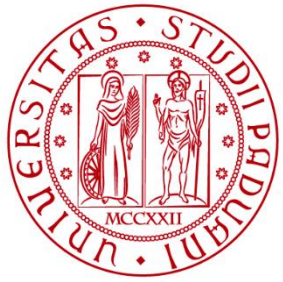
DEI
DIPARTIMENTO DI
INGEGNERIA DELL'INFORMAZIONE



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Sistemi Digitali

Registri e microoperazioni

Marta Bagatin, marta.bagatin@unipd.it

Corso di Laurea in Ingegneria dell'Informazione

Anno accademico 2022-2023

Scopo della lezione

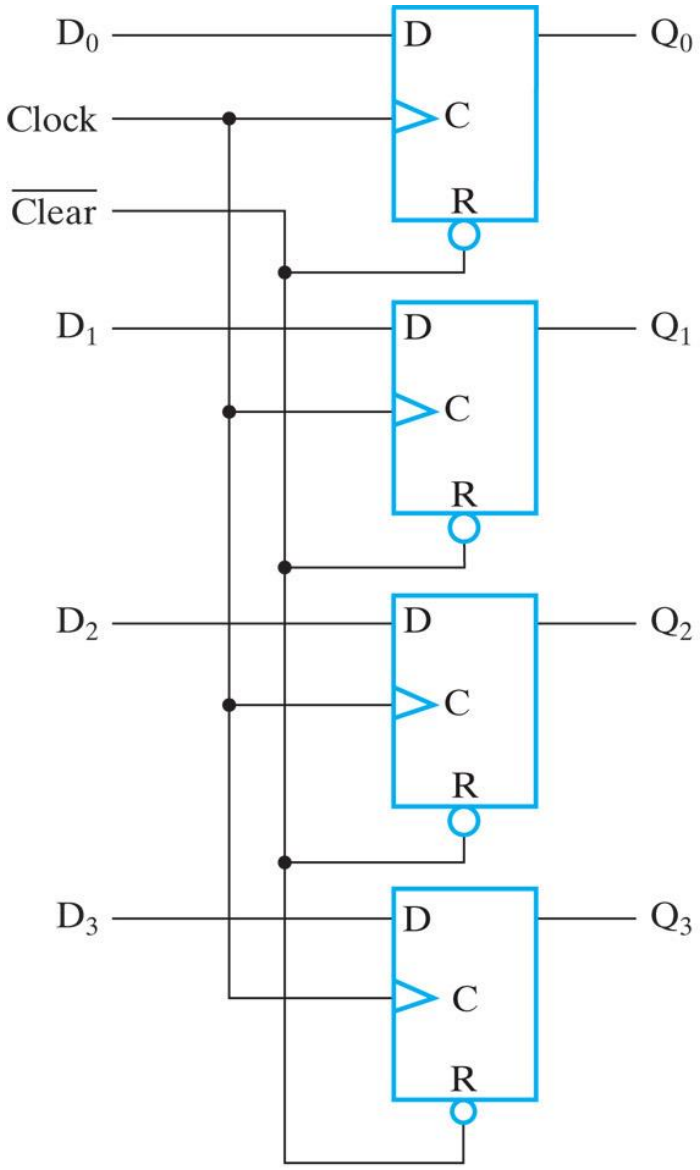
- Finora abbiamo visto circuiti combinatori e sequenziali. I sistemi sequenziali visti avevano un numero limitato di flip-flop ed erano agevolmente rappresentabili da un diagramma di stato
- Ora studieremo strutture più complesse, partizionate in diversi moduli, alcuni dei quali costituiti da un numero di blocchi simili o uguali ripetuti più volte
  - Descriveremo la struttura di un registro e come può avvenire il caricamento di dati in registri multi-bit
  - Definiremo i blocchi di datapath e control unit e capiremo come interagiscono tra loro nei sistemi digitali
  - Studieremo i tipi principali di operazioni di trasferimento di dati tra registri (microoperazioni) e come esprimerle in linguaggio RTL (Register Transfer Language)

Registri e contatori

- Sono blocchi che vengono usati in maniera estensiva nei sistemi digitali
- Un **registro a n bit** è costituito da n flip-flop e da un insieme di porte logiche
  - I flip-flop memorizzano i dati, le porte logiche trasformano i dati da trasferire ai flip-flop
- Un **contatore** è un particolare tipo di registro che transita attraverso una sequenza predefinita di stati

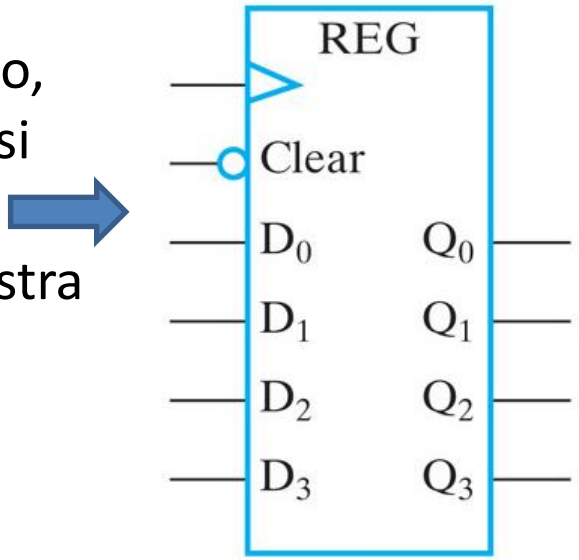
Registri

Registro a 4 bit

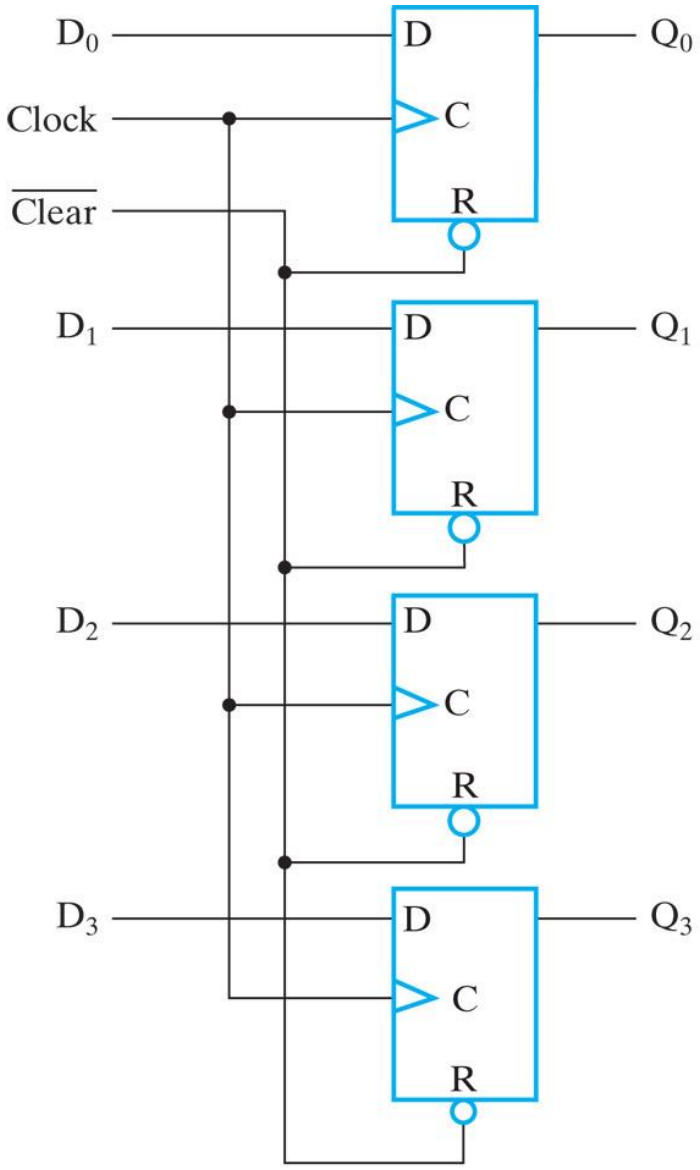


- Il registro più semplice è **costituito da soli flip-flop** e non contiene porte logiche
  - ciascuno dei 4 D flip-flop memorizza un bit

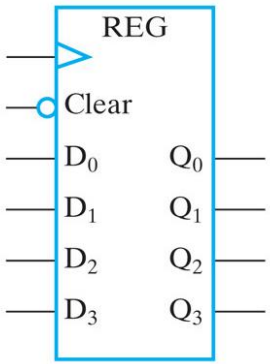
Simbolo logico, con gli ingressi a sinistra e le uscite a destra



Registro a 4 bit

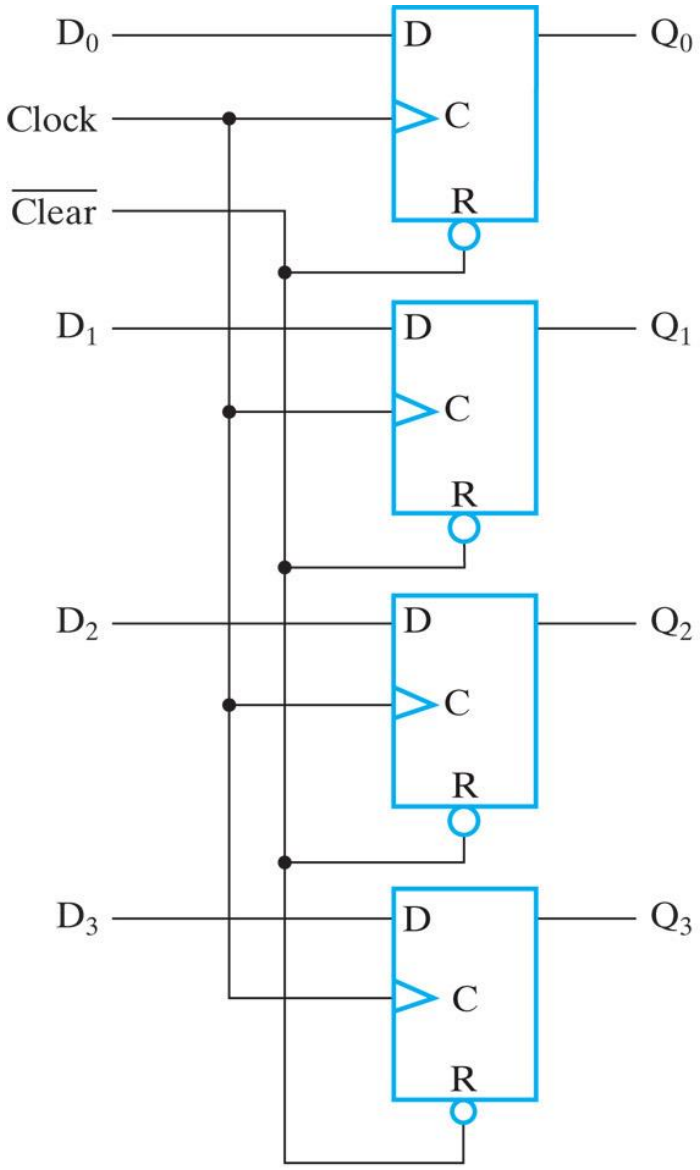


- Il **clock è in comune** tra i 4 flip-flop: al fronte di salita, i dati vengono trasferiti dall'ingresso all'uscita dei FF
- L'operazione di **trasferimento di nuovi dati** nel registro è detta **caricamento** (**load**), che in questo caso è eseguito in parallelo
- Un segnale opzionale di **RESET comune** ($\overline{clear}$) consente di porre a 0 tutte le uscite Q_i in modo **asincrono**



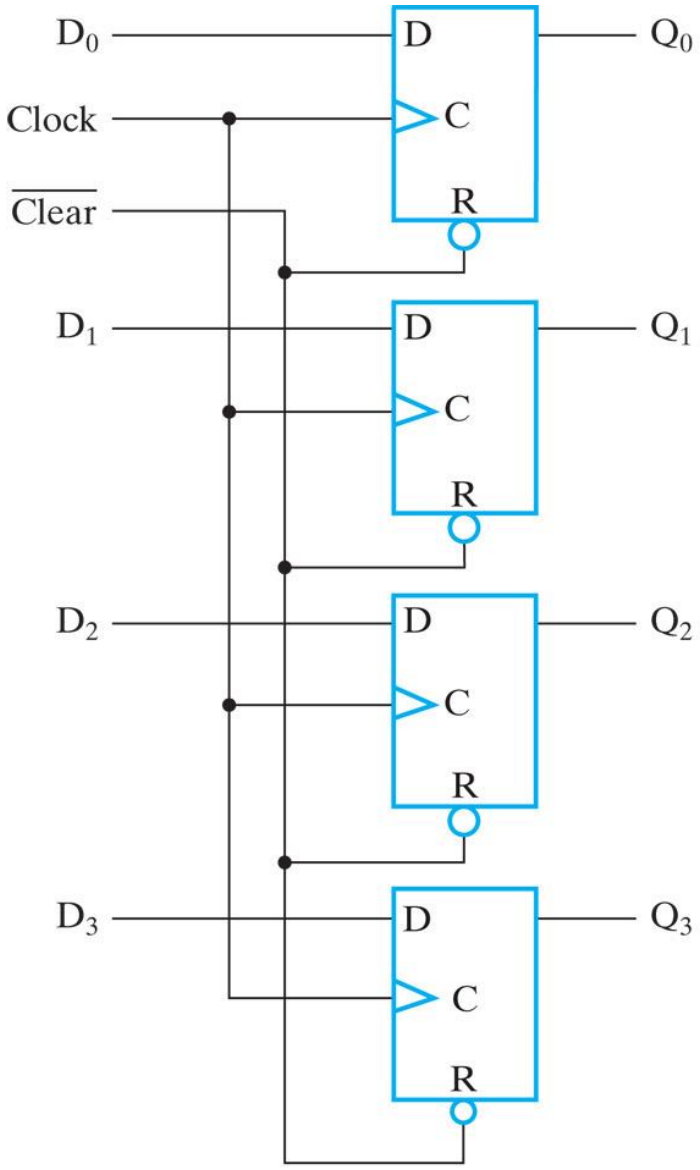
– Essendo negato (bubble), $\overline{clear}$ è 1 durante il normale funzionamento. Quando viene posto a 0, tutti i bit del registro vengono resettati (si dice che $\overline{clear}$ è un segnale attivo basso)

Registro: controllo del caricamento



- Si può prevedere un **controllo del caricamento dei dati**, qualora non si voglia che il contenuto dei flip-flop venga aggiornato ad ogni ciclo di clock, ma solo quando desiderato
- Vedremo due tecniche complementari per controllo del caricamento
  a) Clock gating
  b) Load enable

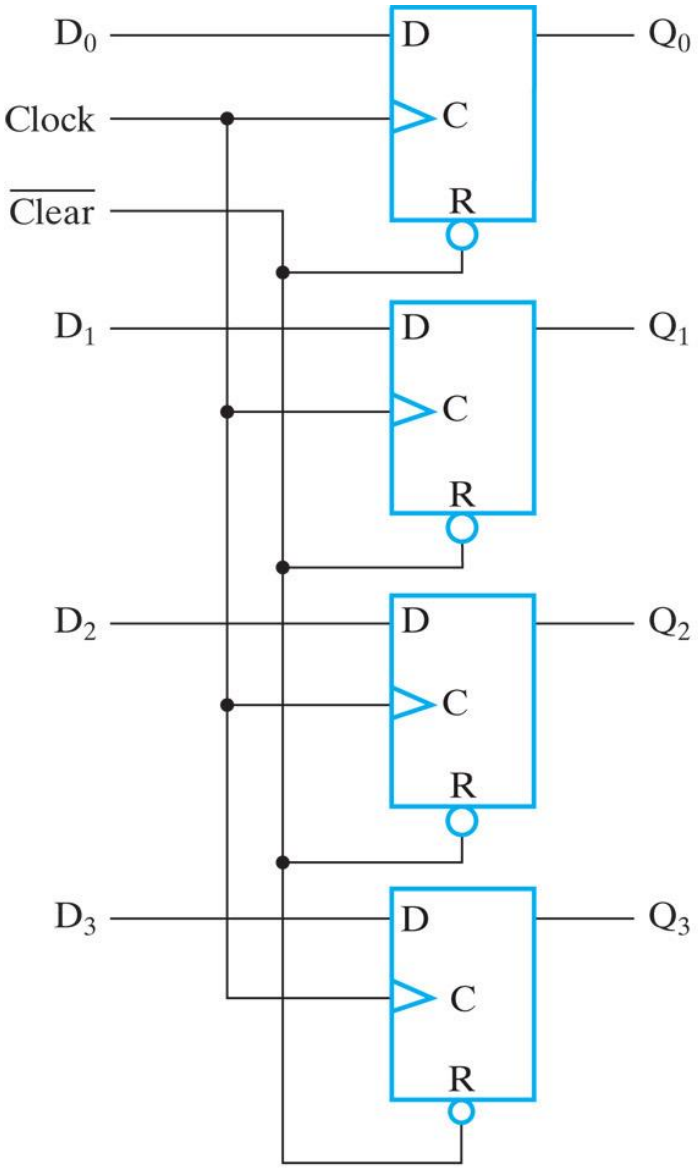
Controllo del caricamento: clock gating

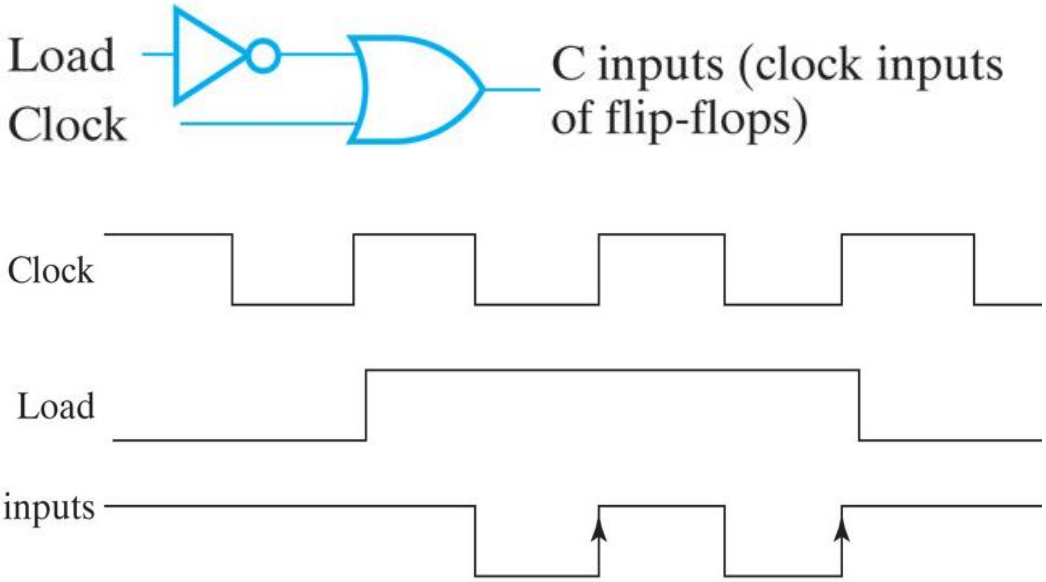


- Il clock del sistema (=master clock) genera una sequenza di impulsi continua
- La tecnica del clock gating consente di fare arrivare il **clock** agli elementi di memoria **solo quando desiderato**
  - Una porta logica OR dedicata controllata da Load inibisce l'arrivo del clock ai flip-flop del sistema (e.g., se non si vuole che contenuto del registro sia aggiornato)

$$\text{C inputs} = \text{Clock} + \overline{\text{Load}}$$

Controllo del caricamento: clock gating





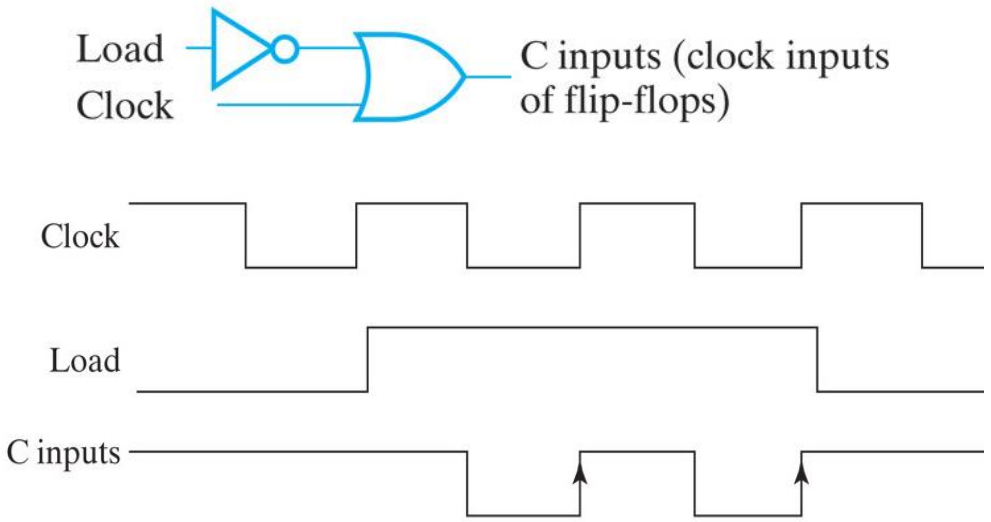
- **se Load = 1 => C inputs = Clock**

(il registro riceve il clock e il contenuto viene aggiornato ad ogni fronte di salita del clock)

- **se Load = 0 => C inputs = 1**

(il contenuto del registro non viene aggiornato, dato che il suo clock è stabile al valore 1)

Clock gating: osservazioni e svantaggi



$$\text{C inputs} = \text{Clock} + \overline{\text{Load}}$$

NOTA: Per il funzionamento corretto del circuito, il valore di Load deve essere costante al suo valore corretto durante il periodo in cui il clock è 0 (es. Load proviene da un FF-PET)

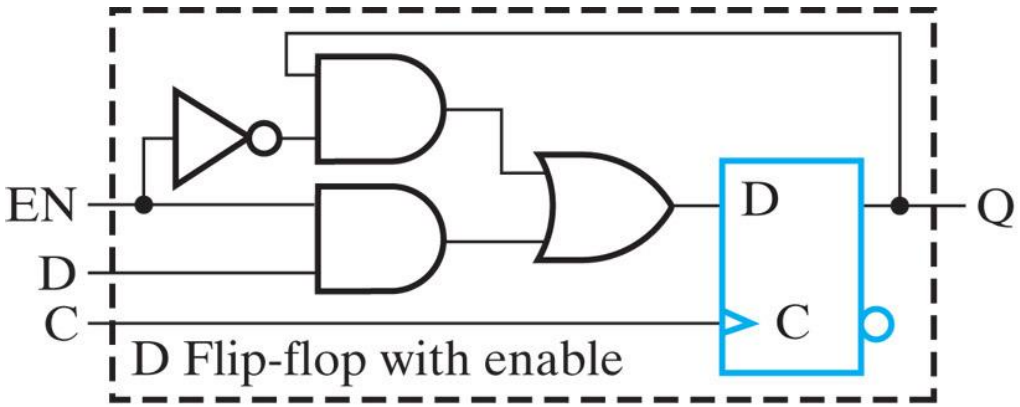
- La presenza di porte logiche tra il clock del sistema e il clock che effettivamente arriva ad ogni elemento di memoria introduce dei **ritardi di propagazione**
  - Il fenomeno per cui il clock arriva in momenti diversi ai vari elementi del sistema è chiamato **clock skew** (il sistema non è più sincrono!)
  - E' necessario un attento controllo dei tempi di ritardo al fine di minimizzare il clock skew, specie se si tratta di design ad alta velocità

Clock gating: vantaggi

$$\text{C inputs} = \text{Clock} + \overline{\text{Load}}$$

- La tecnica del clock gating in circuiti digitali complessi porta dei **vantaggi dal punto di vista del consumo di potenza**
- Tipicamente, non tutti i sottosistemi di un sistema funzionano tutti in contemporanea: il clock gating permette di **spegnere le parti del chip che non lavorano**, senza pregiudicare le prestazioni globali del sistema
- In sistemi complessi, tipicamente ogni blocco funzionale contiene una logica di clock gating

Controllo del caricamento: Load enable



- In alternativa a inibire il clock, si può controllare il caricamento dei dati in un registro tramite un apposito **segnale di abilitazione** (**Load EN**)

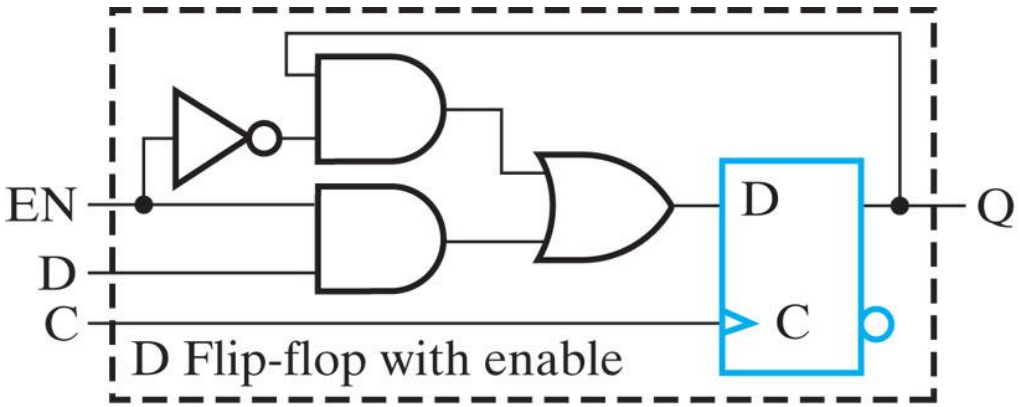
Il circuito consiste in un MUX 2-to-1 e un D-FF: il segnale EN seleziona il valore da trasmettere all'ingresso del FF tra D e l'uscita del FF

- **se EN = 1: viene selezionato D**

  → il registro viene caricato con il dato D al successivo fronte del clock
- **se EN = 0: viene selezionato Q**

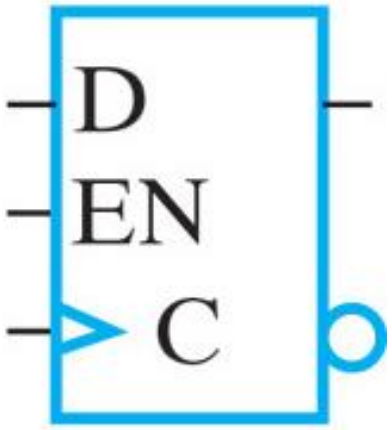
  → il registro viene caricato con la sua uscita Q, cioè non viene aggiornato e mantiene il valore precedente

Controllo del caricamento: Load enable

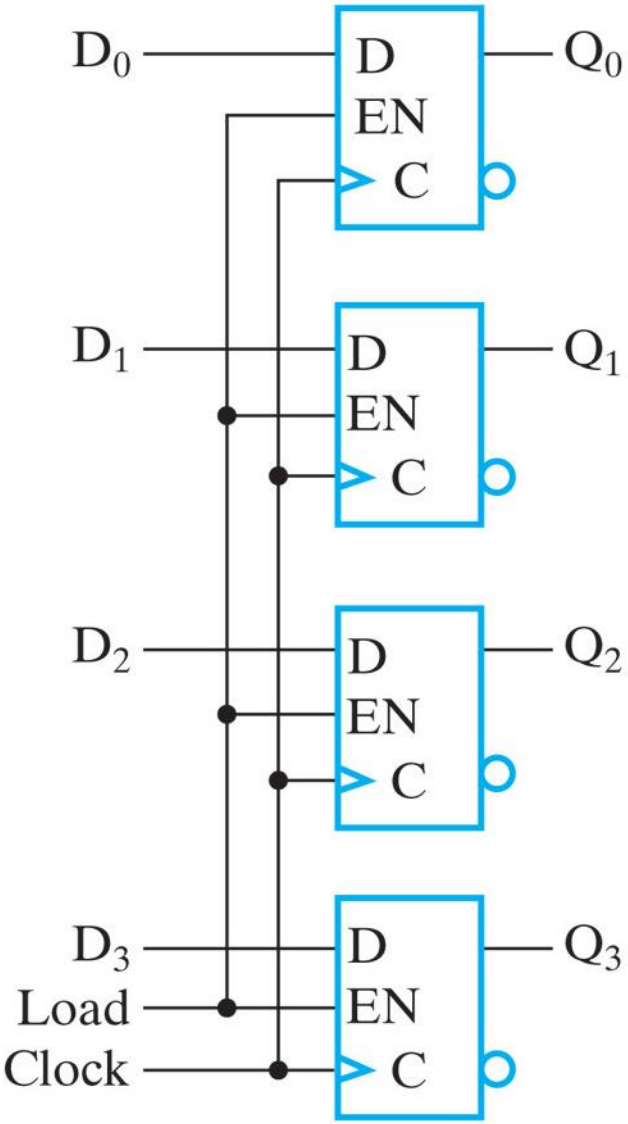


- In alternativa a inibire il clock, si può controllare il caricamento dei dati in un registro tramite un apposito **segnale di abilitazione** (**Load EN**)

Definiamo un nuovo elemento:
Flip-flip D con enable



Registro con Load enable



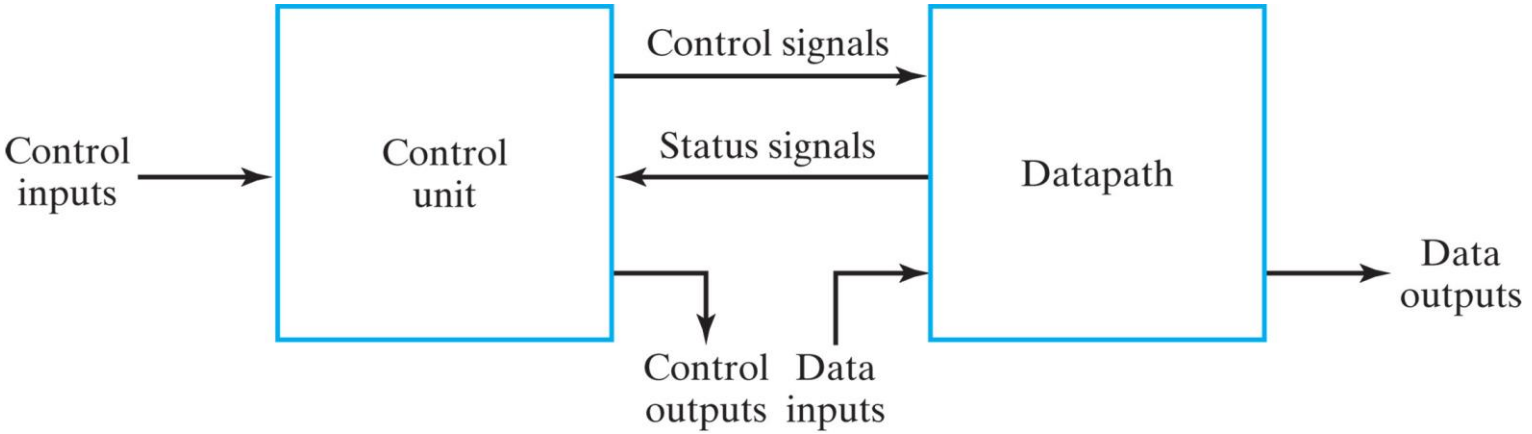
- Il registro è realizzato con **4 flip-flop D con EN in parallelo**, controllati dallo stesso segnale di clock
- Il clock è sempre applicato agli ingressi C dei flip-flop. E' il segnale Load a determinare se al successivo fronte di salita del clock saranno caricati nuovi valori all'interno del registro o se verrà mantenuto il valore precedente
- Questo metodo è **preferibile al clock gating per controllare il trasferimento di dati in un registro**, perché evita problematiche di clock skew e potenziali malfunzionamenti

Partizionamento di un sistema digitale in datapath e control unit

Datapath e Control Unit

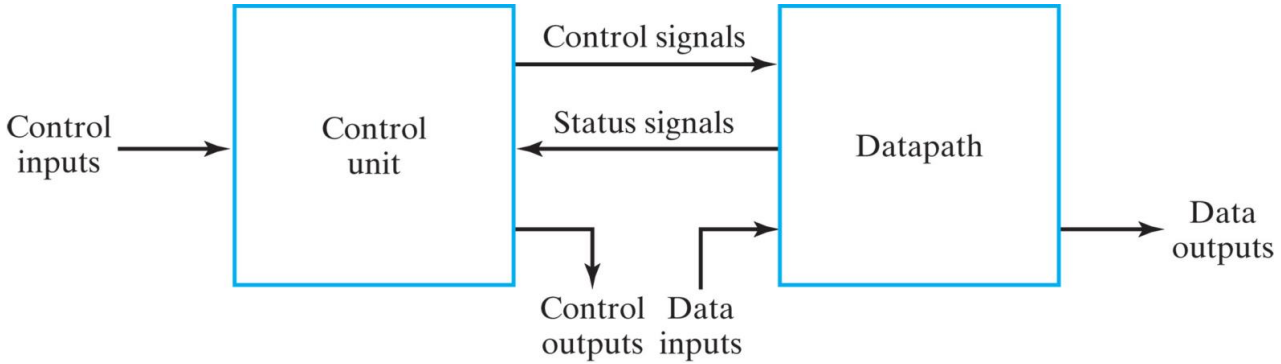
- Abbiamo visto che un sistema sequenziale può essere rappresentato con un diagramma (o tabella) degli stati
- In sistemi con un elevato numero di stati, specificare il comportamento del circuito con un diagramma/tabella degli stati può essere difficile: è comodo usare un **approccio gerarchico** e **partizionare il sistema in sottoblocchi**, ciascuno con la propria funzione

Datapath e Control Unit



- Molti sistemi digitali complessi possono essere partizionati in due macroblocchi: il **datapath svolge le operazioni di processamento dei dati** (aritmetico, logico, etc.), la **control unit determina l'ordine di tali operazioni**
  - La control unit invia al datapath dei segnali di controllo, che determinano le operazioni di processamento dei dati
  - Il datapath, a sua volta, manda i segnali di stato alla control unit, in base a cui quest'ultima decide quale sequenza di operazioni attivare
- Datapath e control unit possono anche comunicare con altri blocchi del sistema, come memoria e I/O

Datapath e Control Unit



- Possiamo pensare la **control unit** come una **macchina sequenziale sincrona** che, sulla base dei segnali di ingresso (control inputs) e dei segnali di stato del datapath (status signals), determina quali operazioni il datapath deve eseguire, generando opportuni segnali di controllo (control signals)
- Il **datapath** è costituito da **registri e blocchi aritmetico-logici** che eseguono le operazioni sui dati. Il datapath ha in ingresso i segnali di controllo prodotti dalla control unit, oltre ai segnali di ingresso dati (data inputs). In uscita il datapath ha le uscite dati (data outputs) e altri segnali che informano la control unit su stato/esito delle operazioni (status signals)
- Le operazioni di trasferimento di dati tra registri e le operazioni elementari sui dati che possono avvenire nel datapath sono dette **microoperazioni**

Sistemi programmabili e non programmabili

- In un **sistema programmabile** gli ingressi dell'unità di controllo contengono un flusso di istruzioni (programma)
  - Ogni istruzione attiva un'opportuna sequenza di microoperazioni da impartire al datapath. Le istruzioni specificano le microoperazioni che devono essere eseguite, con quali dati, e dove devono essere memorizzati i risultati
  - Il programma è tipicamente immagazzinato in una memoria (RAM o ROM). L'unità di controllo contiene un registro, program counter (PC), che in ogni istante specifica l'indirizzo dell'istruzione da prelevare dalla memoria durante il successivo ciclo di clock
  - Esempio: CPU
- In un **sistema non programmabile** la control unit non esegue le istruzioni contenute in un programma, ma determina le microoperazioni da assegnare al datapath unicamente sulla base dei segnali di ingresso e dei segnali di stato del datapath
  - E' un sistema adibito a svolgere un unico compito
  - Esempio: macchina a stati in cui l'aggiornamento dello stato e il calcolo delle uscite è realizzato tramite una logica fissa (es. chip usato in un orologio digitale)

Microoperazioni

Microoperazioni e RTL

- Le **operazioni elementari eseguite sui dati di un registro** sono dette microoperazioni
  - Es: incremento del contenuto di un registro, scambio dei dati presenti in due registri, scorrimento a sinistra delle cifre di un dato
- Il risultato della microoperazione può sostituire il valore presente nel registro stesso oppure essere trasferito in un altro registro, lasciando invariato il contenuto del primo
- Il compito della **control unit è fornire i segnali di controllo per dirigere la sequenza di microoperazioni svolte dal datapath**
- Il linguaggio con cui sono espresse le microoperazioni è detto **Register Transfer Language (RTL)**
  - Set di espressioni che ricordano gli statement usati negli HDL

Register Transfer Language

Operation	Text RTL	VHDL
Combinational assignment	=	<= (concurrent)
Register transfer	←	<= (concurrent)
Addition	+	+
Subtraction	−	−
Bitwise AND	∧	and
Bitwise OR	∨	or
Bitwise XOR	⊕	xor
Bitwise NOT	– (overline)	not
Shift left (logical)	Sl	sll
Shift right (logical)	Sr	srl
Vectors/registers	$A(3:0)$	A(3 down to 0)
Concatenation	‖	&

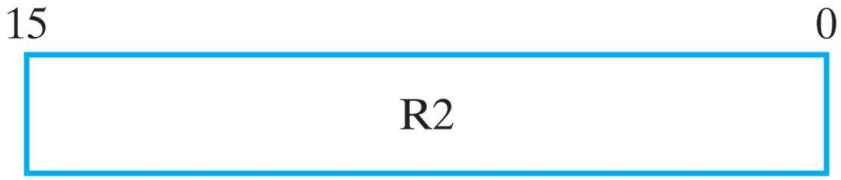
Le espressioni RTL che useremo sono simili ma non identiche a quelle usate nel linguaggio VHDL

Notazioni per indicare i registri

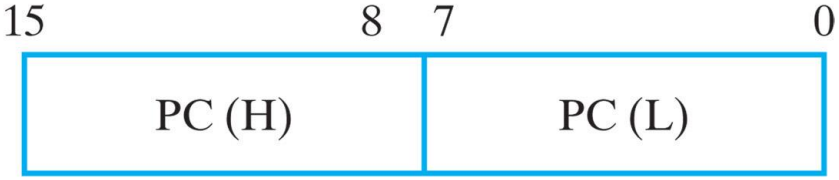
- Indichiamo un **registro** con lettere maiuscole, talvolta seguite da numeri (Address Register: AR, Program Counter: PC, etc.)
- Gli n bit del registro 0, 1, …, n-1 (flip-flop del registri) possono essere indicati tra parentesi dopo il nome del registro
  - Talvolta sono divisi in byte più significativo (H) e meno significativo (L)

(a) Register R

(b) Individual bits of 8-bit register



(c) Numbering of 16-bit register



(d) Two-part 16-bit register

Tipi di microoperazioni

- Sono le **operazioni elementari tra dati immagazzinati nei registri**
- Le microoperazioni più frequenti in un sistema digitale sono
  a) Microoperazioni di **trasferimento**: trasferiscono dati binari tra registri
  b) Microoperazioni **aritmetiche**: realizzano operazioni aritmetiche tra i dati contenuti nei registri
  c) Microoperazioni **logiche**: realizzano manipolazioni logiche sui bit dei dati contenuti nei registri
  d) Microoperazioni di **shift**: realizzano operazioni di scorrimento sui bit dei dati contenuti nei registri
- Una microoperazione può appartenere a più di una tipologia (per es. il complemento a 1 è un'operazione sia logica che aritmetica)

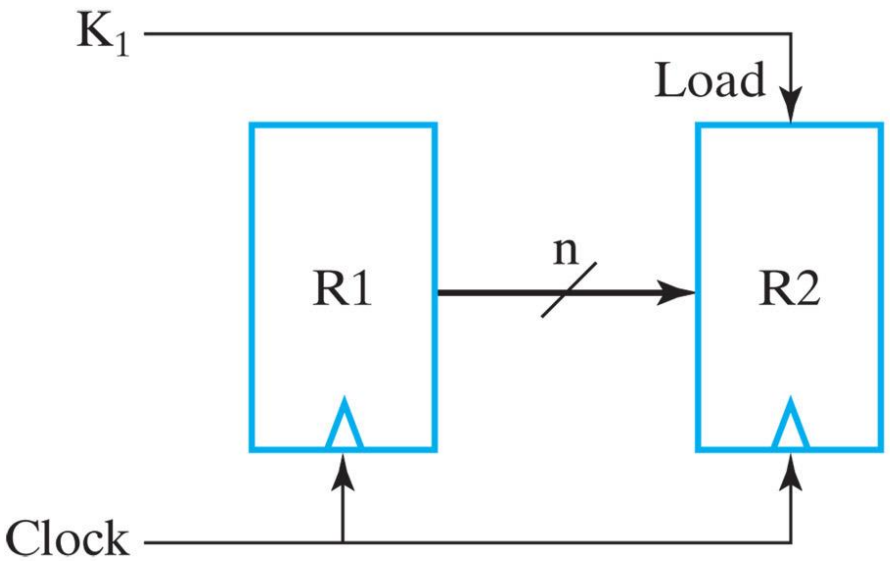
Trasferimento di dati tra registri

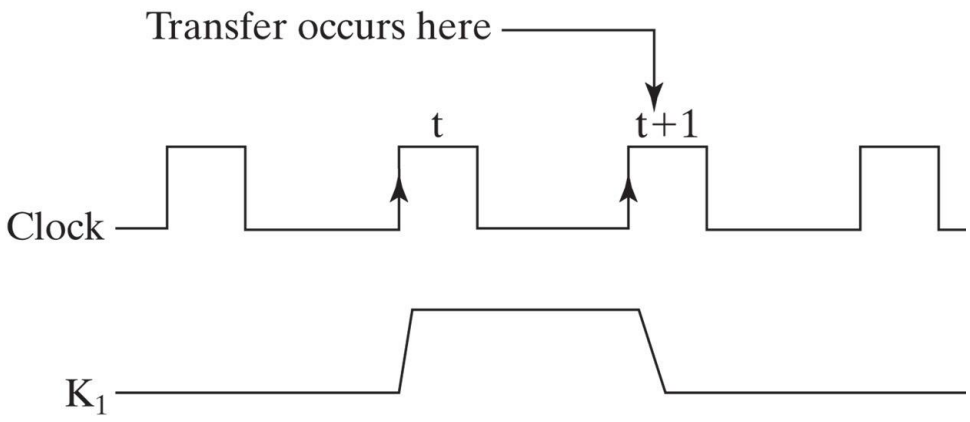
- Il trasferimento dei dati contenuti nei registri viene indicato con l'**operatore di trasferimento:** $\leftarrow$
  - Es: $\mathbf{R2 \leftarrow R1}$

    Indica il trasferimento del contenuto di R1 (sorgente) in R2 (destinazione): per definizione in un trasferimento cambia il contenuto del registro di destinazione ma non del sorgente
- Un trasferimento tra registri può essere **condizionale**, cioè **basato sul valore di uno o più segnali di controllo** generati nella CU
  - Esempio: $\mathbf{if\ (K_1 = 1)\ then\ (R2 \leftarrow R1)}$
  - Notazione compatta: $\mathrm{K_1}\colon\ \mathrm{R2} \leftarrow \mathrm{R1}$

    Indica il trasferimento del contenuto di R1 in R2 alla condizione (espressa prima di **:**) che il segnale di controllo K_1 valga 1

Trasferimento di dati tra registri





$\mathbf{if\ (K_1 = 1)\ then\ (R2 \leftarrow R1)}$

- n bit di uscita di R1 sono connessi all'ingresso di R2
- K_1 (sincronizzato con il clock che regola i registri) è collegato all'ingresso Load di R2. **K_1 attiva il trasferimento del contenuto di R1 a R2, al successivo fronte del clock**

NOTA: Il clock non compare tra gli ingressi nello statement RTL: si assume che i trasferimenti avvengano in corrispondenza alle transizioni del clock

Register transfer: simbologia

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	$AR, R2, DR, IR$
Parentheses	Denotes a part of a register	$R2(1), R2(7{:}0), AR(L)$
Arrow	Denotes transfer of data	$R1 \leftarrow R2$
Comma	Separates simultaneous transfers	$R1 \leftarrow R2,\ R2 \leftarrow R1$
Square brackets	Specifies an address for memory	$DR \leftarrow M[AR]$

- Trasferimenti simultanei di dati tra registri vengono separati da virgole
  - Esempio: $\mathbf{K_3}$: $\mathbf{R2 \leftarrow R1,\ R1 \leftarrow R2}$

    Indica lo scambio tra il contenuto di R1 e di R2 al fronte di clock in cui il segnale di controllo K_3 vale 1

Microoperazioni aritmetiche

- Le **operazioni aritmetiche di base** sono somma (+), sottrazione (-), incremento (+1), decremento (-1) e complemento ($\overline{x}$)
- Moltiplicazione (*) e divisione (/) non sono considerate nel set di operazioni elementari, potendo essere realizzate da una sequenza di microoperazioni di base

Symbolic Designation	Description
$R0 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R0$
$R2 \leftarrow \overline{R2}$	Complement of the contents of $R2$ (1s complement)
$R2 \leftarrow \overline{R2} + 1$	2s complement of the contents of $R2$
$R0 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus 2s complement of $R2$ transferred to $R0$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ (count up)
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ (count down)

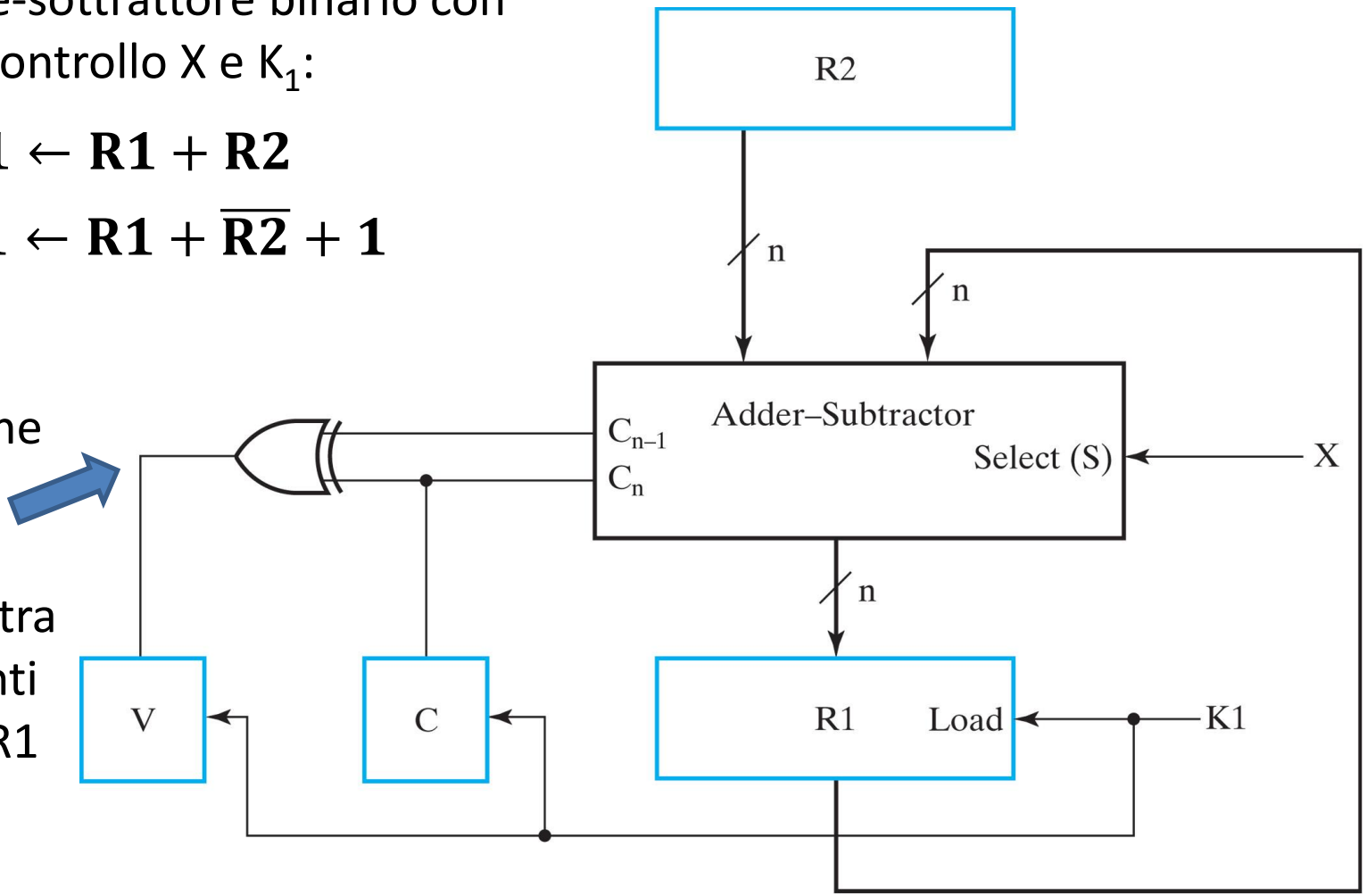
Microoperazioni aritmetiche: esempio

Statement RTL che descrivono un sommatore-sottrattore binario con segnali di controllo X e K_1:

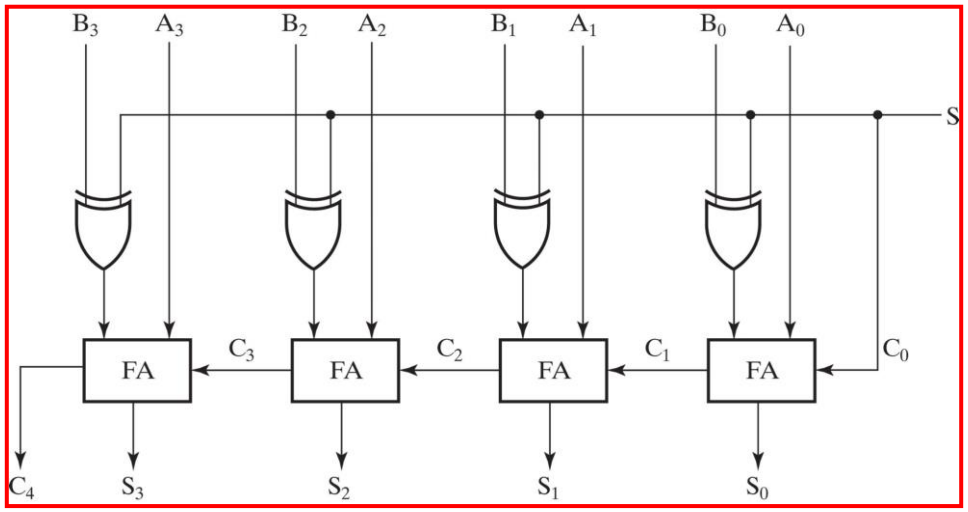
$$\overline{X}K_1: \; R1 \leftarrow R1 + R2$$

$$XK_1: \; R1 \leftarrow R1 + \overline{R2} + 1$$

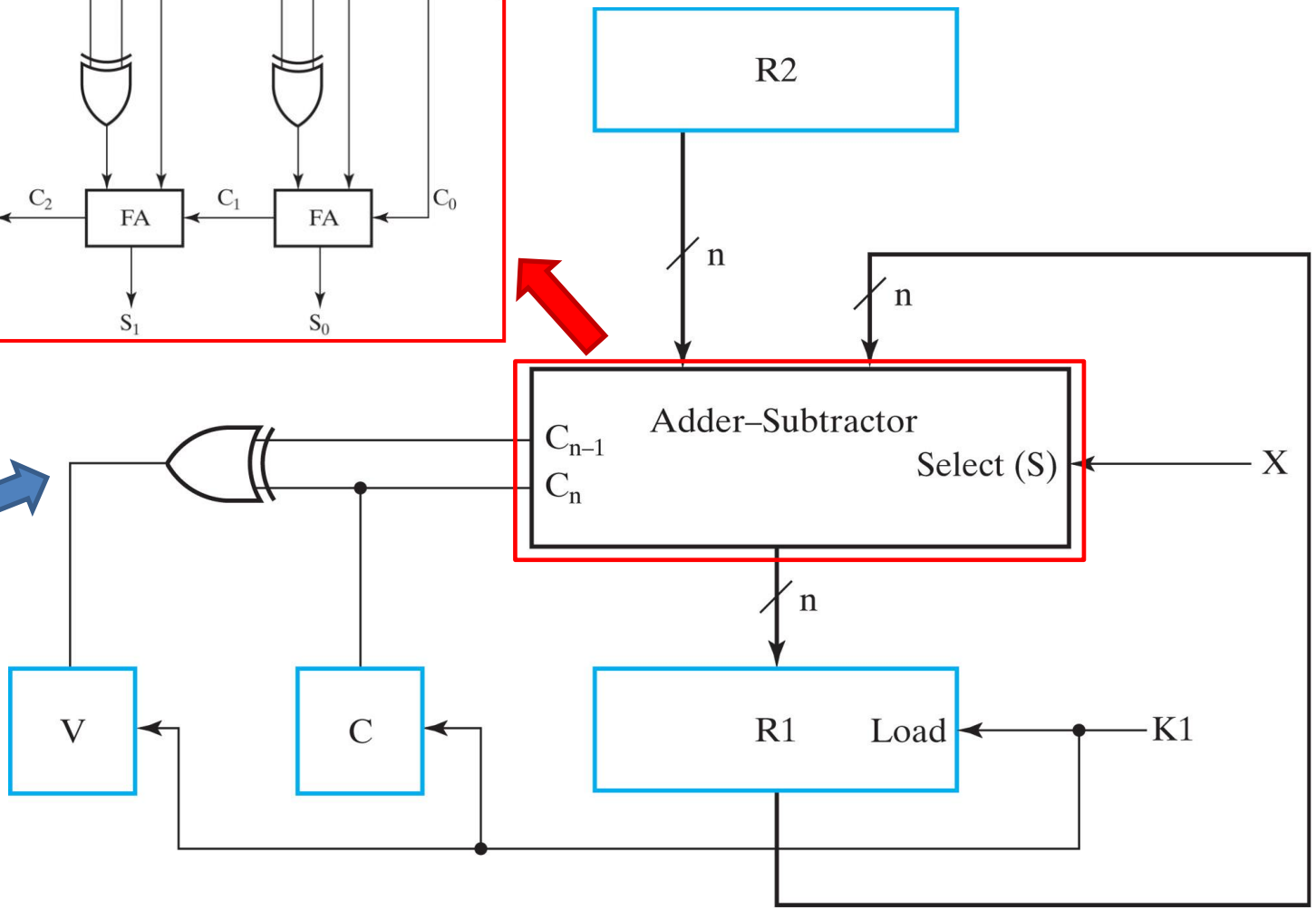
Hardware che realizza la somma o sottrazione tra i dati presenti nei registri R1 e R2



Richiamo: adder-subtractor a n bit



Hardware che realizza la somma o sottrazione tra i dati presenti nei registri R1 e R2



Microoperazioni aritmetiche: esempio

$\overline{\mathbf{X}}\mathbf{K_1}$: $\mathbf{R1} \leftarrow \mathbf{R1} + \mathbf{R2}$

$\mathbf{X}\mathbf{K_1}$: $\mathbf{R1} \leftarrow \mathbf{R1} + \overline{\mathbf{R2}} + \mathbf{1}$

K_1: segnale di controllo che avvia l'esecuzione dell'operazione

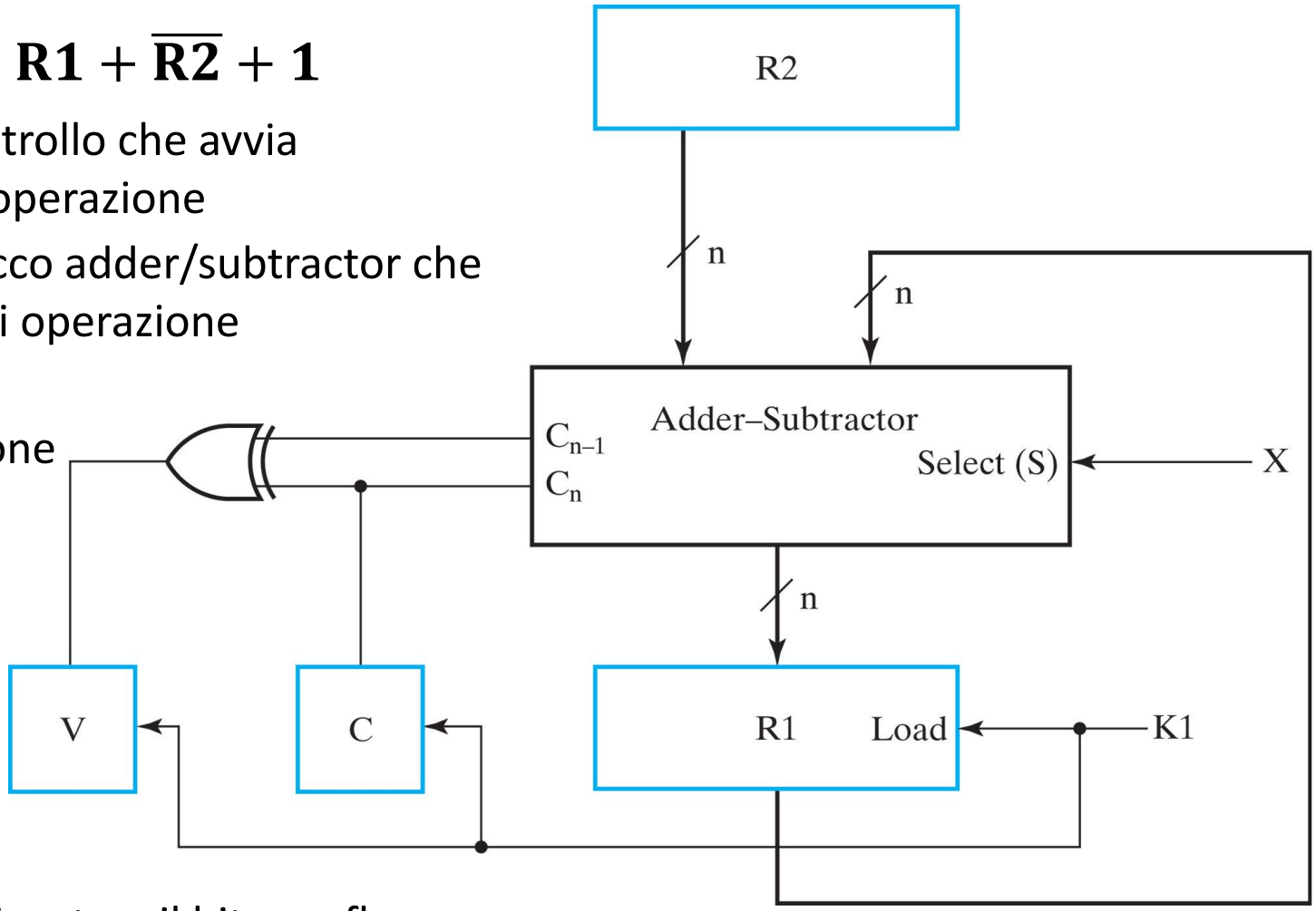
X: input S del blocco adder/subtractor che seleziona il tipo di operazione

X = 0: somma

X = 1: sottrazione

Ad ogni fronte di salita del clock in cui K_1 vale 1, la somma o differenza tra R1 e R2 viene caricata in R1.

Il bit di uscita di riporto e il bit overflow sono trasferiti ai relativi flip-flop quando $K_1 = 1$



Microoperazioni logiche

- Servono a manipolare i bit contenuti nei registri, sono tutte operazioni bit a bit (**bitwise**), cioè considerano separatamente ogni bit del registro come singola variabile binaria
- **Operazioni logiche di base:** NOT ($\overline{x}$), AND ($\wedge$), OR (V), XOR ($\oplus$)

Symbolic Designation	Description
$R0 \leftarrow \overline{R1}$	Logical bitwise NOT (1s complement)
$R0 \leftarrow R1 \wedge R2$	Logical bitwise AND (clears bits)
$R0 \leftarrow R1 \vee R2$	Logical bitwise OR (sets bits)
$R0 \leftarrow R1 \oplus R2$	Logical bitwise XOR (complements bits)

NB: il simbolo + può indicare addizione o OR logico, va evinto dal contesto

Esempio $K_1 + K_2$: $R1 \leftarrow R2 + R3$, $R4 \leftarrow R2 \text{ V } R3$

- Se è all'interno di una funzione booleana, + significa OR
- altrimenti + sta per somma

Microoperazioni logiche

- Possono essere usate anche per cambiare il valore di alcuni bit, azzerare o rimuovere alcuni bit all'interno di un registro
  - **AND**: può essere usata per porre a 0 (**clear**) un gruppo di bit di un registro
  - **OR**: può essere usata per porre a 1 (**set**) un gruppo di bit di un registro
  - **XOR**: può essere usata per fare il **complemento** di un gruppo di bit di un registro
- Si definisce maschera (**mask**) un gruppo di bit con cui eseguire un'operazione logica al fine di ottenere il cambiamento desiderato in uno o più bit del registro

Microoperazioni logiche: Clear

- **Clear**: **AND** tra il registro e una maschera avente ‘0’ in corrispondenza ai bit del registro che vogliamo azzerare e ‘1’ in corrispondenza ai bit che vogliamo lasciare invariati

$$X \wedge 0 = 0, \quad X \wedge 1 = X$$

Esempio:

10101101 10101011	R1	(data)
00000000 11111111	R2	(mask)
00000000 10101011	**R1 ← R1 ∧ R2**	

Microoperazioni logiche: Set

- **Set**: **OR** tra il registro e una maschera avente ‘1’ in corrispondenza ai bit del registro che vogliamo settare (porre a 1) e ‘0’ in corrispondenza ai bit che vogliamo lasciare invariati

$$X \vee 1 = 1, \quad X \vee 0 = X$$

Esempio:

10101101 10101011	R1	(data)
11111111 00000000	R2	(mask)
11111111 10101011	**R1 ← R1 V R2**	

Microoperazioni logiche: Complemento

- **Complemento**: **XOR** tra il registro e una maschera avente ‘1’ in corrispondenza ai bit del registro che vogliamo complementare e ‘0’ in corrispondenza ai bit che vogliamo lasciare invariati

$$X \oplus 1 = \overline{X}, \quad X \oplus 0 = X$$

Esempio:

10101101 10101011	R1	(data)
11111111 00000000	R2	(mask)
01010010 10101011	$\mathbf{R1 \leftarrow R1 \oplus R2}$	

Microoperazioni di scorrimento (shift)

- Sono usate per **spostare lateralmente i bit di un registro**
  - **Shift a sinistra** (sl): sposta il contenuto di un registro verso il MSB. Il bit più a dx del registro di destinazione è detto bit entrante, il bit più a sx del registro sorgente è detto bit uscente
  - **Shift a destra** (sr): sposta il contenuto di un registro verso il LSB. Il bit più a sx del registro di destinazione è detto bit entrante, il bit più a dx del registro sorgente è detto bit uscente
  - Con questo tipo di shift, il bit entrante è '0' e il bit uscente viene scartato

		Eight-Bit Examples	
Type	**Symbolic Designation**	**Source *R2***	**After Shift: Destination *R1***
Shift left	$R1 \leftarrow \text{sl } R2$	10011110	00111100
Shift right	$R1 \leftarrow \text{sr } R2$	11100101	01110010

Riepilogo

- Abbiamo definito due tra i blocchi più usati nei sistemi digitali: **registri** e **contatori**
- Nei registri il controllo del caricamento parallelo dei dati può essere realizzato con diverse tecniche, **clock gating** e **load enable**
- **Control unit** e **datapath** sono i due blocchi in cui può essere partizionato un sistema digitale: il datapath esegue le operazioni, la control unit coordina la sequenza di operazioni
- Le operazioni elementari sui bit di un registro vengono dette **microoperazioni**, espresse in **RTL**, possono essere di 4 tipi
  - Microoperazioni di **trasferimento**
  - Microoperazioni di **aritmetiche**
  - Microoperazioni **logiche**
  - Microoperazioni di **scorrimento** (shift)

Disclaimer

Figures from *Logic and Computer Design Fundamentals*, Fifth Edition, GE Mano | Kime | Martin