



ALU + Bistabili

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento Patterson: sezioni B.7 & B.8.



Sommario

ALU: Comparazione, Overflow, Test di uguaglianza

Bistabili

Latch SC



Sottrazione: ALU a 32 bit



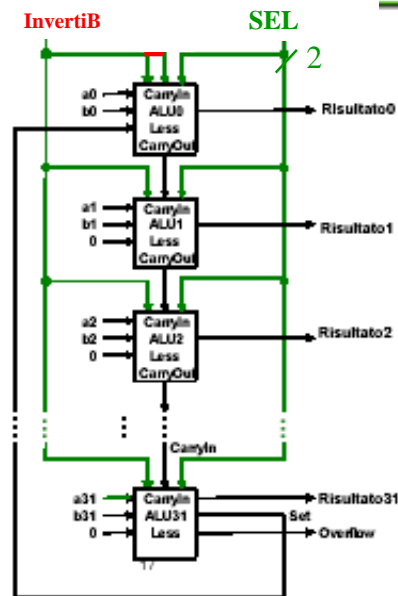
$r_{in}(0) = \text{InvertiB} = 1$
se sottrazione

- AND
- OR
- SOMMA
- SOTTRAZIONE

InvertiB e r_0 sono lo stesso segnale, si può ancora ottimizzare.

$r_{in}(0)$ entra solo in ALU_0

InvertiB entra in tutte le ALU_i



Confronto

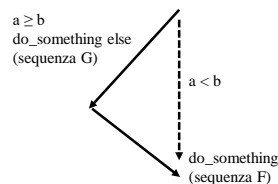


Fondamentale per **dirigere il flusso** di esecuzione (test, cicli,...). Costruito «if»:

if (**Condizione**) then
 salto alle istruzioni nella sequenza F
else
 Istruzioni in sequenza, G

Condizione può essere:

- $A = B$
- $A \neq B$
- $A < B$
- $A > B$
- $A \geq B$
- $A \leq B$



In MIPS solamente le condizioni $A = B$ e $A \neq B$ vengono utilizzate **direttamente** per dirigere il flusso. Corrispondono alle istruzioni di salto condizionato: beq e bne.

Come vengono trattate le condizioni $A < B$ e $A > B$?



Confronto di minoranza



Il **confronto di minoranza** produce una variabile che assume il valore: FALSO o VERO, $\text{flag} = \{0,1\}$

```
if (a less_than b) then
    s = flag = TRUE = 1;
else
    s = flag = FALSE = 0;
```

```
if (a < b) then
    flag = 1;
else
    flag = 0;
```

```
if (b ≥ a) then
    flag = 1;
else
    flag = 0;
```

Occorre che la ALU sia attrezzata per eseguire questo confronto di minoranza ($a < b$). Questa sarà una quinta operazione possibile: AND, OR, Somma, Sottrazione, SLT (istruzione: *Set on Less Than*).



Utilizzo del confronto di minoranza



Linguaggio ad alto livello

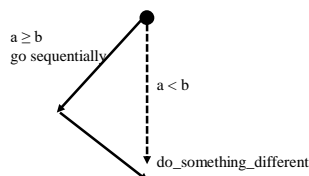
```
if (a < b) then
    do_something_different;
end;
```

```
if (a < b) then
    flag = 1;
else
    flag = 0;
```

```
if (flag != 0) // (flag = 1) => (a < b) => salta
    go sequentially
end;
```

dopo:

```
do_something_different
```



Assembler

```
slt regdest, reg1, reg2
# regdest contiene il flag
```

```
bne regdest, 0, dopo
go sequentially
```

dopo:

```
do_something_different
```



Come sviluppare la comparazione: slt - I?



$a < b \iff a - b < 0$

if (**a < b**) then
 flag = 1
 else
 flag = 0

if (**((a - b) < 0)**) then
 flag = 1
 else
 flag = 0

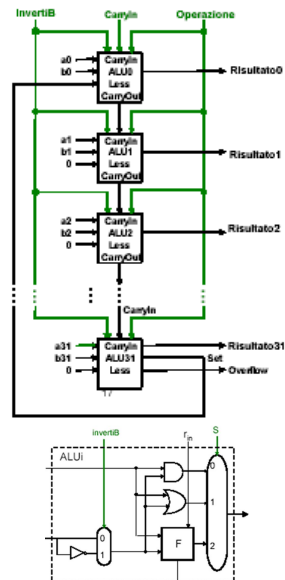
Si controlla che il primo bit del risultato della sottrazione
(bit di segno) sia = 1.

Esempio: consideriamo numeri su 4 bit: 3 per l'ampiezza
e 1 per il segno.
a - b = 3 - 4; su 4 bit

3 -> 0011
-4 -> 1100 in complemento a 2
-1 -> 1111 in complemento a 2

0 0 1 1 +
1 1 0 0 =
1 1 1 1

MSB = bit di segno



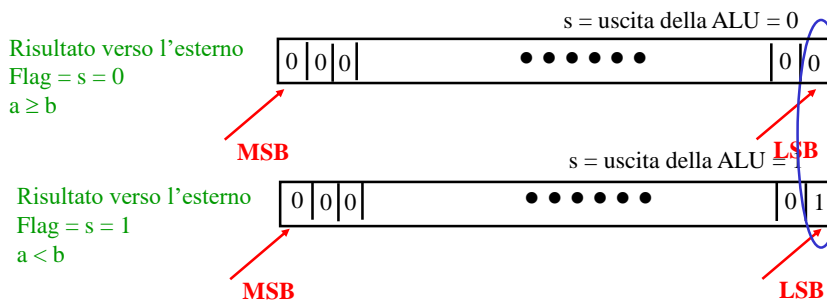
Come sviluppare la comparazione - II?



if ((a - b) < 0) then
 flag = 1
 else
 flag = 0

-> **Flag (bit singolo che segnala un evento)**
assume i valori {Falso, Vero} - {0, 1}

Valori possibili in uscita della ALU: s = flag = {0, 1} ma su 32 bit!!







Comparatore – ALUⁱ ($i \neq 0$)



Prevedo una quarta operazione: SLT

AND (0 - 00)

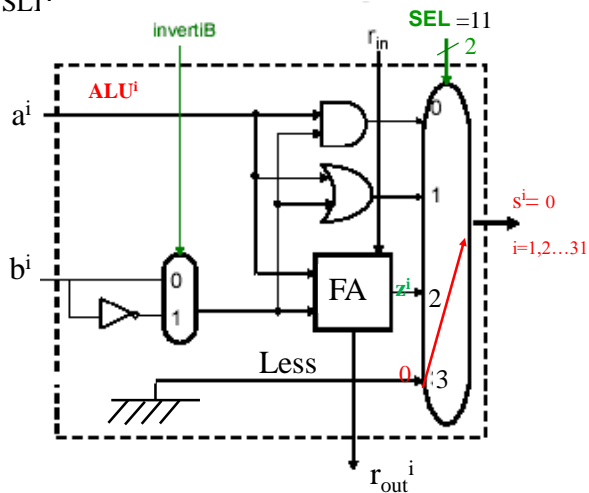
OR (1 - 01)

Somma/Sottrazione (2 - 10)

SLT (3 - 11)

SEL = 11 (ingresso #3 al mux)

InvertiB = 1 (sottrazione)



Less(i) = 0 $i = 1, 2, 3, \dots, 31$ $i \neq 0$



Comparatore – ALU⁰ : ALU³¹



InvertiB = $r_{in}^0 = 1$ (sottrazione)

SEL = 11 (ingresso #3 al mux)

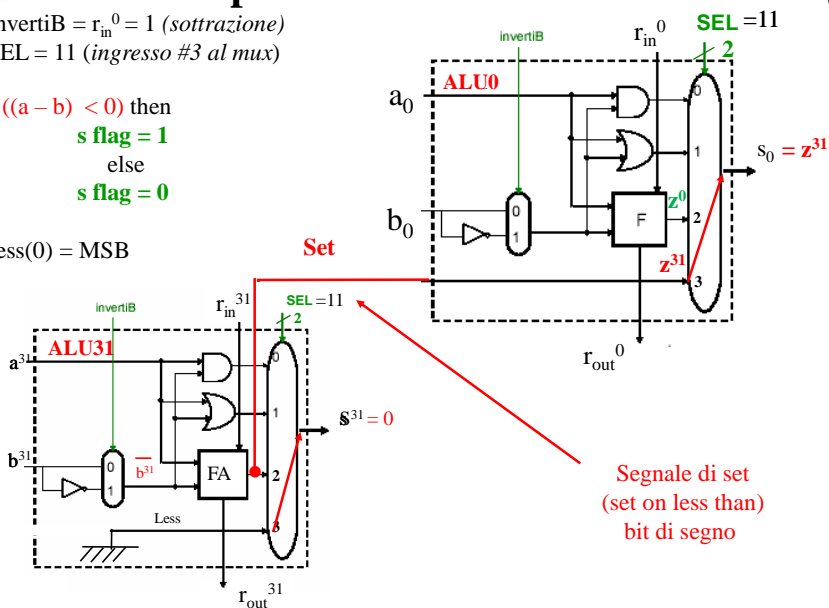
if $((a - b) < 0)$ then

s flag = 1

else

s flag = 0

Less(0) = MSB





Comparatore – ALU⁰ : ALU³¹



InvertiB = $r_{in}^0 = 1$ (sottrazione)

SEL = 11 (ingresso #3 al mux)

Esempio:

$a = 5 - 0101_2$

$b = 7 - 1011_2$

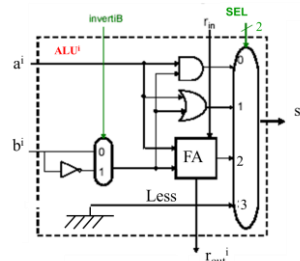
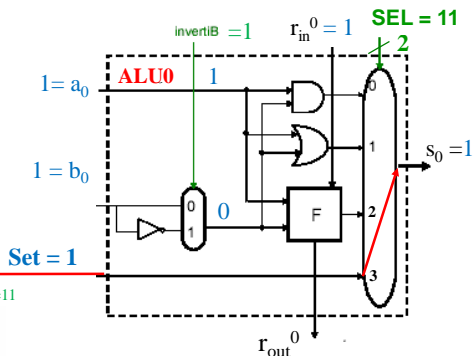
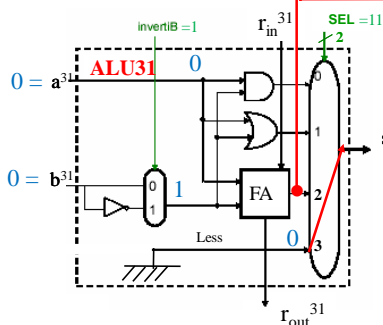
$a - b = -2 - 111...110_2$

if $((a - b) < 0)$ then

s flag = 1

else

s flag = 0



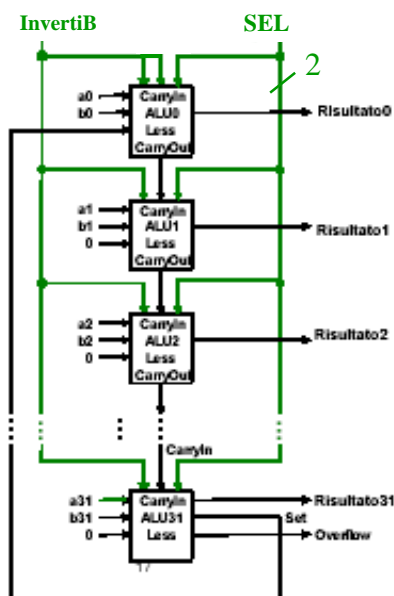
Comparazione - ALU completa a 32 bit



- AND
- OR
- Somma/Sottrazione
- Comparazione ($a < b$)

Come implemento ($a > b$)?

Come implemento ($a \geq b$)?





Overflow decimale



$a + b = c$ - codifica su 2 cifre,

$a = 12, b = 83 \Rightarrow$ Not Overflow.

```
012+
083=
-----
095
```

$a = 19, b = 83 \Rightarrow$ Overflow

```
019+
083=
-----
102
```

Si può avere overflow con la differenza?



Overflow binario



$a + b = c$ - codifica su 4 cifre binarie, di cui 1 per il segno.

No overflow

0000

```
0010+  2+
0011=  3=
-----
0101    5
```

1110

```
1010+ -6+
1111= -1=
-----
1001   -7
```

Overflow

0110

```
0010+  2+
0111=  7=
-----
1001   -7 in complemento a 2??
```

0110

```
1010+ -6+
1011= -5=
-----
0101   +5 in complemento a 2??
```

Quindi si ha overflow nella somma quando $a + b = s$:

- $a \geq 0, b \geq 0$ MSB di a e $b = 0$, MSB di $s = 1$.
- $a < 0, b < 0$ MSB di a e $b = 1$, MSB di $s = 0$.

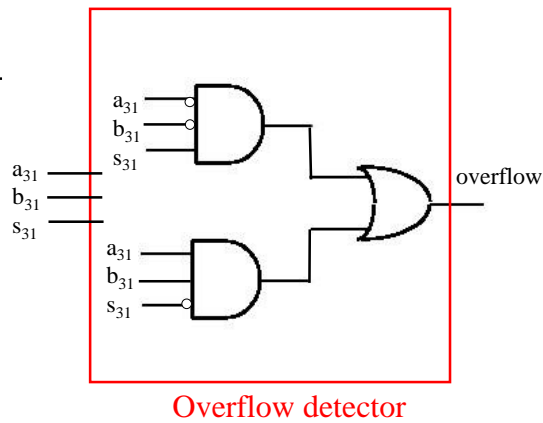


Circuito di riconoscimento dell'overflow



Lavora sui MSB

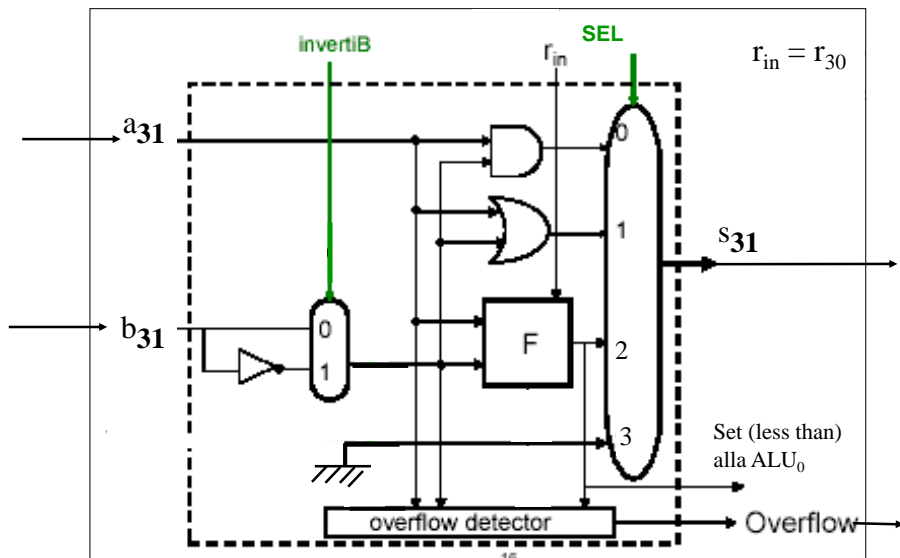
a_{31}	b_{31}	s_{31}	overflow
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



$$\text{Overflow} = !a_{31}!b_{31}s_{31} + a_{31}b_{31}s_{31}$$



ALU₃₁





Uguaglianza - prima implementazione



beq reg1, reg2 label

iff (reg1 == reg2),
salta

then

prosegui in sequenza

iff ((rs -rt) == 0),
salta.

Occorre quindi:

- Impostare una differenza (reg1 - reg2).
- Controllare che la differenza sia = 0 \rightarrow (reg1 - reg2) = 0 su 32 bit \rightarrow 0000.....00000
- Effettuare l'OR logico di tutti i bit in uscita dai sommatatori $\{z_i\}$.
- L'uscita dell'OR logico = 0 \rightarrow i due numeri sono uguali \rightarrow NOR

Input: a, b su 32 bit

Output è un Flag di zero: $\{0,1\}$. Zero = NOR(z_0, z_1, \dots, z_{N-1}) a 32 ingressi.



Uguaglianza – seconda implementazione



beq reg1, reg2 label

iff (reg1 == reg2), salta.

$$C_k = \overline{a_k \oplus b_k}$$

A_0	B_0	C_0	A_1	B_1	C_1
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	1	1	1

...

$$C = C_0 C_1 \dots C_{n-1}$$

Per la ALU:

- Input: a, b su 32 bit
- Output: $\{0,1\}$ -> segnale di zero.

Per 32 bit occorrono le seguenti porte a 2 ingressi:

32 XOR (già contenute nella ALU)

AND a 32 ingressi

Soluzione per comparatori

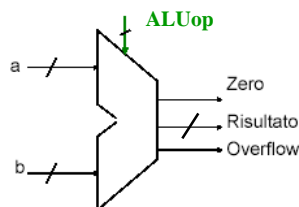


ALU a 32 bit: struttura finale

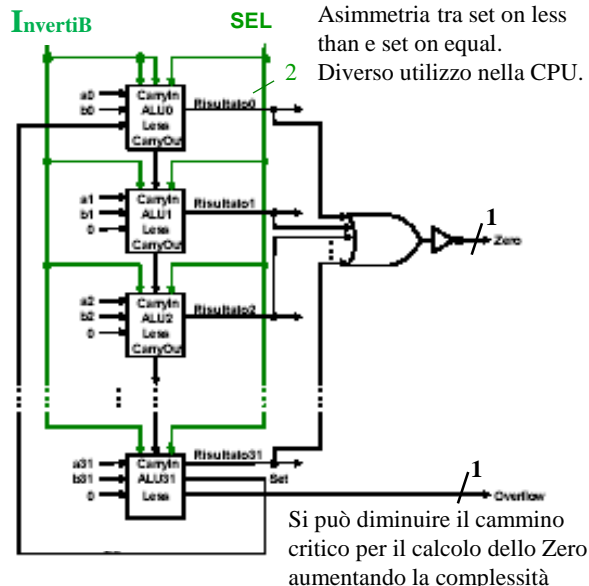


Operazioni possibili:

- AND
- OR
- Somma / Sottrazione
- Comparazione
- Test di uguaglianza



Sono evidenziate
solamente le variabili
visibili all'esterno.



Si può diminuire il cammino critico per il calcolo dello Zero aumentando la complessità



Approcci tecnologici alla ALU.



Tre approcci tecnologici alla costruzione di una ALU (e di una CPU):

- **Approccio strutturato.** Analizzato in questa lezione.
- **Approccio hardware programmabile (e.g. PAL).** Ad ogni operazione corrisponde un circuito combinatorio specifico.
- **Approccio ROM.** E' un approccio esaustivo (tabellare). Per ogni funzione, per ogni ingresso viene memorizzata l'uscita. E' utilizzabili per funzioni molto particolari (ad esempio di una variabile). Non molto utilizzato.
- **Approccio firmware (firm = stabile), o microprogrammato.** Si dispone di circuiti specifici solamente per alcune operazioni elementari (tipicamente addizione e sottrazione). Le operazioni più complesse vengono sintetizzate a partire dall'algoritmo che le implementa.



Approccio ROM alla ALU

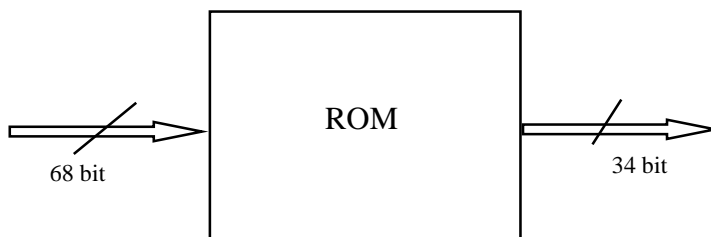
Input:

- A n bit
- B n bit
- SEL k bit
- InvertiB: 1 bit
- Totale:** $2*n + k + 1$ bit

Output:

- s n bit
- zero 1 bit
- overflow 1 bit
- Totale:** $n + 2$ bit

Per dati su 32 bit, 5 operazioni ($k = 4$):



Capacità della ROM: $2^{68} = 2.95.. \times 10^{20}$ parole di 34 bit!

A.A. 202 (Capacità della ROM per dati su 4 bit, 5 operazioni: $2^{12} = 4,098$ parole di 6 bit) se.di.unimi.it/



Sommario

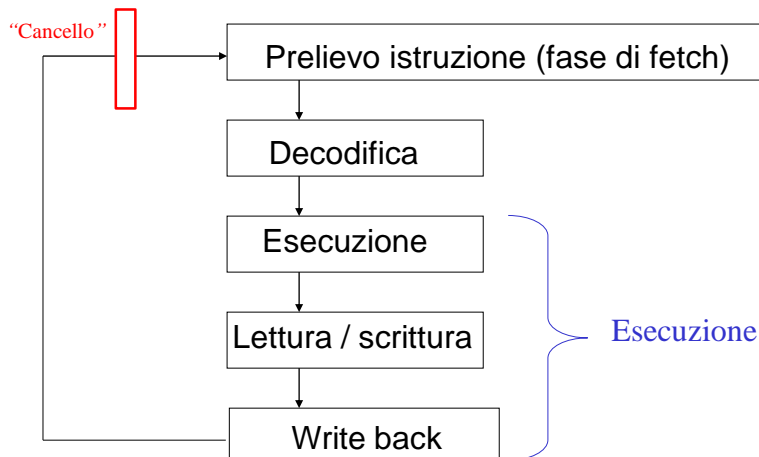
ALU: Comparazione, Overflow, Test di uguaglianza

Bistabili

Latch SC



Dispositivi di sincronizzazione



Circuiti asincroni



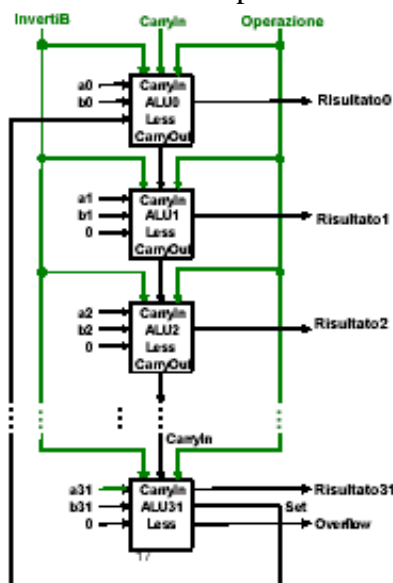
Architettura logica asincrona:

L'elaborazione e propagazione dei segnali avviene in modo incontrollato, secondo le velocità di propagazione dei circuiti.

Progetto asincrono: Devo progettare il circuito in modo che nessun transitorio/cammino critico causi problemi → analisi di tutti i transitori critici possibili.

Improprio per circuiti con feed-back (retroazione).

Esempio: ALU





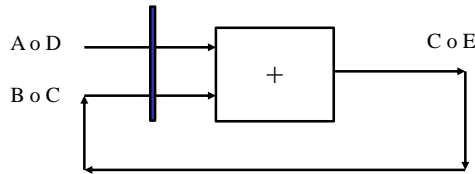
Circuiti retro-azionati e sincronizzazione



Esempio:

$$C = A + B$$

$$E = D + C$$



I problemi dei circuiti retroazionati

- Quando posso calcolare E con lo stesso sommatore?
- Quando devo fare passare A e B oppure A e C?

Soluzione

“Cancello” davanti all’ingresso del sommatore prima della seconda somma.

Occorre una sincronizzazione dell’attività del sommatore.



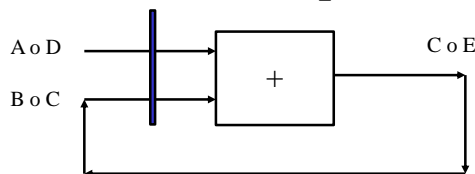
Sincronizzazione temporizzata



Esempio:

$$C = A + B$$

$$E = D + C$$



I problemi dei circuiti retroazionati

- Quando posso calcolare E con lo stesso sommatore?
- Quando devo fare passare A e B oppure A e C?

Occorre una sincronizzazione dell’attività del sommatore.

Dobbiamo essere ragionevolmente sicuri che il risultato sia stato calcolato ed utilizzato.

Soluzione

- “Cancello” davanti all’ingresso del sommatore prima della seconda somma.
- Il cancello viene **temporizzato**: si apre ogni intervallo di tempo prefissato.



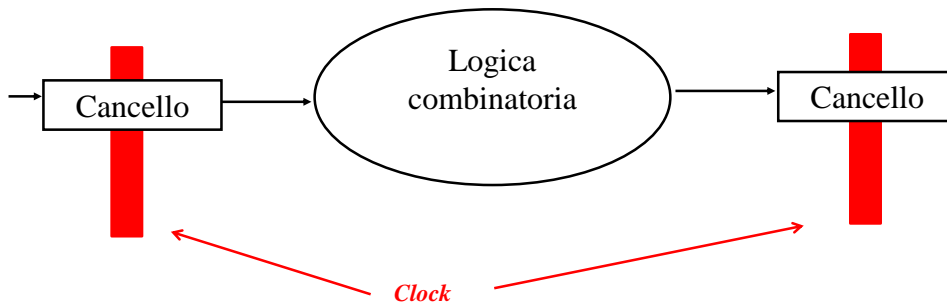
Circuiti sincroni

- **Architettura logica sincrona:**

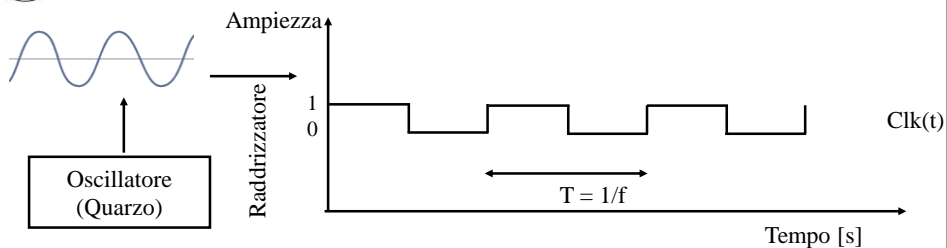
Le fasi di elaborazione sono scandite da un orologio comune a tutto il circuito (**clock**).

- Ad ogni fase di clock, la parte combinatoria del circuito ha tempo di elaborare (i segnali di percorrere il cammino critico) e quindi il circuito ha il tempo di stabilizzarsi (transitori critici). Questo deve avvenire entro il “tick” successivo.
- **Progetto sincrono**: il controllo dei transitori/cammini critici è limitato alla parte di circuito tra due **cancelli** (porte di **sincronizzazione**)

Esempio: CPU



Clock di sistema



Frequenza: numero di cicli/s

Si misura in Hertz, Hz.

Periodo: tempo necessario a completare 1 ciclo

Si misura in secondi, s.

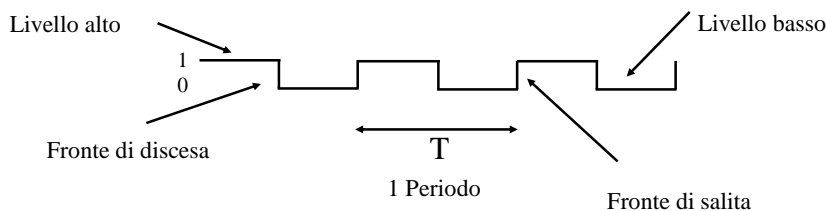
$$T = 1/f$$

$$1\text{ ns} = 1/1\text{ GHz}$$

Tempo di salita e discesa trascurabile rispetto al tempo di commutazione delle porte logiche.



Utilizzo del clock



- **Metodologia sensibile ai livelli:**

Le variazioni di stato possono avvenire per tutto il periodo in cui il clock è al livello alto (basso).

- **Metodologia sensibile ai fronti:**

Le variazioni di stato avvengono solamente in corrispondenza di un fronte di clock. Noi adotteremo questa metodologia.



Architetture sequenziali

• I circuiti combinatori **non hanno memoria e non hanno bisogno di sincronizzazione**. Gli output al tempo t dipendono unicamente dagli input al tempo t che provengono dall'esterno: $y^{t+1} = f(u^{t+1})$

1) Sono necessari circuiti **con memoria**, per consentire comportamenti diversi a seconda della situazione dell'architettura. Nella memoria viene memorizzato lo **stato** del sistema che riassume la storia passata (e.g. Register file, memoria, PC sono elementi di stato di una CPU). **Sono alla base delle architetture retro-azionate.**

• Sono necessari dispositivi di **sincronizzazione** (cancelli) per eseguire operazioni **sequenzialmente** e il risultato di un'operazione è l'input dell'operazione successiva (architetture con feed-back (e.g. CPU, Distributore di bibite)).

Memoria e sincronizzazione sono assolve nei circuiti digitali dai **bistabili**.



Bistabili: latch e flip-flop



Elemento base della memoria è il **bistabile**: dispositivo in grado di mantenere *indefinitamente* uno dei due possibili valori di output: (0 o 1).

Il suo valore di uscita coincide con lo stato. L'uscita al tempo t , dipende:

- dallo stato (uscita) al tempo $t-1$
- dal valore presente agli input.

Tipi di bistabili:

- Bistabili non temporizzati (asincroni, **latch asincroni**) / temporizzati (sincroni).
- Bistabili sincroni che commutano sul livello del clock (**latch**) o sul fronte (**flip-flop**).



Sommario



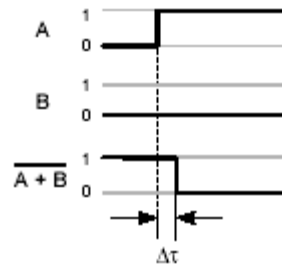
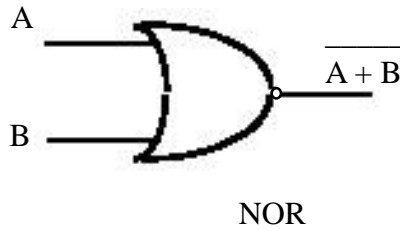
ALU: Comparazione, Overflow, Test di uguaglianza

Bistabili

Latch SC



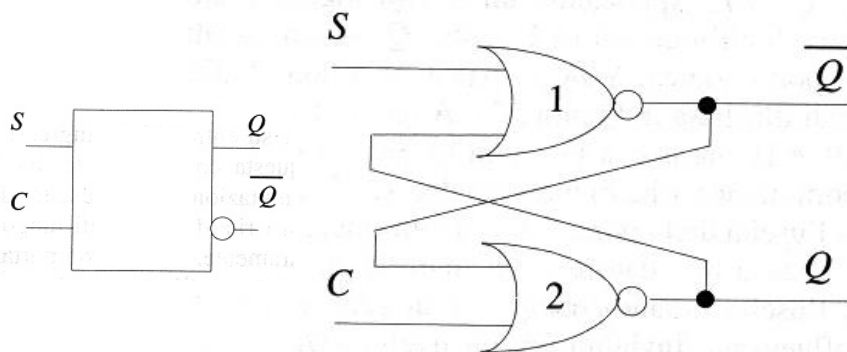
Principio di funzionamento



Il ritardo, $\Delta\tau$, introdotto tra la commutazione dell'input e la commutazione dell'output è alla base del funzionamento di un bistabile.



Latch asincrono SC (o SR)

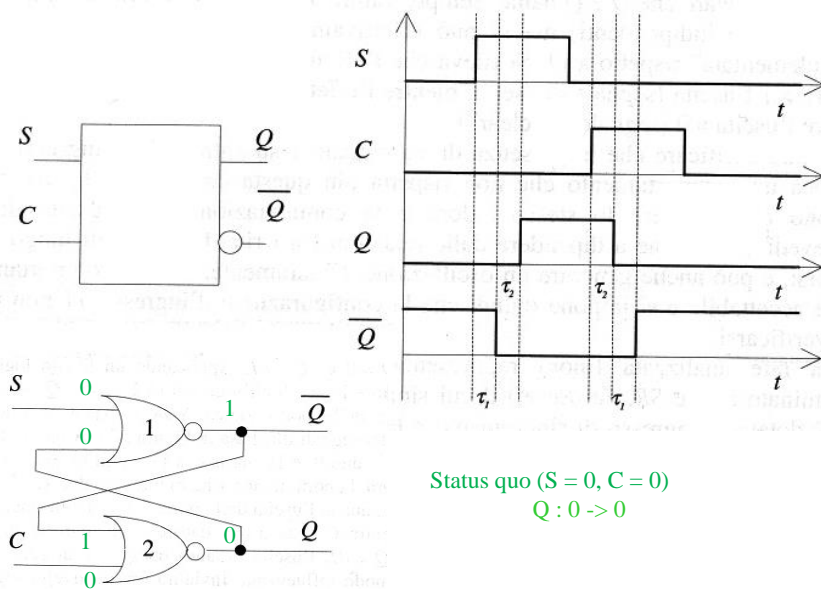


Una coppia di porte NOR retro-azionate può memorizzare un bit.

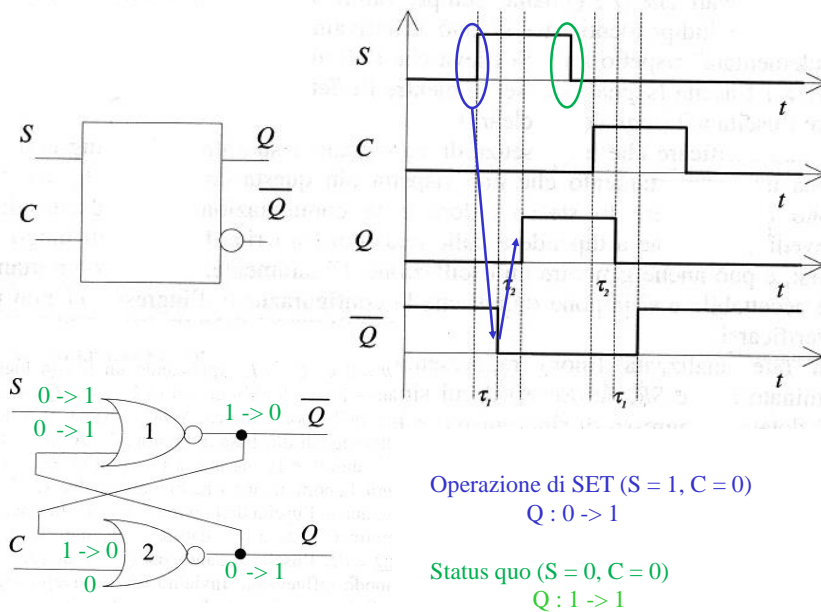
Cioè se in ingresso arriva $S = C = 0$, l'uscita non commuta e rimane al valore attuale ($Q = 0$; oppure $Q = 1$)



Funzionamento del circuito SC – Status quo

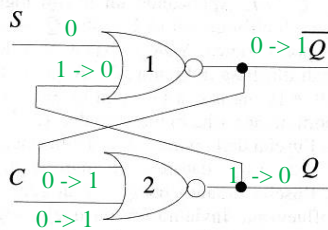
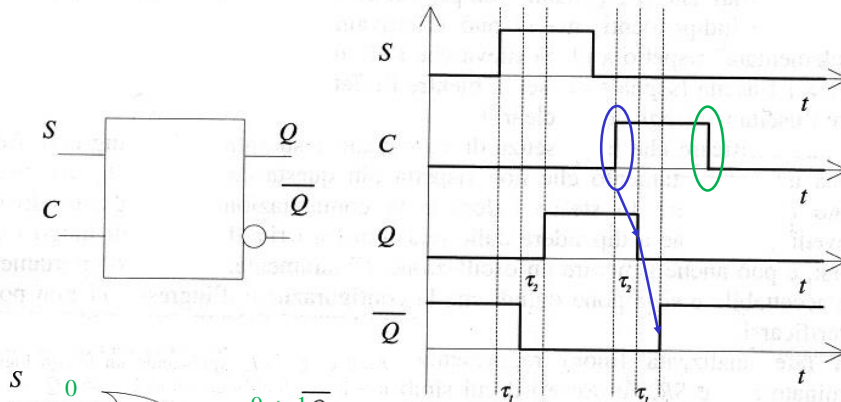


Funzionamento del circuito SC - set





Funzionamento del circuito SC - clear



Operazione di CLEAR ($S = 0, C = 1$)

$Q : 0 \rightarrow 1$

Status quo ($S = 0, C = 0$)

$Q : 0 \rightarrow 0$



Osservazioni



Il circuito è molto semplice, costituito da 2 NOR retroazionati

Accetta in ingresso le configurazioni

SC

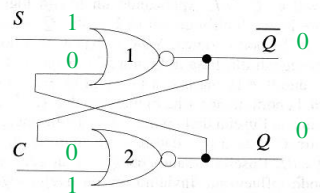
00 (status quo)

01 (clear)

10 (set)

Non accetta la configurazione 11

\rightarrow in uscita avessimo 00 \rightarrow errore



Come posso evitare una condizione di errore in uscita per ingresso $S=C=1$?



Tabella della verità del latch



Se considero Q (**lo stato**) e S e C ingressi, ottengo la **tabella della verità di Q***:

$$Q^* = f(Q, S, C)$$

Q è l'uscita del latch: **stato presente**, Q_t
Q* è il valore dell'uscita al tempo successivo: **stato prossimo**, Q_{t+1}

S	C	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X



Tabella delle transizioni



$$Q^* = f(Q, S, C)$$

Variabile di Stato
(interna)

Variabili di Ingresso
(esterne)

S	C	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

		ingressi esterni			
ingressi interni - stato	Q	SC = 00	SC = 01	SC = 10	SC = 11
	0	0	0	1	X
1	1	1	0	1	X
		No change (Q* = Q) Memory	Clear Reset Write 0	Set Write 1	



Tabella della verità del latch ($X = \{0,0\}$)

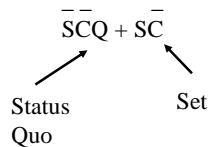


Se considero Q (**lo stato**) e S e C ingressi, ottengo la **tabella della verità di Q^*** :

$$Q^* = f(Q, S, C)$$

Q è l'uscita del latch: **stato presente**, Q_t
 Q^* è il valore dell'uscita al tempo successivo: **stato prossimo**, Q_{t+1}

$$Q^* = \bar{S}\bar{C}Q + S\bar{C}\bar{Q} + S\bar{C}Q = \bar{S}\bar{C}Q + S\bar{C}(\bar{Q} + Q) =$$



If ($S=C=1$) then
 $Q^* = 0$ (non dipende da Q)

S	C	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X = 0
1	1	1	X = 0



Confronto tra i 2 circuiti

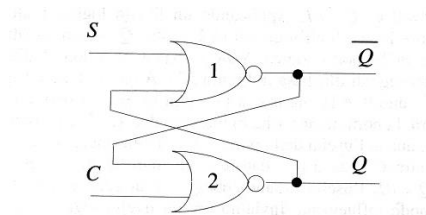
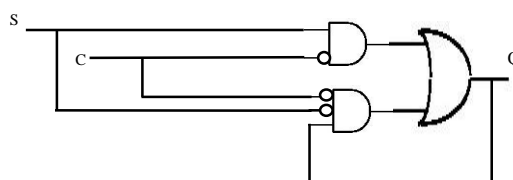




Tabella della verità del latch (X= {1,1})



Impostando X = 1 1, si ottiene:

$$Q^* = \bar{S}\bar{C}Q + \bar{S}\bar{C}\bar{Q} + \bar{S}CQ + S\bar{C}\bar{Q} + SCQ =$$

$$= \bar{S}\bar{C}Q + \bar{S}\bar{C}(\bar{Q}+Q) + SC(\bar{Q}+Q) =$$

$$= \bar{S}\bar{C}Q + S(\bar{C}+C) =$$

$$= \bar{C}Q + S \quad (\text{per assorbimento di } \bar{S})$$

↖ Status quo ↘ Set

(SC = 11 -> Q* = 1 -> Set)

S	C	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X=1
1	1	1	X=1



Tabella della verità del latch (X= {0,1})



Impostando X = {0 1}, si ottiene:

$$Q^* = \bar{S}\bar{C}Q + S\bar{C}\bar{Q} + S\bar{C}Q + S C Q =$$

$$= \bar{S}\bar{C}Q + S C Q + S\bar{C}(Q + \bar{Q}) =$$

$$= (\bar{S} \oplus C) Q + S\bar{C}$$

↖ Status quo ↘ Set

(SC = 11 -> Q* = Q -> Status quo)

S	C	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X=0
1	1	1	X=1



Tabella delle eccitazioni

Q	Q*	S	C
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Data la transizione $Q \rightarrow Q^*$, qual'è la coppia di valori di ingresso che la determina?

$$(Q, Q^*) = f(S, C)$$



Sommario

ALU: Comparazione, Overflow, Test di uguaglianza

Bistabili

Latch SC