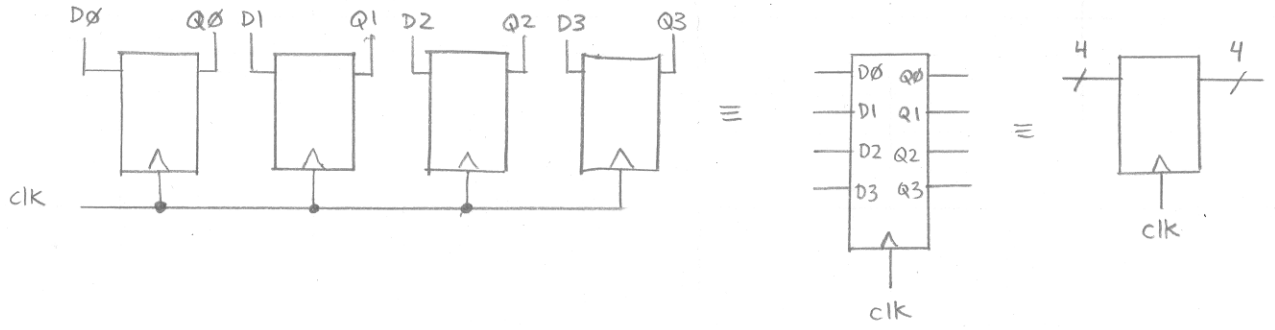
12.1 Registers

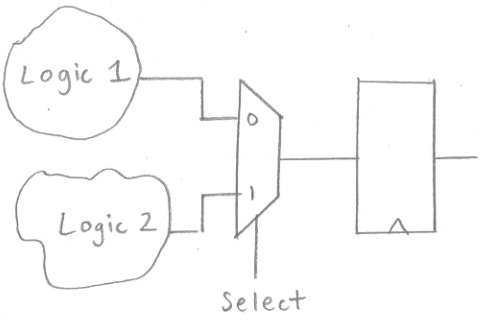
Def: A register is a set of flip-flops with a common clock signal

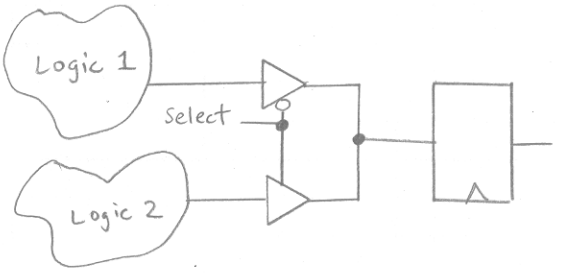


Registers are fundamental blocks in any digital system or processor:

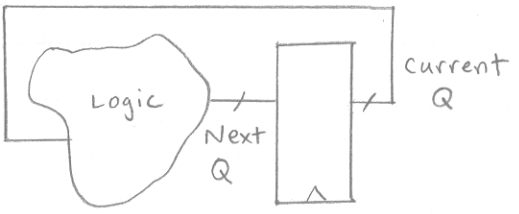
Ex: Pipelining to increase throughput

Ex: Storing values from muxes or tri-state buses



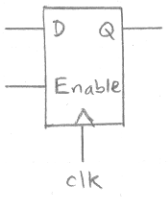


Ex: Implementing feedback → basis of finite-state machines (FSMs)



"Enable-able Registers" or "Registers with an Enable Signal"

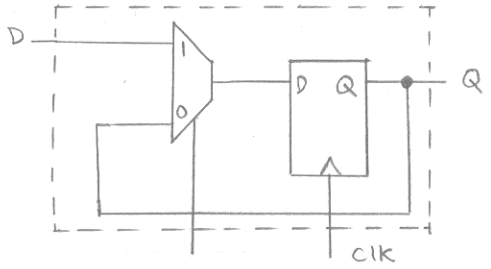
A signal (in addition to clock) determines when register output can change - analogous to a gated latch



D	Enable	Clk	Q+
X	0	↑	Q
0	1	↑	0
1	1	↑	1

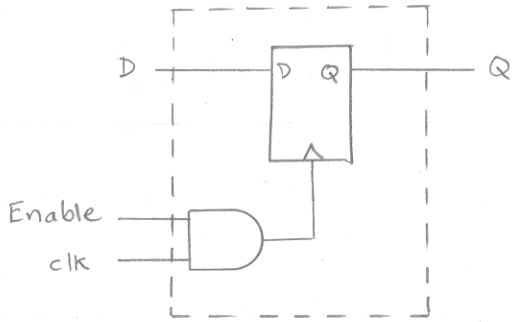
(hold, update of register was not enabled)

Implementation: Use a D flip-flop and a 2-to-1 mux



(Book refers to this as "CE", "clock enable")

Alternative Implementation: Use a D flip-flop and an AND gate



Why is this bad?

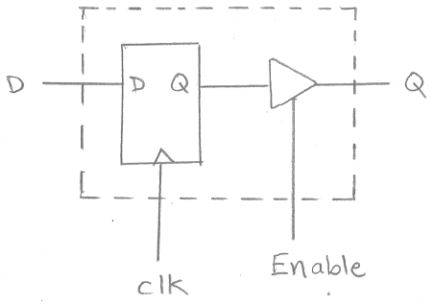
Glitches (e.g., static 0 hazard) on Enable could cause the D flip-flop to sample D by mistake, at some time other than the rising edge of clock!

Registers With Tri-State Outputs

Same as a D flip-flop except output is high impedance when disabled:

D	Enable	clk	Q+
X	0	↑	Hi Z, high impedance, floating, etc.
0	1	↑	0
1	1	↑	1

Implementation: D flip-flop and tri-state buffer



Many possible variations/combinations, e.g. active low or active high enables, with or without tristate outputs → always check data sheet to understand correct operation.

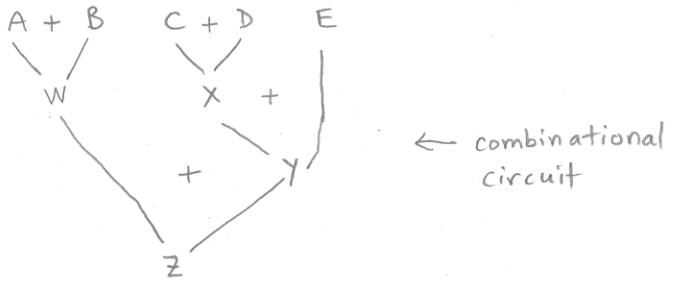
Accumulators

It is often necesseray to keep a running total of a set of numbers in digital applications → store current sum in a register and add next input to it (accumulate the numbers into a total).

Ex: $Z = A+B+C+D+E$

1.) Implement using intermediate sums and a "tree" of adder circuits:

$W = A+B$
$X = C+D$
$Y = X+E$
$Z = W+Y$

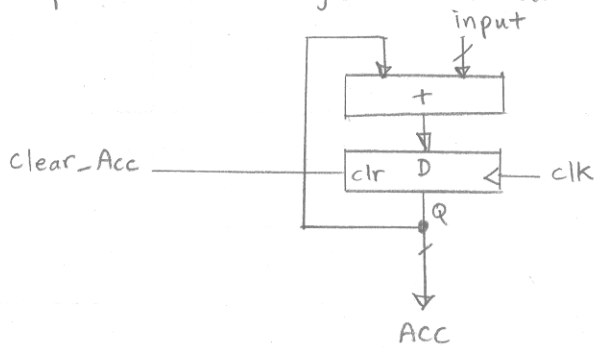


2.) Do sum sequentially using accumulator

- Clear Accumulator (ACC = 0)

$ACC = ACC + A$ ⇒ Abbreviate $ACC += A$
$ACC = ACC + B$ $ACC += B$
$ACC = ACC + C$ ⋮
$ACC = ACC + D$
$ACC = ACC + E$ $ACC += E$

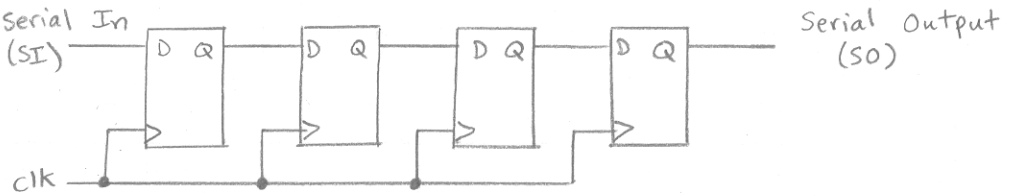
Implementation: register and adder



Register allows the use of feedback.

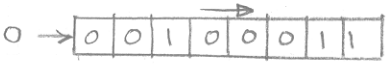
12.2 Shift Registers

By connecting flip-flops in series, we can shift bits around in a digital word:

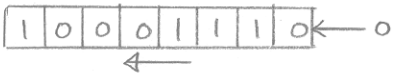


Original Data 0 1 0 0 0 1 1 1

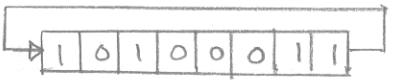
Shift Right $\underline{=}$ 0 0 1 0 0 0 1 1



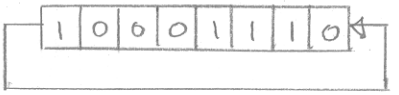
Shift Left 1 0 0 0 1 1 1 $\underline{=}$ 0



Rotate Right (Right Circular Shift)



Rotate Left (Left Circular Shift)



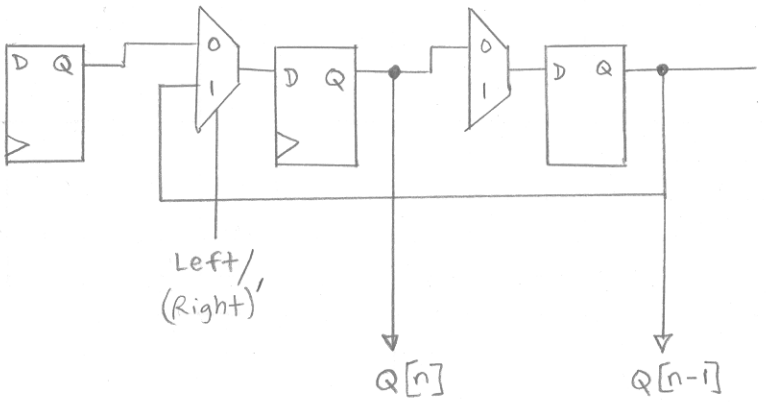
Changes take place on active edge of the clock (either ↑ or ↓)

Multiple combinations of serial/parallel inputs and outputs:

Outputs \ Inputs	Serial	Parallel
Serial	X	X
Parallel	X	X

Accumulator example

Ex: Implement a serial-in parallel-out shift register which can shift left or right



12.3 Binary Counters

State: the set of values (0 or 1) stored in a flip-flop, latch, register, or memory

Counter: a circuit which cycles through a fixed sequence of states

Synchronous counters change state based on a common clock pulse (all flip-flops within the register change values simultaneously).

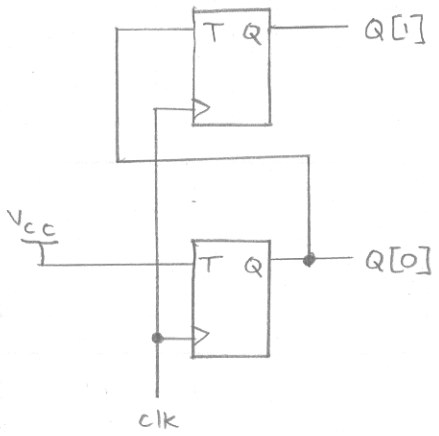
Ex: Simple 2-bit binary counter

State sequence: 00, 01, 10, 11, 00, 01, ...

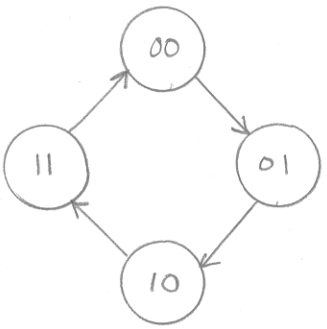
Note that the counter "wraps around" or returns to its initial value after reaching its maximum

The LSB switches value every cycle, so we can implement it using a T flip-flop with the input set high.

The MSB switches whenever the LSB = 1, so we use the LSB as the T input to a second T flip-flop:



We can also describe counter operation using a state graph (or state transition diagram).



General Counter Design

A general approach to designing arbitrary counters is:

1. Write the desired state sequence/state graph
2. For each present state, write corresponding next state(s)
3. Determine needed FF inputs required to make transitions
4. Use combinational logic minimization techniques to find necessary logic for each flip-flop input

Ex: 4 bit counter, increments by 3 each cycle, at 15, it wraps around to ∅

Present State	Next State	D Flip-Flop Input	T Flip-Flop Input
$D\ C\ B\ A$	$D^+\ C^+\ B^+\ A^+$	$D_D\ C_D\ B_D\ A_D$	$D_T\ C_T\ B_T\ A_T$
0 0 0 0	0 0 1 1	0 0 1 1	0 0 1 1
0 0 1 1	0 1 1 0	0 1 1 0	0 1 0 1
0 1 1 0	1 0 0 1	1 0 0 1	1 1 1 1
1 0 0 1	1 1 0 0	1 1 0 0	0 1 0 1
1 1 0 0	1 1 1 1	1 1 1 1	0 0 1 1
1 1 1 1	0 0 0 0	0 0 0 0	1 1 1 1

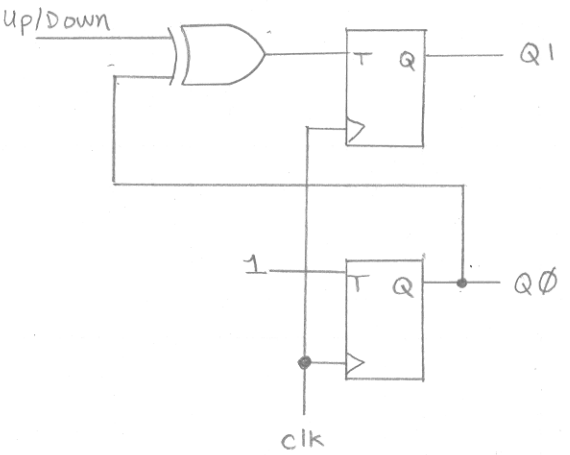
If there are additional inputs (e.g. up/down, load, etc.), we can follow one of two approaches to design:

1.) List multiple next states (each a function of the extra input(s))
2.) Consider all inputs the same: present state and additional inputs

Ex: 2 bit up/down counter using T flip-flops

Up/Down	Present State		Next State		T Flip-Flop Inputs	
	$Q1$	$Q\phi$	$Q1^+$	$Q\phi^+$	$T1$	$T\phi$
0	0	0	0	1	0	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	0	0	1	1
1	0	0	1	1	1	1
1	1	1	1	0	0	1
1	1	0	0	1	1	1
1	0	1	0	0	0	1

Input Equations: $T\phi = 1$ $T1 = Q\phi \oplus Up/Down$



Alternative approach:

Present State		Next State (Up/Down = ϕ)		Next State (Up/Down = 1)	
$Q1$	$Q\phi$	$Q1^+$	$Q\phi^+$	$Q1^+$	$Q\phi^+$
0	0	0	1	1	1
0	1	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0