3 - Principali componenti elettronici, PLA e ALU

DECODER

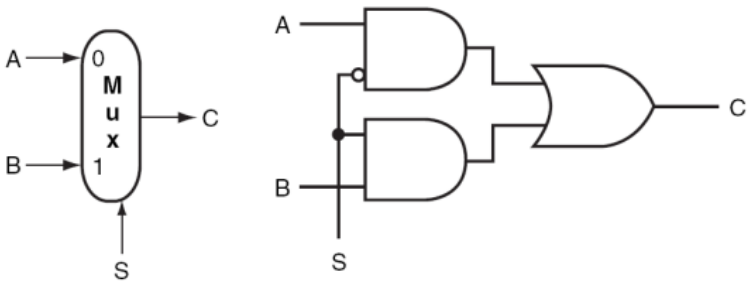
📖 Il **decoder** è un componente elettronico che ha n ingressi e e 2^n uscite. Esso converte i numeri binari in base 10.

Esempio: dati 3 bit A, B, C in ingresso, se tutti valgono 0 l'output della porta 0 sarà 1 (gli altri output varranno 0); se A=0, B=0 e C=1 l'output della porta 1 sarà 1 (gli altri output varranno 0); se A=0, B=1 e C=0 l'output della porta 2 sarà 1 (gli altri output varranno 0) ecc.

MULTIPLEXOR

📖 Il **multiplexor**, o selettore, è un componente elettronico che ha n ingressi principali, $\log_2 n$ entrate di controllo (label dei selettori) e 1 uscita.

$$C = (A \cdot \overline{S}) + (B \cdot S)$$



Multiplexor

Il funzionamento di un multiplexor si identifica in quello della porta logica XOR (*switch-case* in Java).

LOGICHE A 2 LIVELLI E PLA

Utilizzando le porte **AND**, **NOT** e **OR** è possibile implementare qualunque funzione logica più complessa.

E' possibile creare diverse **logiche a due livelli** (ovvero, il segnale incontra al massimo due perte logiche nel suo percorso):

- **somma di prodotti** (OR di più AND).
- **prodotto di somme** (AND di più OR).

In particolare, la logica maggiormente utilizzata la somma di prodotti: essa è comunemente detta **PLA** (Programmable Logic Unit). La PLA riceve in input determinati valori ed i lro complementi, si compone di un primo livello dotato di porte AND e di un secondo livello dotato, generalmente, di una singola porta OR.

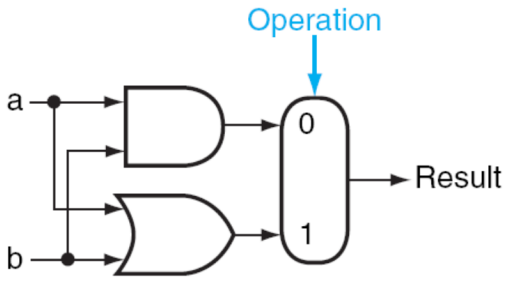
ALU

📖 Un'**ALU** è un'unità aritmetico-logica, basata solitamente su logica PLA, che svolge le operazioni di base di un calcolatore (somma, sottrazione, AND, OR...). Le unità di base di un'ALU sono l'**AND**, l'**OR**, l'**inverter (NOT)** e il **multiplexor**.

Un'ALU può essere costruita a **1 o più bit**. Un'ALU a 1 bit opera su singoli bit in input e output.

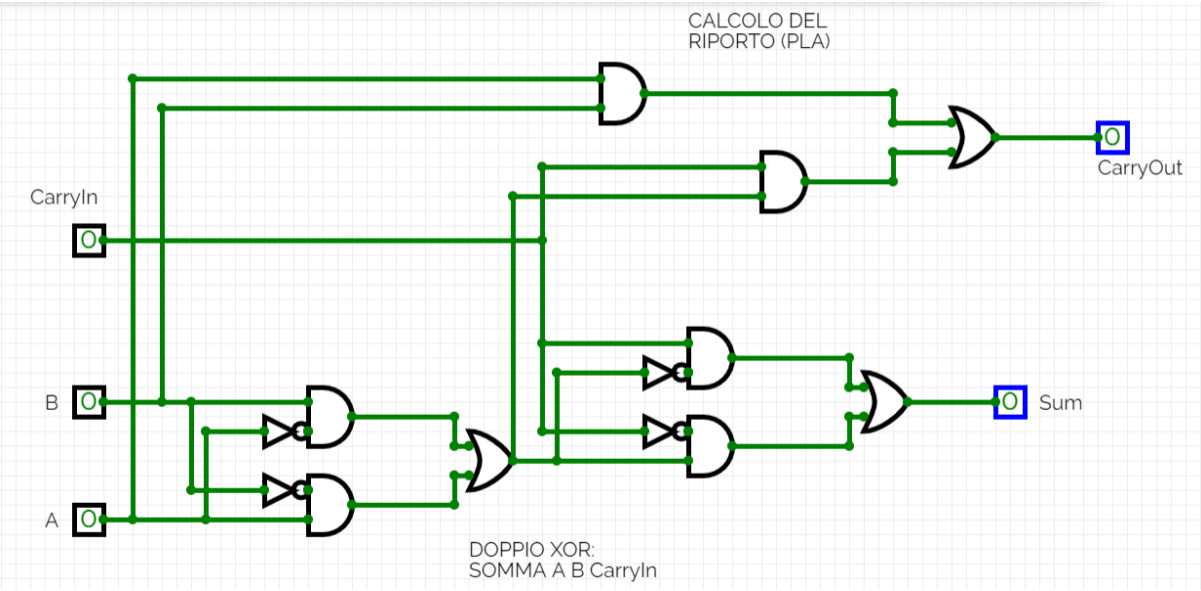
Un'**ALU a 32 bit** si ottiene semplicemente concatenando 32 ALU a 1 bit.

Un'ALU ad 1 bit può implementare l'operazione di **AND e OR bit-a-bit**, utilizzando un multiplexor per scegliere l'operazione da effettuare.



ALU che effettua operazioni di AND e OR

Un'ALU ad 1 bit può implementare la **somma bit-a-bit** mediante un circuito logico detto **semisommatore full-adder**, che riceve in input 2 bit A e B e 1 bit CarryIn e restituisce in output 1 bit Somma e 1 bit CarryOut. Il semisommatore full-adder implementa due XOR per addizionare i tre input, unito ad un circuito PLA utile al calcolo del riporto CarryOut. In buona sostanza, se A e B sono entrambi a 1 o se la loro somma ed il CarryIn sono entrambi a 1, allora si ha CarryOut a 1.



Semisommatore full-adder, spesso identificato da un +

Per ottenere una **sottrazione**, è sufficiente inserire un multiplexor che selezioni, prima di entrare nel semisommatore full-adder, l'utilizzo di B (oppure A) o di NOT B (oppure NOT A).

Un'ALU può eseguire, a scelta, diverse operazioni. Per la selezione si utilizza un **multiplexor**.

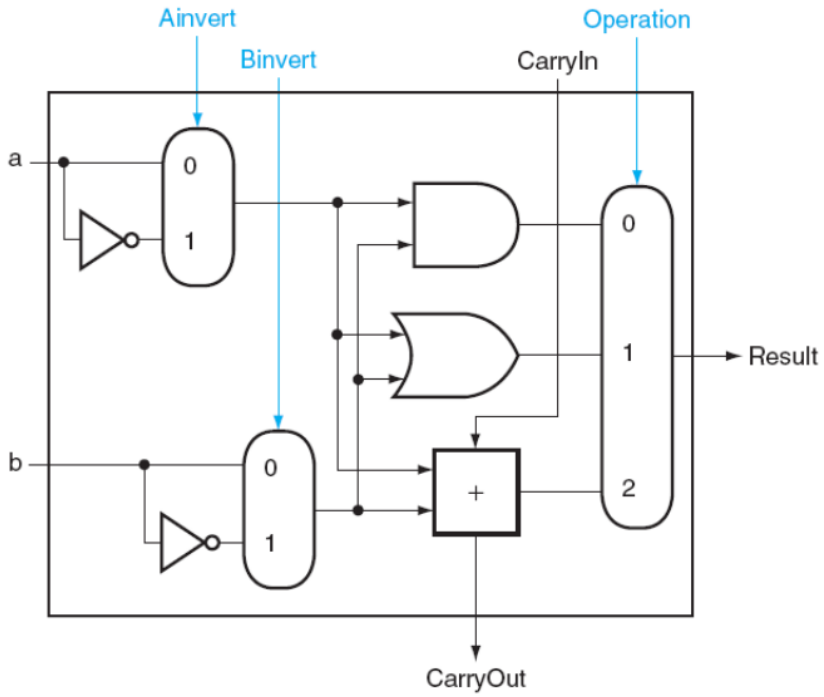
⚠️ ATTENZIONE! Nelle **sottrazioni** il CarryIn dell'ALU meno significativa (ALU_0) è uguale a 1!

Concatenando più ALU, è possibile sommare e sottrarre numeri sempre maggiori: in questi casi il CarryOut del bit meno significativo diviene il CarryIn del bit più significativo. Questo meccanismo è detto **ripple carry**.

Per ottenere **NAND e NOR**, è necessario applicare le leggi di DeMorgan:

- **NOR:** $\neg(A + B) \Leftrightarrow \neg A * \neg B$
- **NAND:** $\neg(A * B) \Leftrightarrow \neg A + \neg B$

Si utilizzano i multiplexor necessari a selezionare il segno di A e B, dopodiché si applica la proprietà corrispondente a NAND o NOR.



ALU che esegue operazioni di AND, OR, NAND, NOR, somma e differenza bit-a-bit

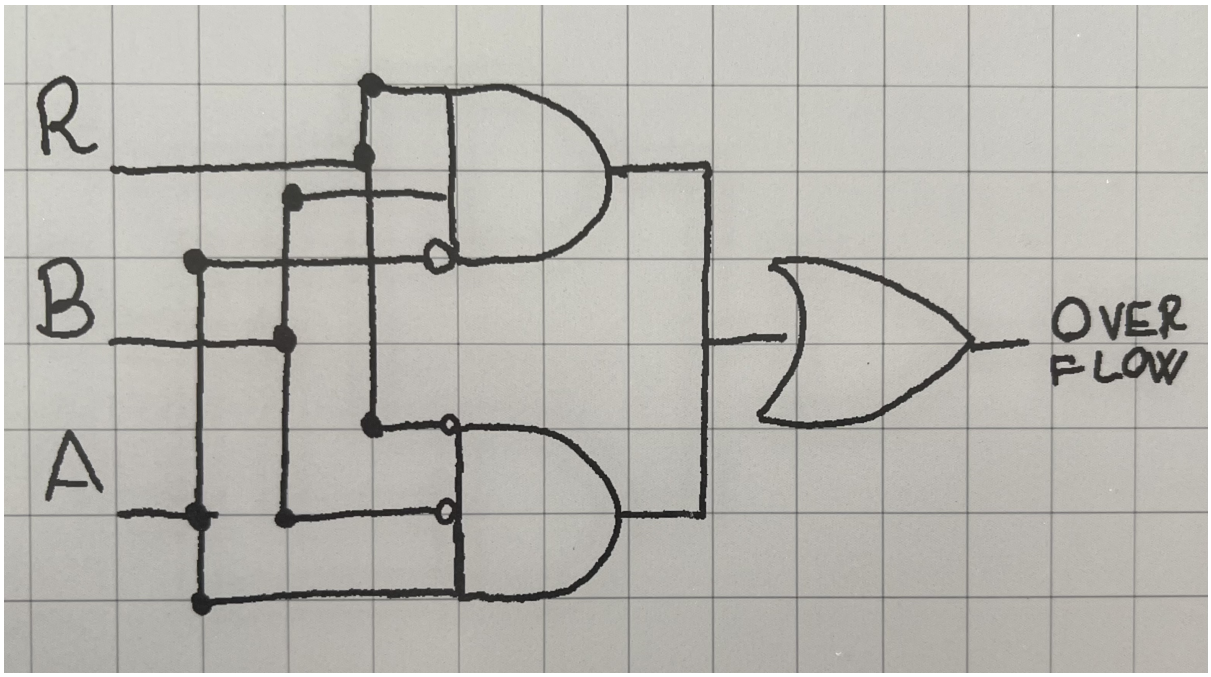
Per effettuare **confronti fra numeri**, si utilizzano ALU a 32 bit secondo i metodi **SLT** (Set on Less Then) o **BEQ** (Brench on EQual).

- **SLT**: è utilizzato per stabilire quale, fra due numeri, è **maggiore** (o **minore**). Tutti i bit assumono valore 0, eccetto l'LSB (ALU_0): esso varrà **1** se $a < b$, **0** se $a \geq b$. Nello specifico, $a < b \; se \; a - b < 0$. In sostanza, il risultato della sottrazione del MSB (ovvero il segno dell'operazione) viene trasmesso dalla

ALU_31 (output *Set*) alla ALU_0 (input *Less*), determinando così il risultato dell'operazione di confronto. Nelle altre ALU, l'input e l'output *Less* saranno uguali a zero.

- **BEQ**: è utilizzato per stabilire se due numeri sono **uguali**. Se due numeri sono uguali, la loro sottrazione bit-a-bit darà come risultato **0**. Dunque, è sufficiente una porta **NOR** che abbia in input i risultati bit-a-bit della sottrazione svolta da ognuna delle 32 ALU. Se tutti i bit sono uguali a zero, i due numeri sono uguali e il NOR avrà valore 1.

La *Most Significant ALU* permette anche di verificare i casi di **overflow**: in generale, se $a - b > 0$ e il risultato della ALU_31 è **1** o se $a - b < 0$ e il risultato della ALU_31 è **0**, si ha overflow.



Circuito di controllo overflow