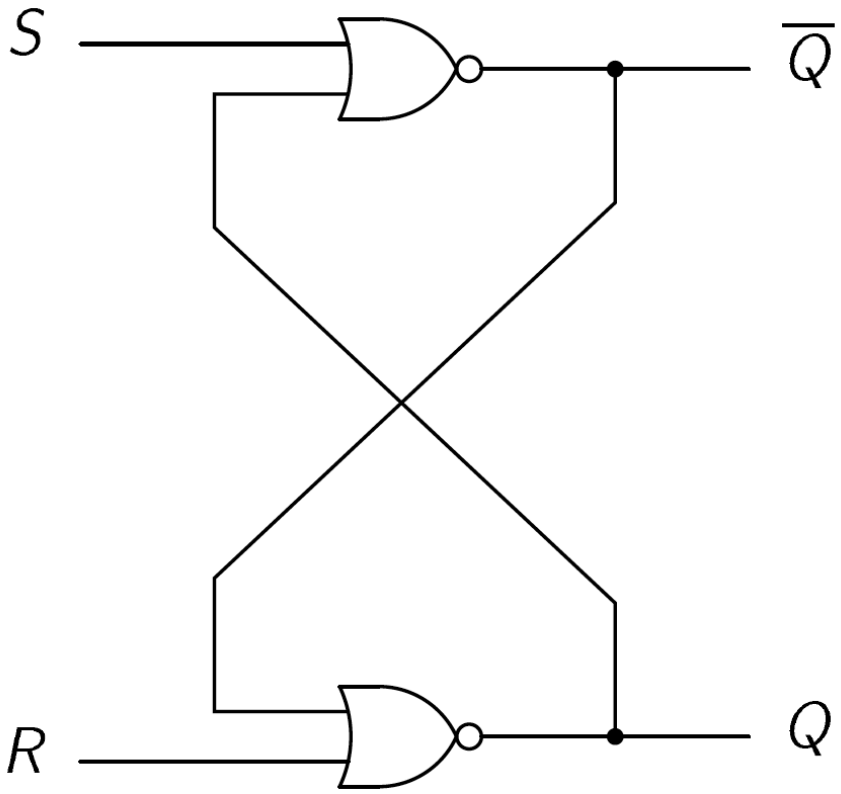
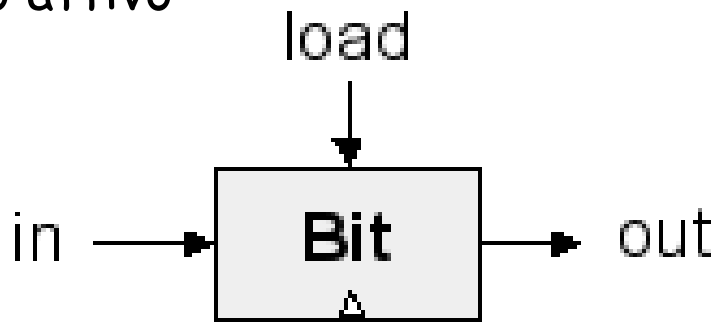
Circuiti Sequenziali



Prof. Ivan Lanese

Circuiti che hanno “memoria”

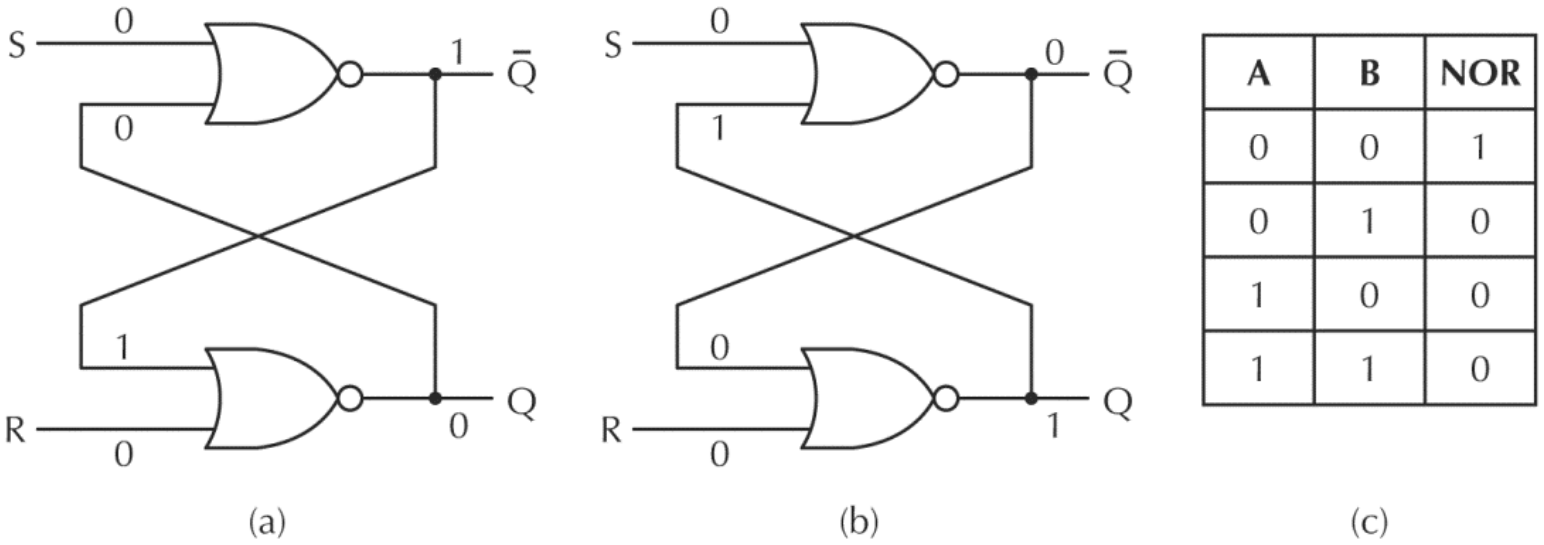
- Nei circuiti logici combinatori l’output in un certo istante dipende solamente dagli input nel medesimo istante
- Esistono però componenti, come le memorie, il cui output dipende da input precedenti
  - Ad esempio, qui sotto è mostrata una memoria di un bit:
    - “out” avrà il valore che aveva “in” nell’ultimo istante in cui “load” è stato attivo



    - in altre parole, questo componente “memorizza” il valore che aveva “in” nell’ultimo momento in cui è stato attivato “load”

Latch SR

- Un esempio base di circuito non combinatorio è il latch SR

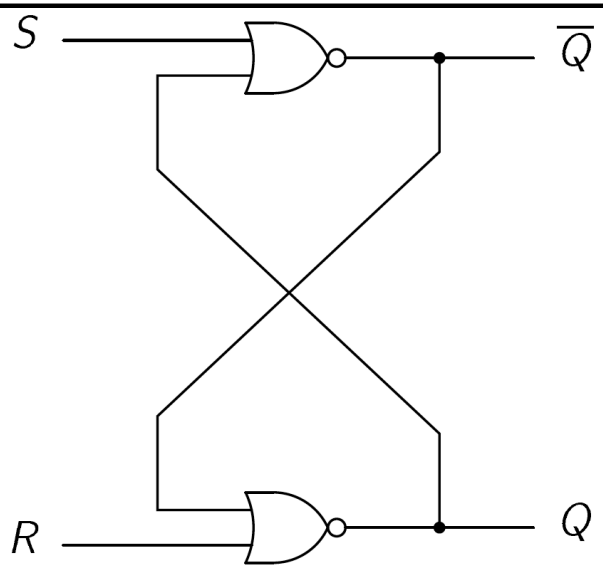


A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Figura 3.21 (a) Latch di tipo NOR nello stato 0. (b) Latch di tipo NOR nello stato 1. (c) Tabella di verità del NOR.

- Se i due input sono a 0, non si sa bene quale sarà l'output
- Vedremo che l'output dipenderà da valori precedenti dell'input
- OSSERVAZIONE: tale circuito contiene un ciclo, cosa che non è permessa nei circuiti combinatori!

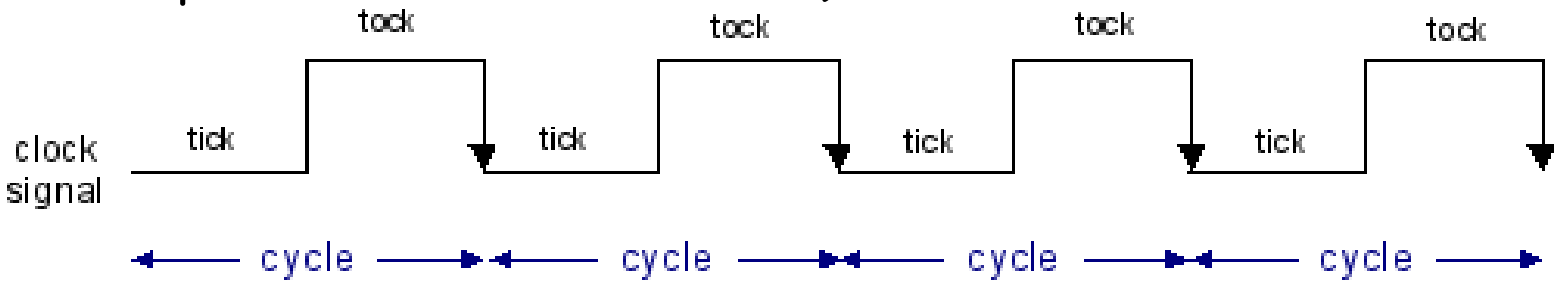
Latch SR - Analisi



- Consideriamo le 4 combinazioni di input
  - Se S=1, R=0 allora Q=1, $\overline{Q}$=0
  - Se S=0, R=1 allora Q=0, $\overline{Q}$=1
  - Se S=1, R=1 allora Q=0, $\overline{Q}$=0
  - Se S=0, R=0 allora … non si sa!
- Tale circuito è usato in modo tale da non avere mai come input S=1, R=1
- Quindi avremo che i due output sono sempre uno la negazione dell'altro
  - Sotto questa ipotesi riconsideriamo il caso S=0, R=0
    - Vediamo che gli output rimangono esattamente gli stessi dell'istante in cui entrambi gli input sono scesi a 0
- Il latch "memorizza" l'ultimo segnale: se è S allora Q=1, se è R allora Q=0

Latch SR temporizzato

- Di solito, si evita che il latch cambi il proprio stato in particolari momenti, e lo si rende sensibile agli input in altri intervalli di tempo
- Per realizzare questo si può utilizzare un segnale alternato 0-1, solitamente chiamato "clock" (durante 0 il latch non cambia stato, durante 1 può invece cambiare stato)



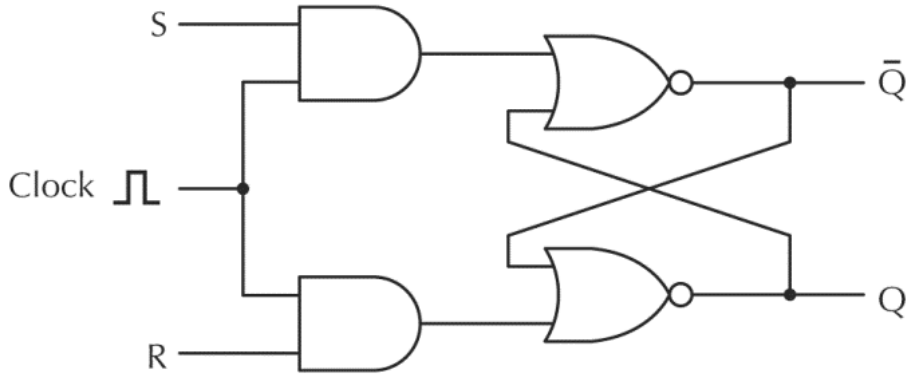


Figura 3.22 Latch SR temporizzato.

Latch D temporizzato

- Per essere sicuri che la situazione indesiderata (S=R=1) non si verifichi, si usa il latch D che ha un solo input (detto appunto D) che viene collegato a S, e negato per l'input R
- Il latch D cambia il proprio stato mentre il clock è attivo, durante la fase 0 del clock memorizza l'ultimo valore di D alla fine della fase 1
  - Questo comportamento è detto "commutazione a livello" (livello 1 di clock permette il cambio di stato)

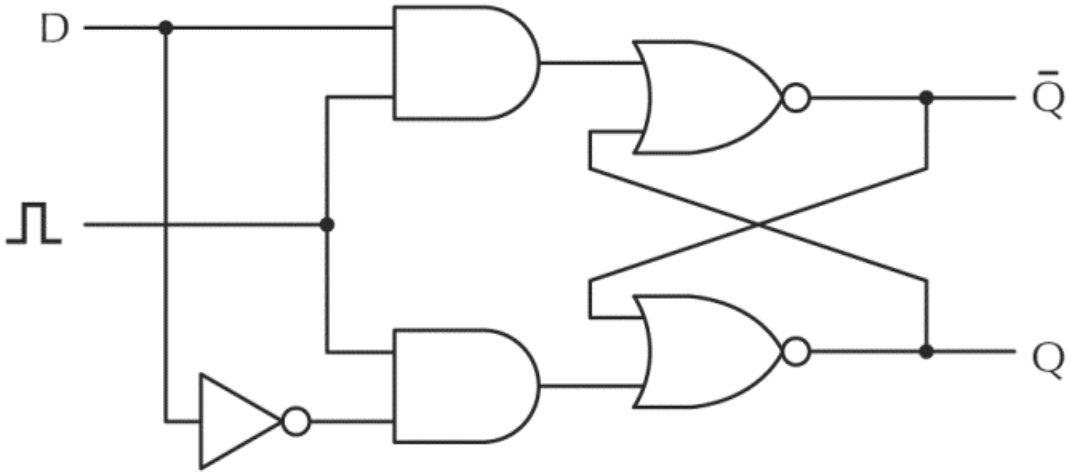


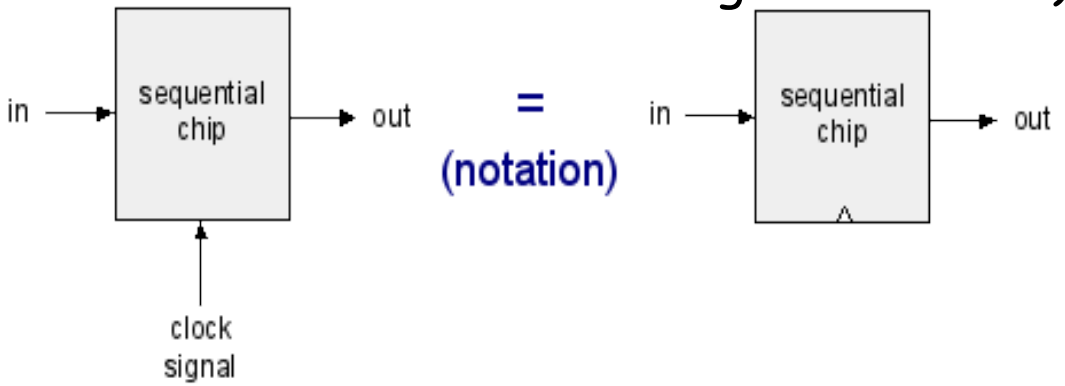
Figura 3.23 Latch D temporizzato.

Flip-Flop

- Circuiti logici con due stati stabili, con "commutazione sul fronte" (cioè cambio di stato solo quando il clock sale oppure quando scende) sono detti flip-flop (il nome deriva dal suono che facevano i primi circuiti bi-stabili a relè durante il loro cambiamento di stato)
- Esistono vari tipi di flip-flop, noi considereremo il flip-flop di tipo D (useremo la sigla DFF). "t-1" e "t" indicano cicli consecutivi del clock:

out(t) = in(t-1)

  - Il triangolo in basso indica che si tratta di circuito temporizzato (controllato dal fronte di salita di un segnale di clock)



Implementazione di DFF

- Ecco come implementare un DFF:

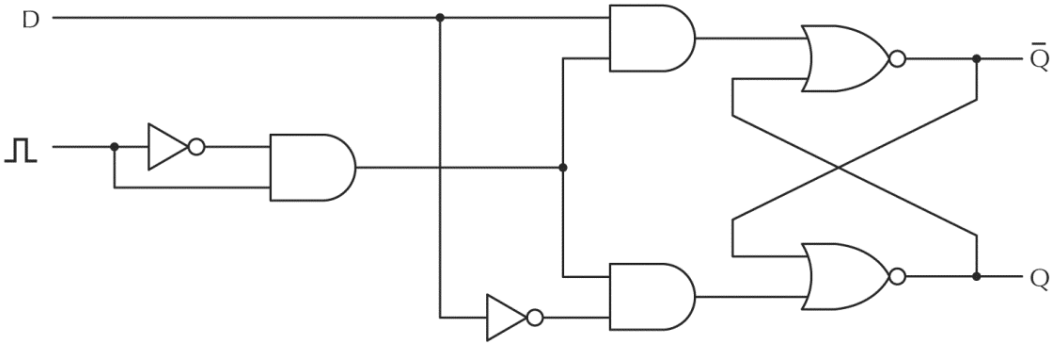


Figura 3.25 Flip-flop D.

- Il circuito sul clock crea un unico breve istante in cui il segnale in uscita è 1 (sul fronte di salita del clock)

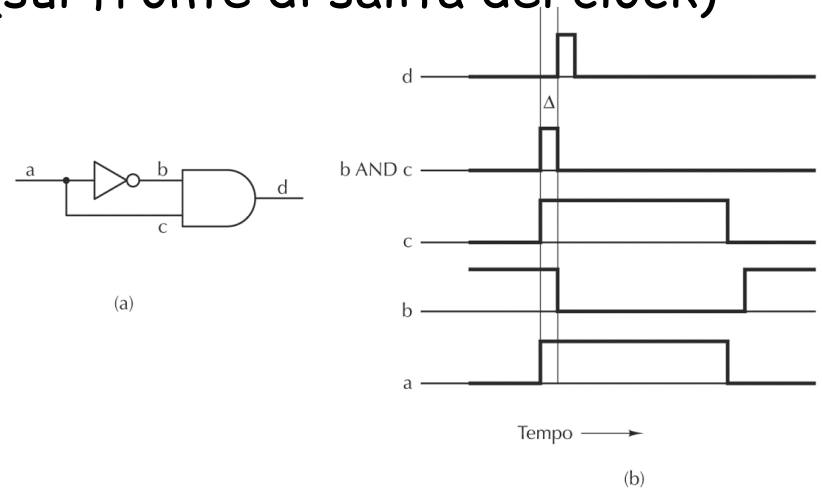
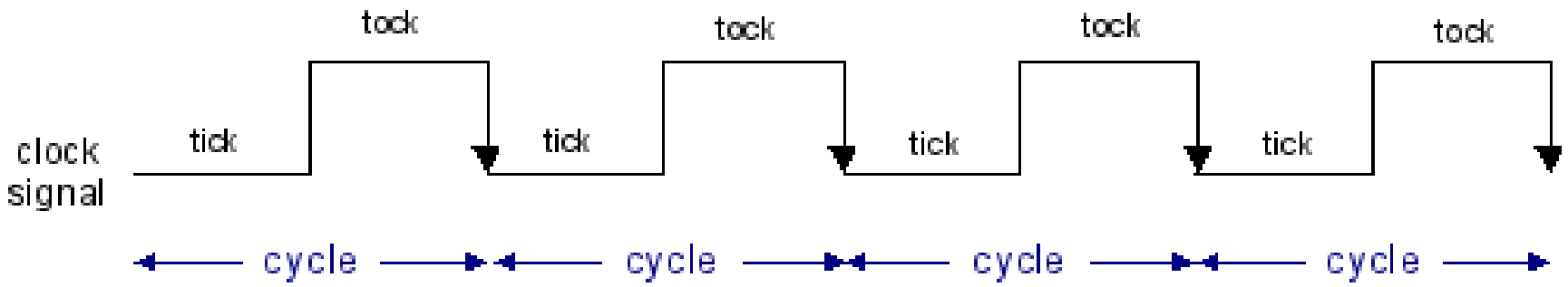


Figura 3.24 (a) Generatore d'impulsi. (b) Diagrammi temporali.

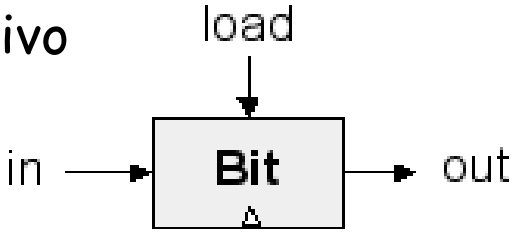
Interpretare la temporizzazione dei DFF (fronte di salita)



- Il DFF cambia stato qualche istante dopo il fronte di salita, in base agli input che ha durante il fronte di salita
- E' importante che il circuito si stabilizzi prima del nuovo fronte di salita

1-bit register

- Vediamo come realizzare una memoria da 1-bit usando un DFF
- Il 1-bit register "Bit" memorizza il valore di "in" nell'ultimo istante in cui "load" è stato attivo

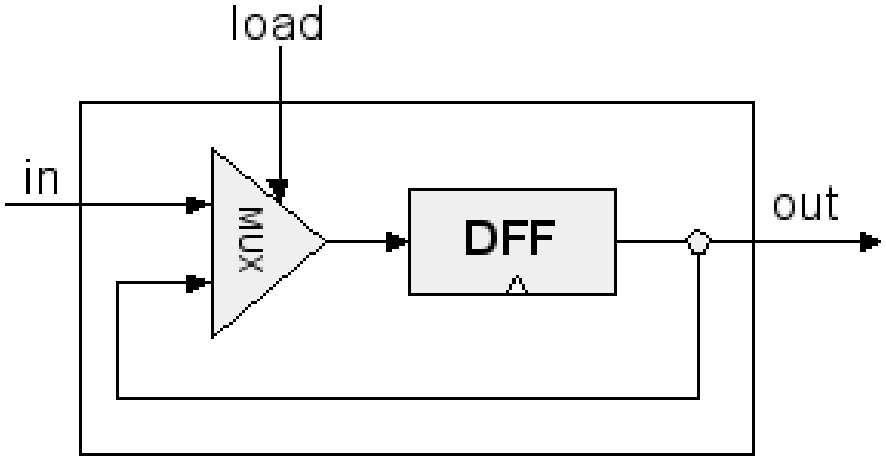


if load(t-1) then out(t)=in(t-1)
else out(t)=out(t-1)

- E' possibile usare un DFF per riportare in uscita all'istante t il valore deciso all'istante t-1 tramite un usuale circuito combinatorio

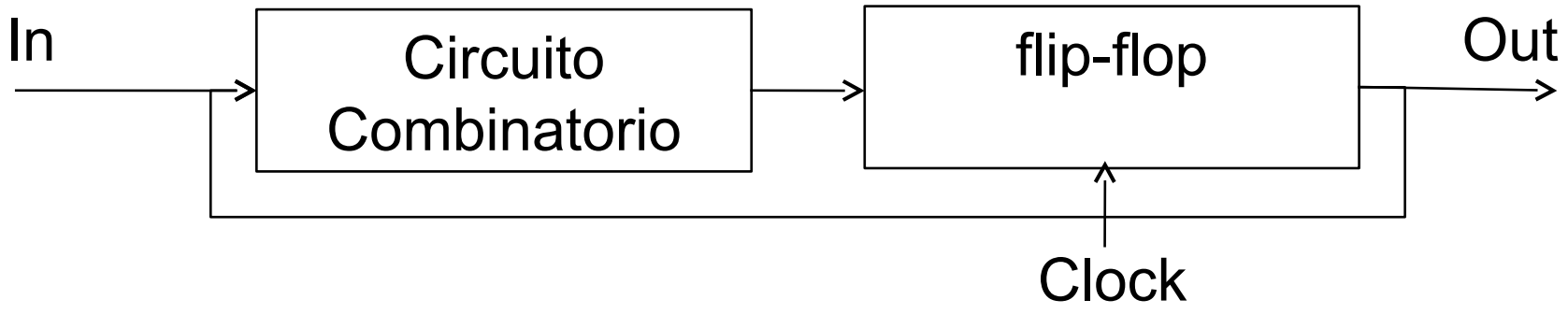
out(t) = in(t-1)

Basic building block



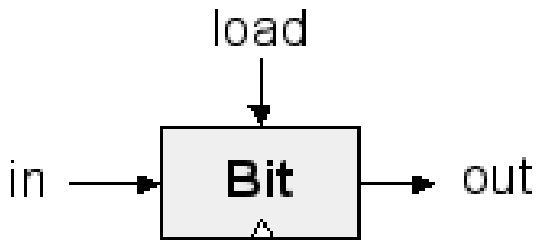
Schema tipico per realizzare circuiti sequenziali

- L'implementazione del 1-bit register ricalca uno schema tipico nella realizzazione di circuiti sequenziali
  - Un circuito combinatorio calcola il valore di uscita
  - Tale valore viene riportato al ciclo di clock successivo tramite un flip-flop, in modo tale che l'uscita successiva dipenda dall'uscita precedente e dagli input precedenti: out[t] = f(out[t-1], in[t-1])



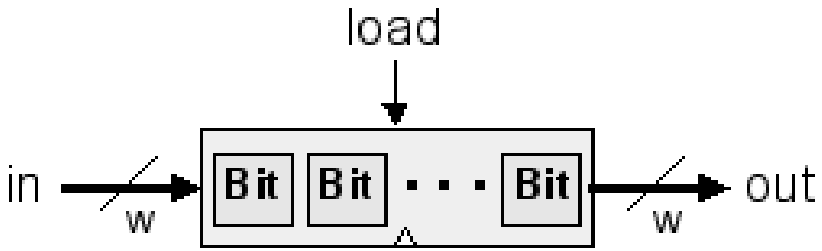
Registro multi-bit

- Vediamo ora come realizzare un memoria per un numero *w* di bit
- Chiameremo questo componente "w-bit register"



if load(t-1) then out(t)=in(t-1)
else out(t)=out(t-1)

1-bit register



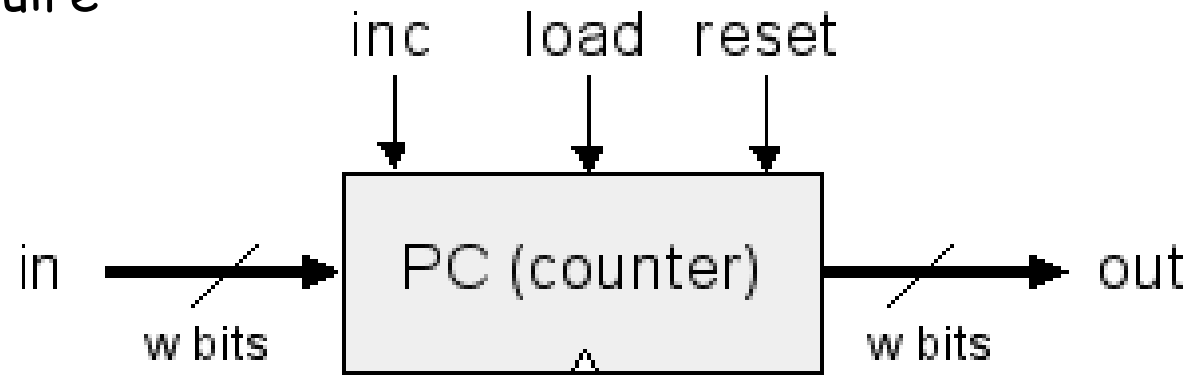
if load(t-1) then out(t)=in(t-1)
else out(t)=out(t-1)

w-bit register

- L'idea è usare un "1-bit register" per ogni linea del bus, e collegare tutti questi "1-bit register" al medesimo "load" e "clock"
- Ogni output dei "1-bit register" verrà collegato alla corrispondente linea del bus "out"

Contatore

- Un ulteriore circuito sequenziale largamente usato è il "counter"
  - Funziona come un registro ma ha una ulteriore modalità di incremento
  - Usato per realizzare il Program Counter (PC) che indica la locazione di memoria da cui prelevare la prossima istruzione da eseguire

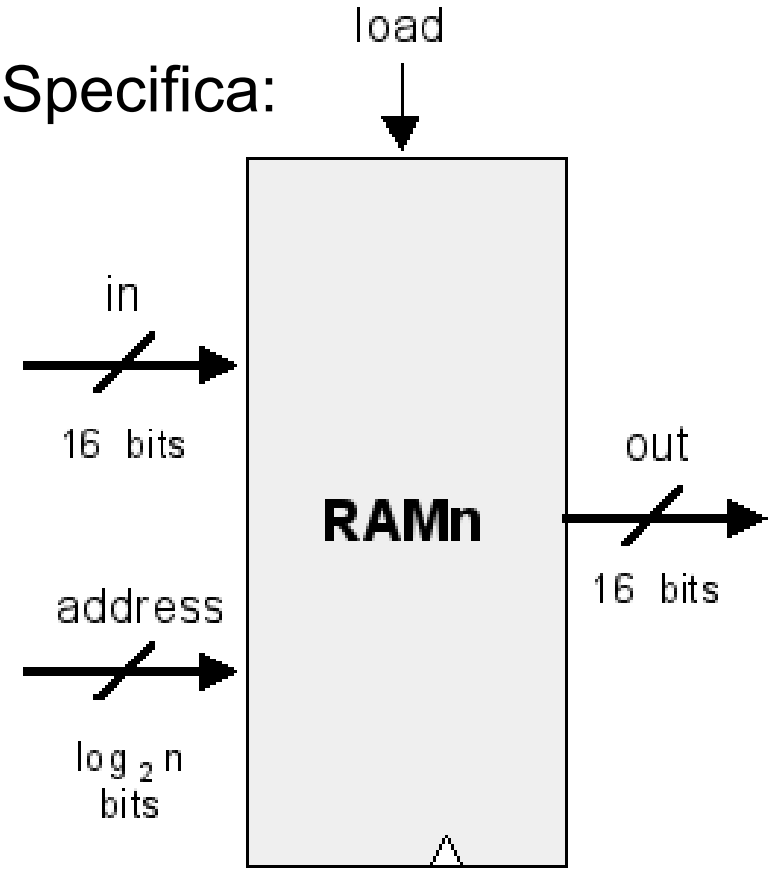



```
if reset(t-1) then out(t)=0
   else if load(t-1) then out(t)=in(t-1)
        else if inc(t-1) then out(t)=out(t-1)+1
             else out(t)=out(t-1)
```

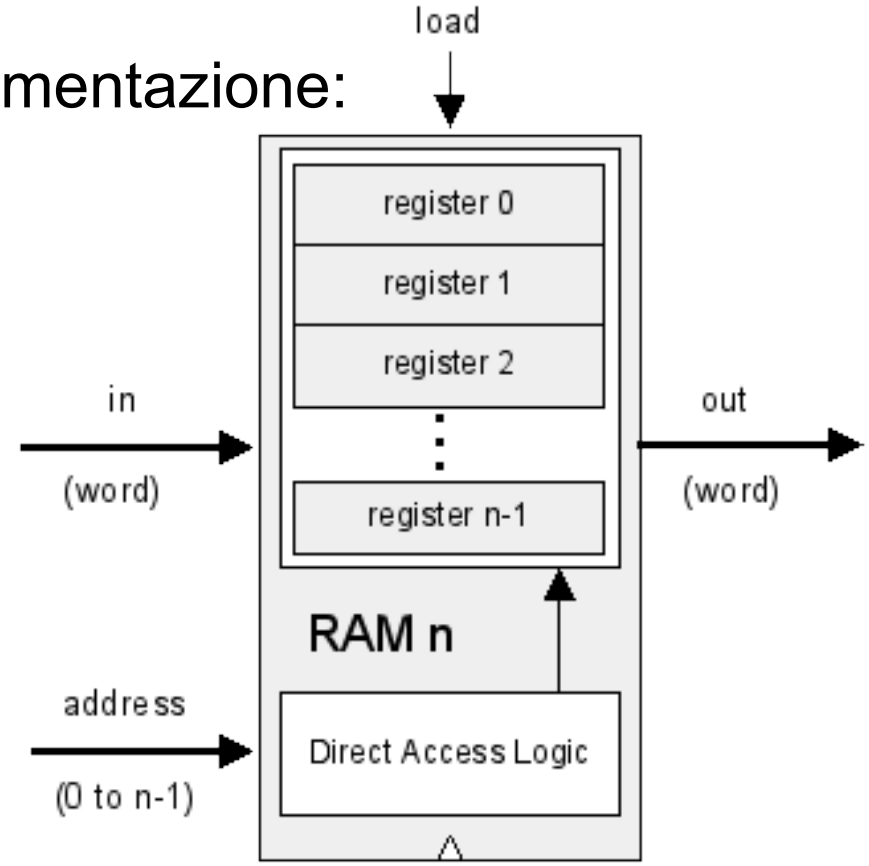
Memoria con n locazioni (implementazione usata per il processore Hack)

- Una memoria con n locazioni da w-bit, può essere realizzata con n "w-bit register" controllati da uno specifico circuito (Direct Access Logic) che indica quale di questi registri deve essere letto (e scritto se "load" è attivo)

Specifica:

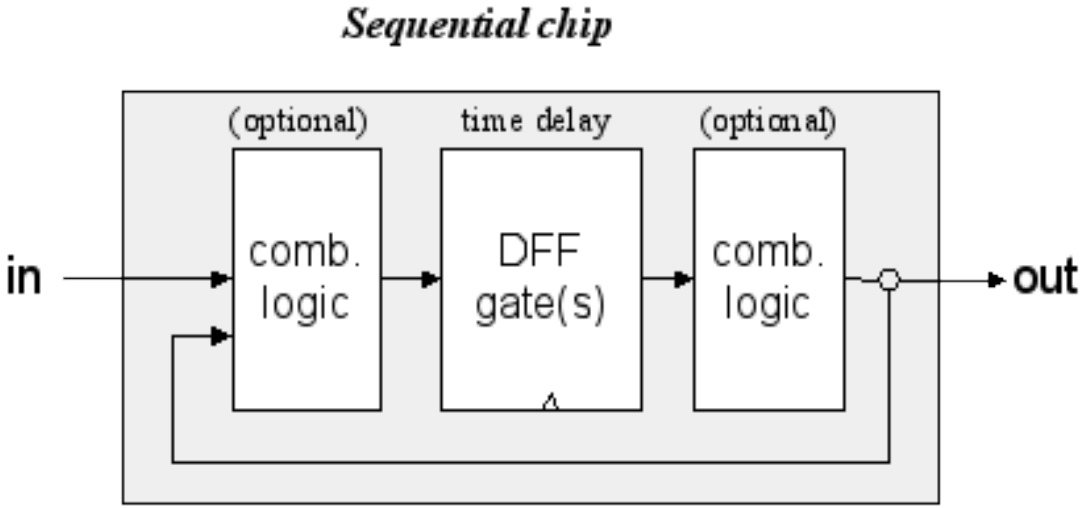


Implementazione:



Conclusioni: schema generale per realizzare circuiti sequenziali

- Nella implementazione della memoria della slide precedente, torna utile avere un circuito combinatorio anche dopo le uscite delle memorie
  - Utile per decidere quale delle uscite dei singoli registri deve essere portata su "out"
- In generale, quindi possiamo avere (o no) circuiti combinatori prima e dopo i flip-flop



Circuiti sequenziali con Hardware Simulator

- In HDL non potete realizzare latch e flip-flop come visto nelle slide precedenti
  - L'Hardware Simulator dà errore a causa del ciclo
- Avete però il DFF come circuito built-in che potete usare
- Costruirete tutti i circuiti sequenziali a partire dal DFF (e dai circuiti combinatori)
- Circuiti con cicli che includono dei DFF non danno errori
- Nell'Hardware Simulator e' possibile testare circuiti sequenziali modificando gli input e inviando impulsi di clock (sia manualmente che usando dei file .tst)

Livello logico digitale

- Ora abbiamo capito come funzionano i principali componenti usati nella relizzazione del processore di un calcolatore digitale: abbiamo cioè completato lo studio del livello logico digitale
- Tutti i componenti di questo tipo (ALU, registri, program counter,…) sono poi gestiti dal livello di "microarchitettura"
- E', ad esempio, la microarchitettura che indica come usare questi componenti per realizzare il ciclo FDE (Fetch-Decode-Execute)
- In alcune architetture tale livello è realizzato tramite microprogrammi, in altri viene realizzato direttamente in hardware
  - Nel nostro progetto del processore Hack, la microarchitettura è realizzata tramite hardware, cioè tramite circuiti logici