



ALU + Bistabili

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione

borgnese@di.unimi.it

Università degli Studi di Milano

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



Sommario

ALU: Comparazione, Overflow, Test di uguaglianza

Bistabili

Latch SC



Sottrazione: ALU a 32 bit



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

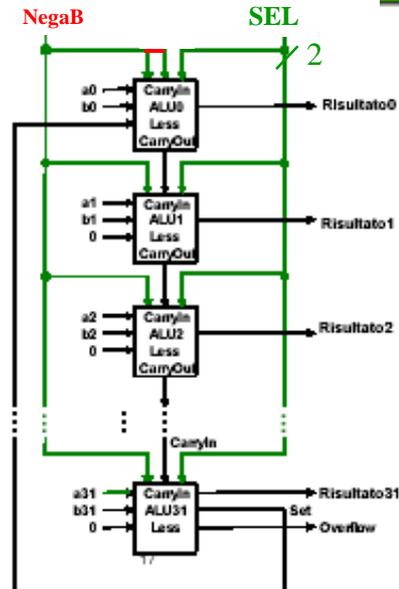
- AND
- OR
- SOMMA
- SOTTRAZIONE

From_UC	SEL	r_0	InvertiB
And	And	0	0
Or	Or	0	0
Somma	Add	0	0
Sottr.	Add	1	1

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



A.A.

3/37

<http://borghese.di.unimi.it/>



Confronto



Fondamentale per **dirigere il flusso** di esecuzione (test, cicli...)

```

if Condizione then
    Sequenza F
else
    Sequenza G
  
```

Condizione può essere:

- A = B
- A ≠ B
- A < B
- A > B

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

Come vengono trattate le condizinoi A < B e A > B?

A.A. 2013-2014

4/41

<http://borghese.di.unimi.it/>



Confronto di minoranza



Il confronto di minoranza produce una variabile che assume il valore {0,1}

```
if a less_than b then
  s = 1
```

```
if a < b then          z = a - b
  s = 1
else
  s = 0
```

```
if z < 0 then
  s = 1
else
  s = 0
```



Come sviluppare la comparazione - I?



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

```
if differenza(a,b) < 0 then
  s = 1
else
  s = 0
```

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

Esempio: $s = 3 - 4$; su 3 bit

3 -> 011
-4 -> 100 in complemento a 2
-1 -> 111 in complemento a 2

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

 MSB = bit di segno



Come sviluppare la comparazione - II?

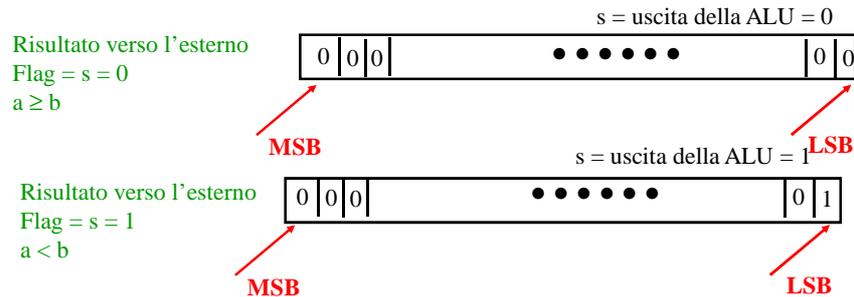


```

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

```

Viene impostato un flag = 1 se $a < b$.
 Valori possibili in uscita della ALU: {0, 1}



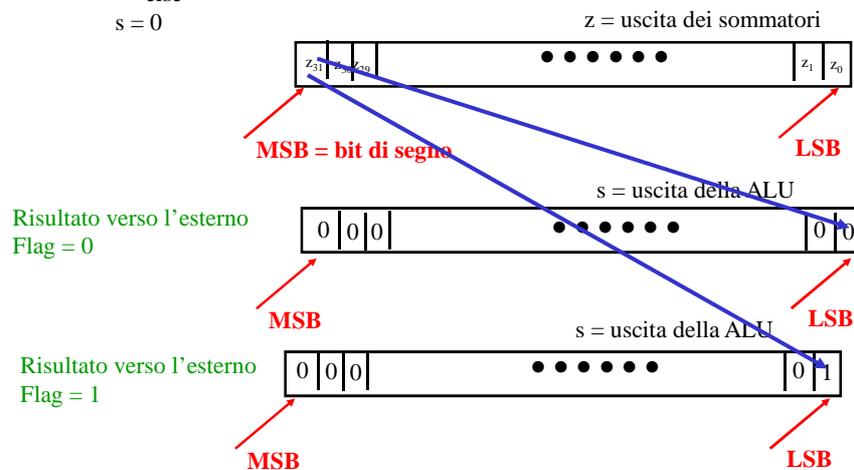
Come sviluppare la comparazione (riassunto)?

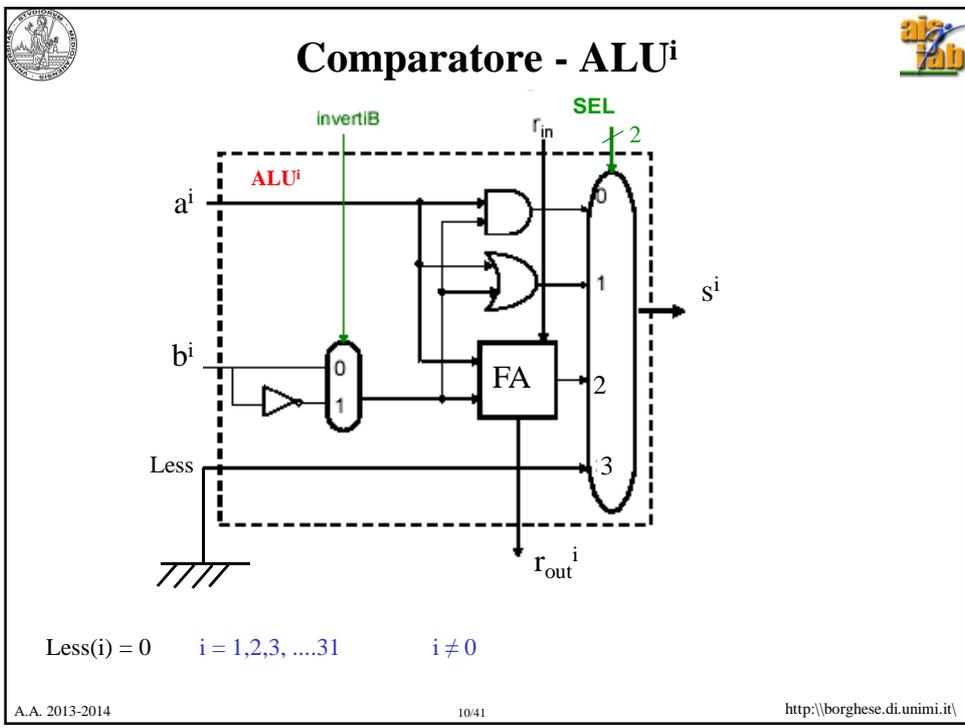
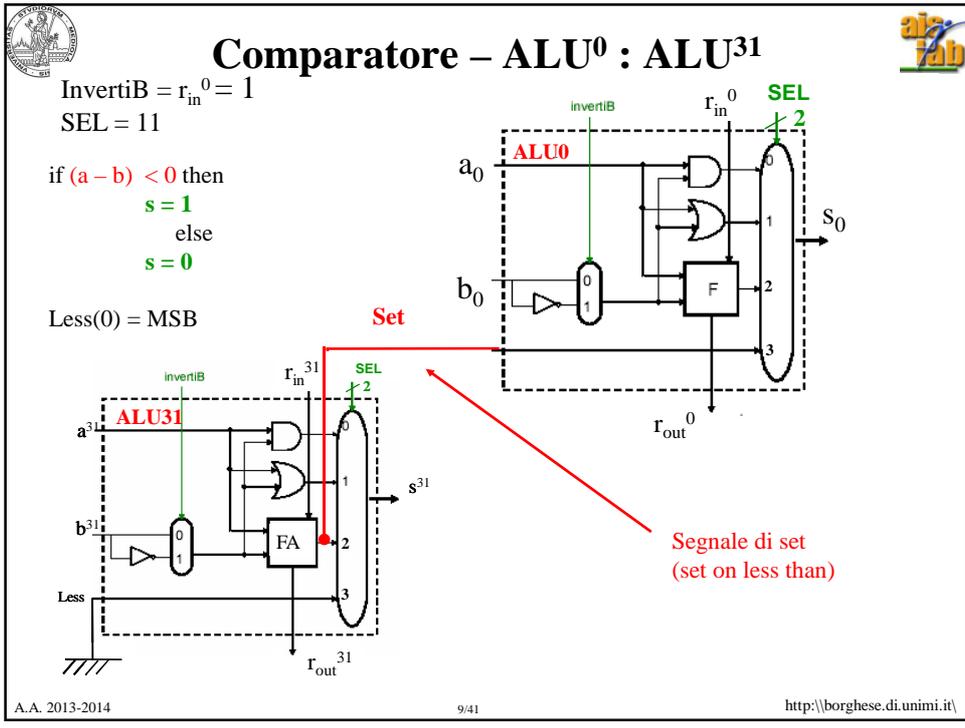


```

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

```



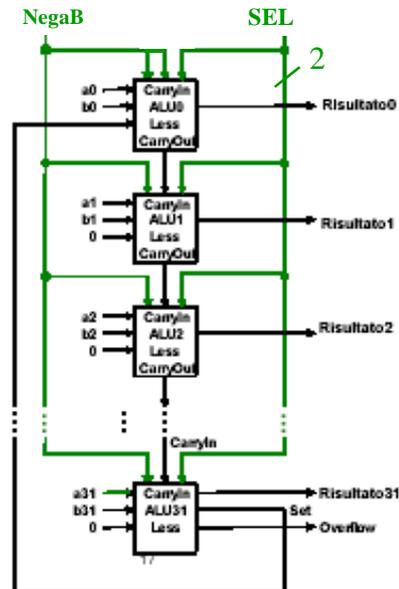




Comparazione - ALU completa a 32 bit



- AND
- OR
- SOMMA
- SOTTRAZIONE
- COMPARAZIONE
 - (a < b)
 - (a > b)



Overflow decimale



a + b = c - codifica su 2 cifre,

a = 12, b = 83 => Not Overflow.

```

012+
083=
----
095

```

a = 19, b = 83 => 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

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

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

Overflow

0010+	2+
0111=	7=
----	----
1001	-7??

1010+	-6+
1011=	-5=
----	----
0101	+5??

Quindi si ha overflow nella somma quando:

$a + b = s, \quad a > 0, b > 0 \quad \text{MSB di } a \text{ e } b = 0, \text{ MSB di } s = 1.$

$a + b = s, \quad a < 0, b < 0 \quad \text{MSB di } a \text{ e } b = 1, \text{ MSB di } s = 0.$

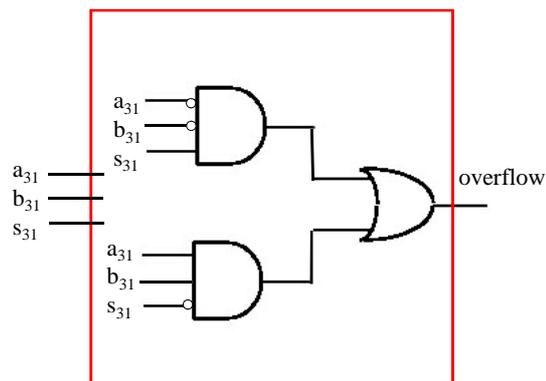


Circuito di riconoscimento dell'overflow



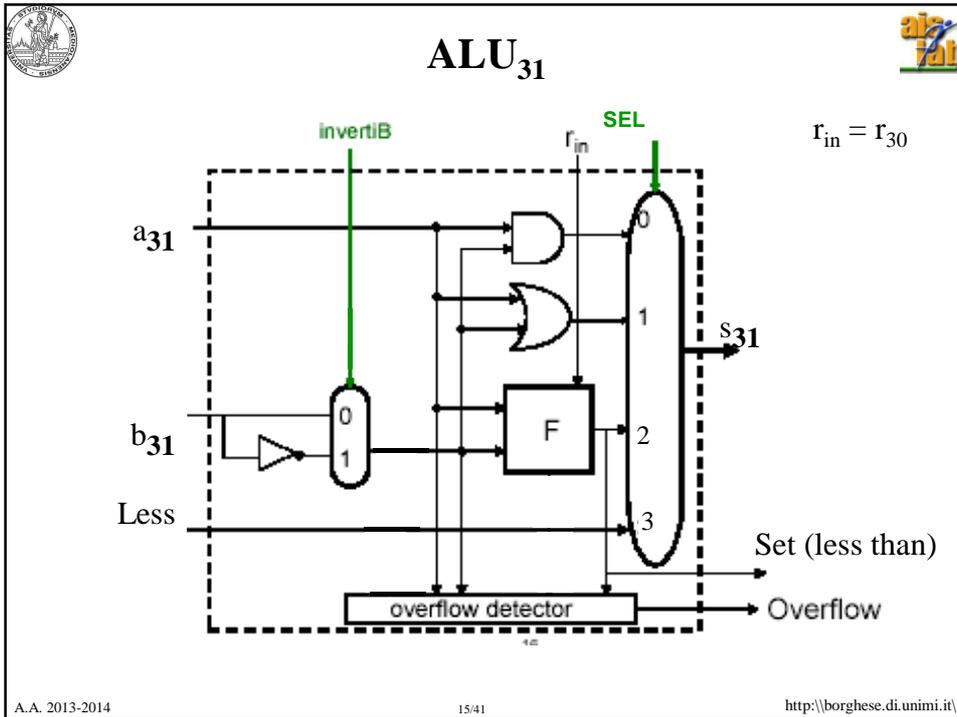
Lavora sui MSB

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



Overflow detector

$$\text{Overflow} = !a!b s + a b !s$$



Uguaglianza – prima implementazione

beq rs, rt label iff (rs – rt) = 0 , salta.

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

...

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

$C_k = a_k \oplus b_k$

Input: a, b su 32 bit
Output: {0,1}.

Per 32 bit occorrono le seguenti porte a 2 ingressi:
32 XOR (già contenute nella ALU)
31 AND

Si può fare di meglio sfruttando il sommatore già presente nella ALU

A.A. 2013-2014

16/41

<http://borghese.di.unimi.it/>



Uguaglianza - seconda implementazione



beq rs, rt label iff $(rs - rt) = 0$, salta.

Occorre quindi:

- Impostare una differenza.
- Effettuare l'OR logico di tutti i bit somma.
- L'uscita dell'OR logico = 0 i due numeri sono uguali.

Input: a, b su 32 bit

Output: {0,1}. In questo caso viene dedicata una linea di uscita.

$$\text{Zero} = \text{NOR}(FA_0, FA_1, FA_{N-1})$$

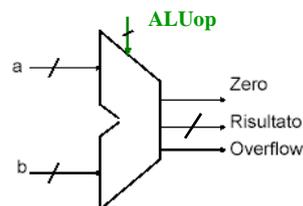


ALU a 32 bit: struttura finale

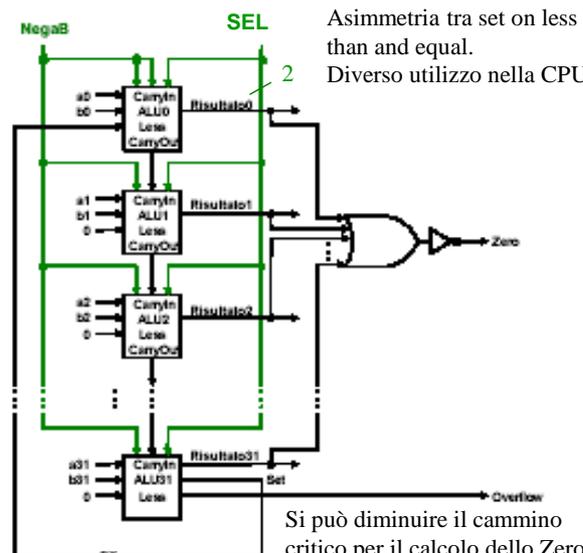


Operazioni possibili:

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



Sono evidenziate
solamente le variabili
visibili all'esterno.



Asimmetria tra set on less than and equal.
Diverso utilizzo nella CPU

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



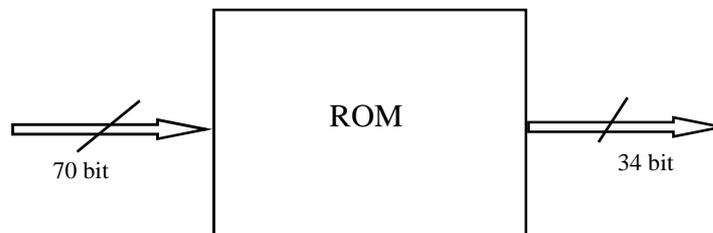
Input:

- 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:



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



L'approccio firmware



Nell'approccio firmware, viene inserita nella ALU una unità di controllo e dei registri. L'unità di controllo attiva opportunamente le unità aritmetiche ed il trasferimento da/verso i registri. Approccio "controllore-datapath".

Viene inserito un microcalcolatore dentro la ALU.

Il primo microprogramma era presente nell'IBM 360 (1964).



L'approccio firmware vs hardware



La soluzione HW è più veloce ma più costosa per numero di porte e complessità dei circuiti.

La soluzione firmware risolve l'operazione complessa mediante una sequenza di operazioni semplici. E' meno veloce, ma più flessibile e, potenzialmente, adatta ad inserire nuove procedure.

La soluzione HW è percorsa per le operazioni frequenti: la velocizzazione di operazioni complesse che vengono utilizzate raramente non aumenta significativamente le prestazioni (legge di Amdahl).



Sommario



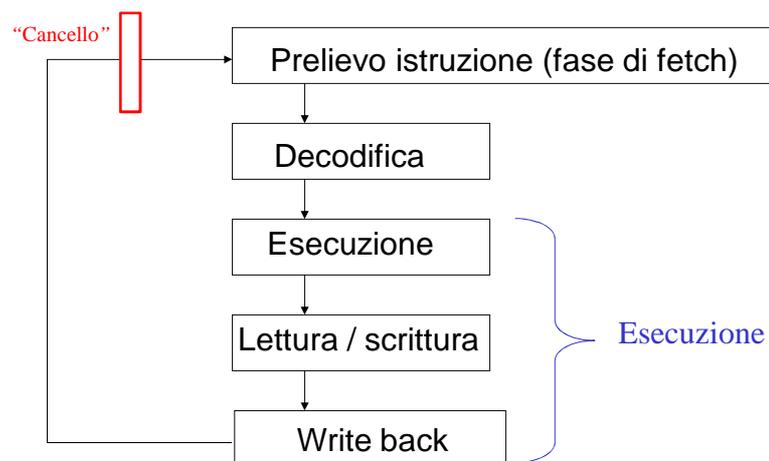
ALU: Comparazione, Overflow, Test di uguaglianza

Bistabili

Latch SC

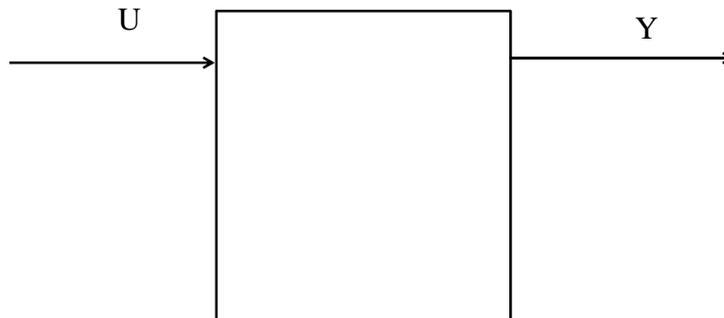


Dispositivi di sincronizzazione





Memorie



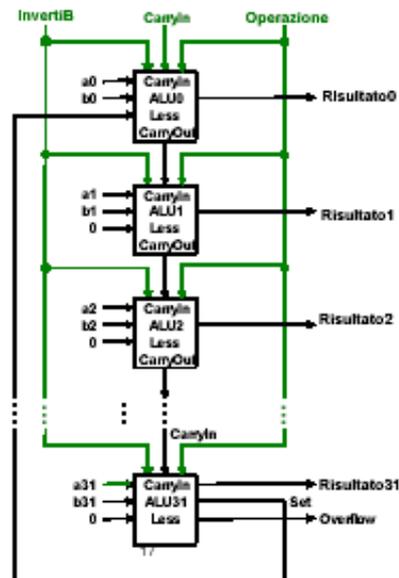
L'uscita Y mantiene il suo valore, {0,1}, anche quando l'ingresso U cambia valore.



Circuiti sincroni / asincroni



- **Architettura logica asincrona:**
L'elaborazione e propagazione dei segnali avviene in modo incontrollato, secondo le velocità di propagazione dei circuiti.
 - Non devo mai aspettare il “tick” di un clock → massima velocità
 - **Progetto asincrono:** Devo progettare il circuito in modo che nessun transitorio/cammino critico causi problemi → analisi di tutti i transitori critici possibili.
Imponibile per circuiti con feed-back.
Esempio: ALU.





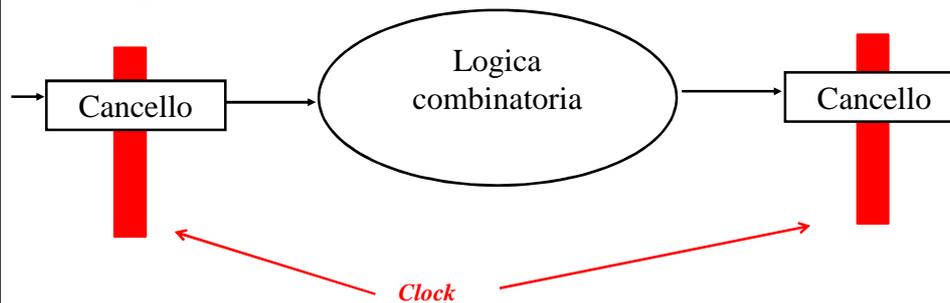
Circuiti sincroni

- **Architettura logica sincrona:**

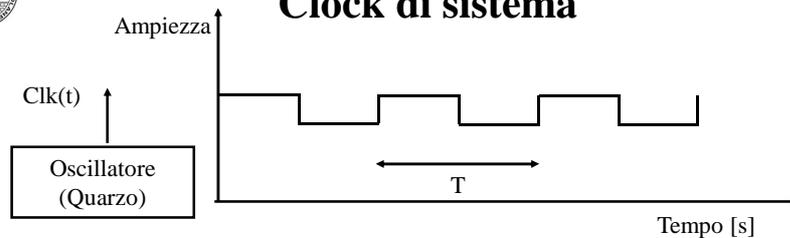
Le fasi di elaborazione sono scandite da un orologio comune a tutto il circuito (**clock**).

- 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



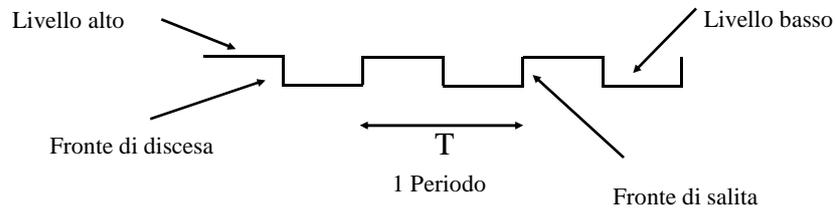
Frequenza: numero di cicli/s Si misura in Hertz, Hz.
Periodo: tempo necessario a completare 1 ciclo Si misura in secondi, s.

$$T = 1/f$$

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**. Gli output al tempo t dipendono unicamente dagli input al tempo t che provengono dall'esterno: $y^{t+1} = f(u^{t+1})$

- 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 la sequenza delle situazioni precedenti.

- I circuiti che hanno elementi di memoria consentono di eseguire operazioni **sequenzialmente** (scandite dal clock (e.g. CPU) o meno (e.g. Distributore di bibite)).



Perchè esiste il clock?



Esempio:

$$C = A + B$$

$$E = D + C$$

Quando posso calcolare E con lo stesso sommatore?

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

Ogni quanto tempo possiamo presentare gli ingressi al sommatore?

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

Occorre una sincronizzazione dell’attività del sommatore.



Bistabili: latch e flip-flop



Elemento cardine dei circuiti sequenziali è lo **stato**. Lo stato riassume il funzionamento negli istanti precedenti e deve essere immagazzinato (memorizzato).

Necessità di elementi con memoria (bistabili -> registri -> memorie).

Elemento base della memoria è il **bistabile**: dispositivo in grado di mantenere *indefinitamente* un certo valore 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 e 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



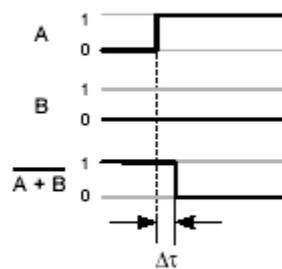
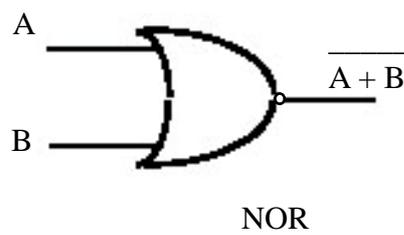
ALU: Comparazione, Overflow, Test di uguaglianza

Bistabili

Latch SC



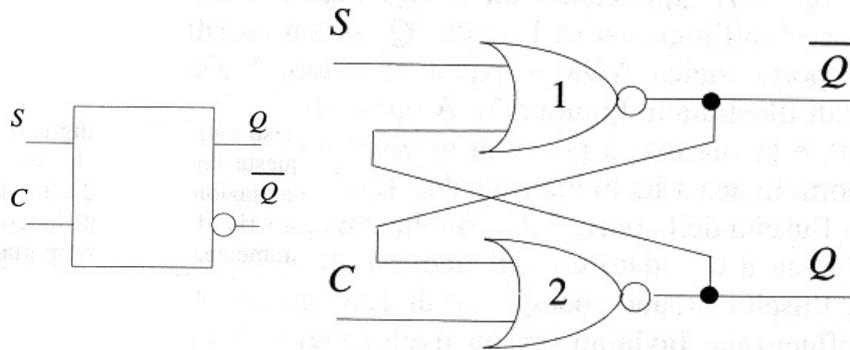
Principio di funzionamento



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



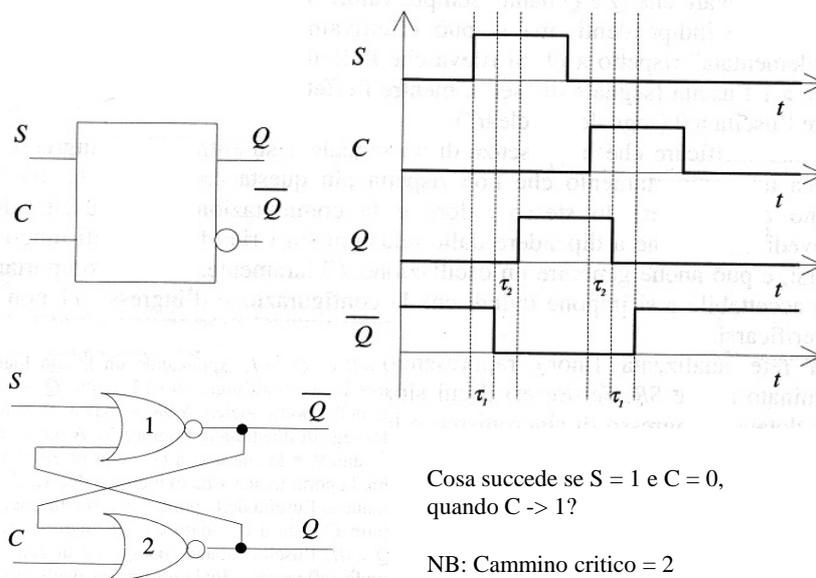
Latch asincrono SC (o SR)



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



Funzionamento del circuito SC



Cosa succede se $S = 1$ e $C = 0$,
quando $C \rightarrow 1$?

NB: Cammino critico = 2



Tabella della verità di SC

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



Q	SC = 00	SC = 01	SC = 10	SC = 11
0	0	0	1	X
1	1	0	1	X

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

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

X è impostato = 0

Status Quo Set



Tabella delle transizioni

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

Variabile di Stato (interna) Variabili di Ingresso (esterne)

Q	SC = 00	SC = 01	SC = 10	SC = 11
0	0	0	1	X
1	1	0	1	X

No change (Q* = Q) Memory Clear Reset Write 0 Set Write 1

Q è l'uscita del latch: **stato presente**, Q_t

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



Tabella della verità di SC - II



Impostando $X = 1$, si ottiene:

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

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

↙ Status quo

SC = 11 \rightarrow $Q^* = 1$ \rightarrow Set

Impostando $X = 0$, si ottiene:

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

$$\overline{S+C} Q + S\bar{C}$$

↙ Status quo

SC = 11 \rightarrow $Q^* = 0$ \rightarrow Reset

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 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