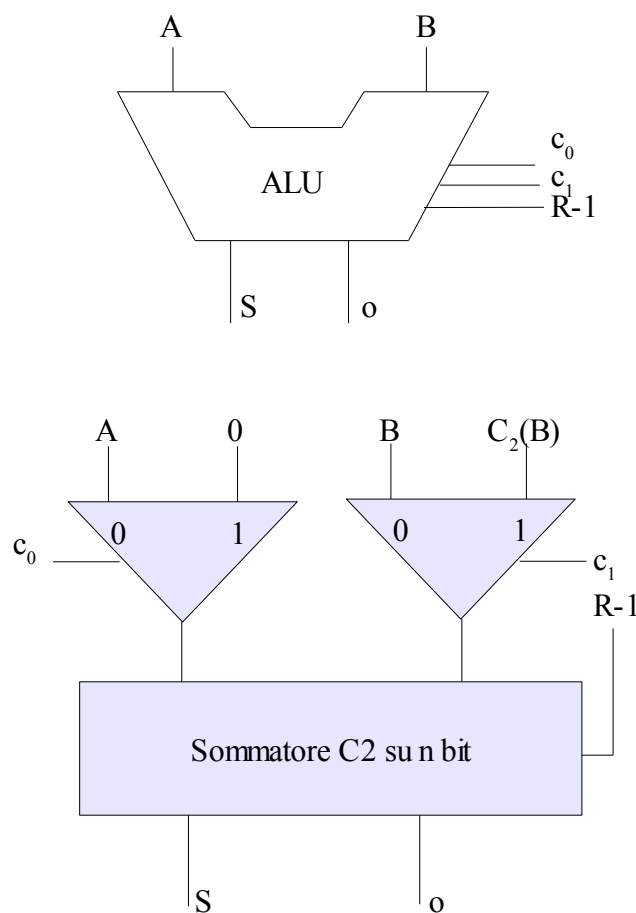
- selezione di B
- incremento di B ( $B+1$ )
- complementazione di B ( $\bar{B}$ )
- cambio segno di B ( $-B$ )
- somma di A e B ( $A+B$ )
- sottrazione di A e B ( $A-B$ )

Per realizzare tutto questo abbiamo bisogno di un sommatore C2 su n bit e di due multiplexer: uno tra A e 0 che decide se le operazioni saranno su due operandi o se su uno soltanto, ed uno che sceglie tra B ed il complemento di B. Quindi per ora abbiamo due flag ( $c_0$ ,  $c_1$ ) selettori dei multiplexor. Ma per incrementare B, cambiare di segno a B ( $\bar{B}+1$ ) o sottrarre B ad A ( $A+\bar{B}+1$ ) ho bisogno di aggiungere un bit alla somma. Faccio questo mediante il bit di carry del sommatore.

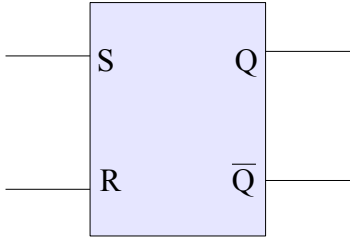
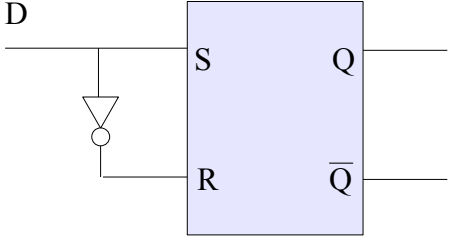
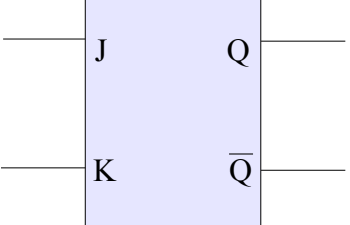
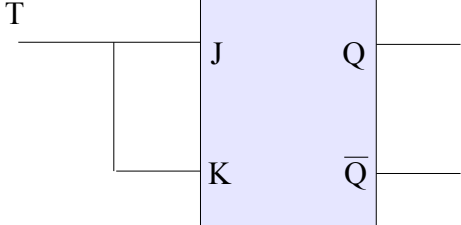
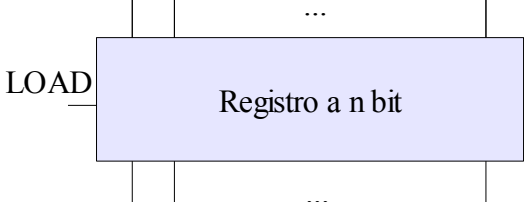
Senza molte difficoltà realizzo un encoder nel quale asserendo bit di ingresso come ADD, SUB, INCR\_B, ... codifica  $c_0$ ,  $c_1$  ed il carry per eseguire le operazioni richieste.

**INPUT:** A,B parole di n bit, elenco di flag (ADD, SUB, ...) mutuamente esclusivi

**OUTPUT:** s parola di n bit rappresentante il risultato dell'operazione richiesta, bit di overflow



## Reti Sequenziali

<p><b>FLIP-FLOP SR:</b></p> <p>Il flip flop SR e' ottenuto aggiungendo al LETCH due porte and che attivano la transazione solo se il CLK è positivo. Quindi se S=0 e R=0, manitene lo stato corrente, se S=1 (set) pone lo stato futuro ad 1, se R=1 (reset) pone lo stato futuro a 0: La situazione S=1 ed R=1 non deve mai verificarsi.</p> <p><b>FUNZIONE:</b> <math>Q' = S + \bar{R}Q</math></p>	
<p><b>FLIP-FLOP D:</b></p> <p>Aggiungendo una negazione al flip-flop SR otteniamo il flip-flopo D. Tale macchina a stati porta esattamente ciò che ha in ingresso come suo stato futuro. Se l'ingresso e' 0 si ha <math>Q' = 0 + 0Q = 0</math> se l'ingresso e' 1 invece <math>Q' = 1 + 1Q = 1</math>.</p> <p><b>FUNZIONE:</b> <math>Q' = Q</math></p>	
<p><b>FLIP-FLOP JK:</b></p> <p>Il flip-flop JK funziona come l'SR salvo il fatto che il caso S=1 R=1 non è più inammissibile, ma in questo caso attiva la funzione <i>toggle</i> che inverte lo stato corrente (<math>Q' = \bar{Q}</math>)</p> <p><b>FUNZIONE:</b> <math>Q' = J \bar{Q} + \bar{K}Q</math></p>	
<p><b>FLIP-FLOP T:</b></p> <p>Collegando le uscite del JK ad un unico ingresso comune, otteniamo il T. Si vede facilmente che se l'ingresso è 0 si ha una fase di Hold, viceversa lo stato cambia.</p> <p><b>FUNZIONE:</b> <math>Q' = T\bar{Q} + \bar{T}Q</math></p>	
<p><b>REGISTRO:</b></p> <p>Come visto sopra un flip-flop e' un componente dotato di memoria di un bit. Ora, ponendo vari di questi componenti in cascata possiamo ottenere un componente capace di</p>	

memorizzare  $n$  bit. Sarà inoltre possibile (tramite una rete combinatoria che funge da interfaccia e tramite un segnale di LOAD) modificare il contenuto di tale memoria. Lo stato è proiettato direttamente sull'uscita e pertanto sempre "accessibile" (non c'è bisogno di un segnale per abilitare l'output del dato in memoria)

**INPUT:** il dato da inserire nel registro, LOAD per abilitare l'immissione di quel dato

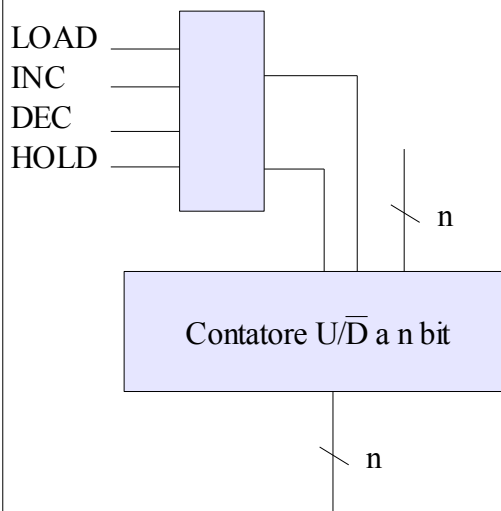
**OUTPUT:** lo stato rappresentante il valore contenuto nel registro

### CONTATORI UP/DOWN:

Non è difficile realizzare, a partire da dei JK o dei flip-flop T, un oggetto il cui stato commuta ad ogni colpo di CLK. Inoltre, con poca logica sequenziale intorno e ponendo in cascata oggetti del genere (facendo sì che l' $i+1$ -esimo commuti sul fronte di salita/discisa dell' $i$ -esimo) è possibile realizzare oggetti il cui stato complessivo è su  $n$  bit che rappresenta un numero naturale codificato in binario che viene incrementato/decrementato di 1 ogni colpo di CLK. Tale oggetto viene detto contatore.

Tale dispositivo è capace di contare solo da 0 a  $2^n-1$ . Mediante l'utilizzo di una **rete di controllo** (vedi sotto) appropriata è possibile generalizzare il concetto di contatore. Per fare questo sono necessari 4 segnali di controllo (codificati poi da un encoder):

- il INC che abilita il contatore a contare UP
- il LOAD che permette di caricare una dato all'interno del contatore attraverso degli appositi ingressi e salvarlo nello



stato (vedi registro)

- DEC che abilita il contatore a contare DOWN
- HOLD manitene lo stato corrente

**INPUT:** LOAD, INC, DEC, HOLD, il possibile dato da immettere nel contatore tramite LOAD

**OUTPUT:** lo stato totale rappresentante il numero "contato"

### SHIFT REGISTER:

Lo *shift register* e' un registro a scorrimento molto utile in particolari circostanze. Tramite dei segnali di controllo permette di:

- HOLD mantenere il contenuto del registro invariato (piu' una possibile codifica degli ingressi che un vero e proprio segnale)
- LOAD caricare un dato nel registro
- SHIFTR spostare l'intero contenuto del registro di un bit verso destra ed inserire un dato in ingresso (IL) nel posto vuoto a sinistra (nota: il bit meno significativo viene perso)
- SHIFTL analogo allo SHIFTR ma verso sinistra. Il dato viene reperito in IR ed il bit piu' significativo ad essere perso.

