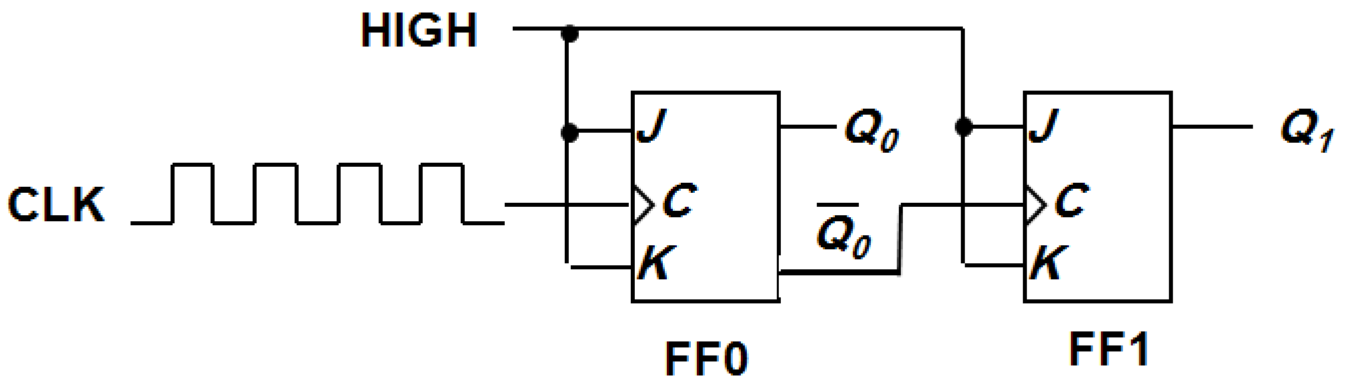
College of Computer and Information Sciences
Department of Computer Science

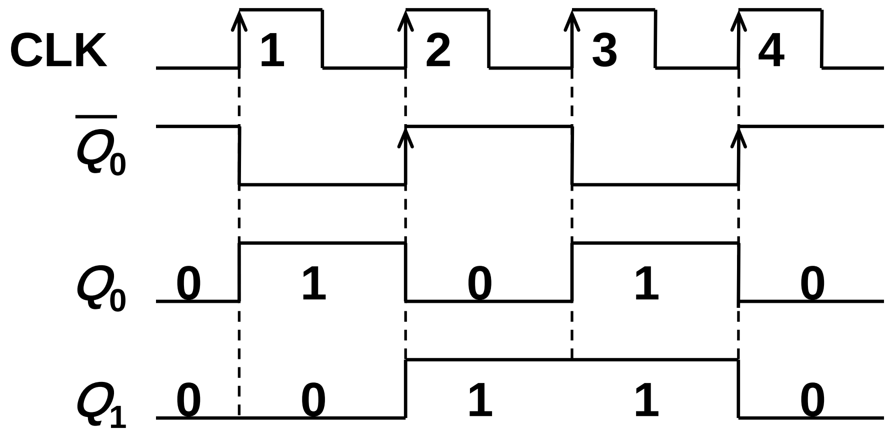
CSC 220: Computer Organization

Unit 8
Counters, RAM

Asynchronous (Ripple) Counters

- Example: 2-bit ripple binary counter.
- Output of one flip-flop is connected to the clock input of the next more-significant flip-flop.



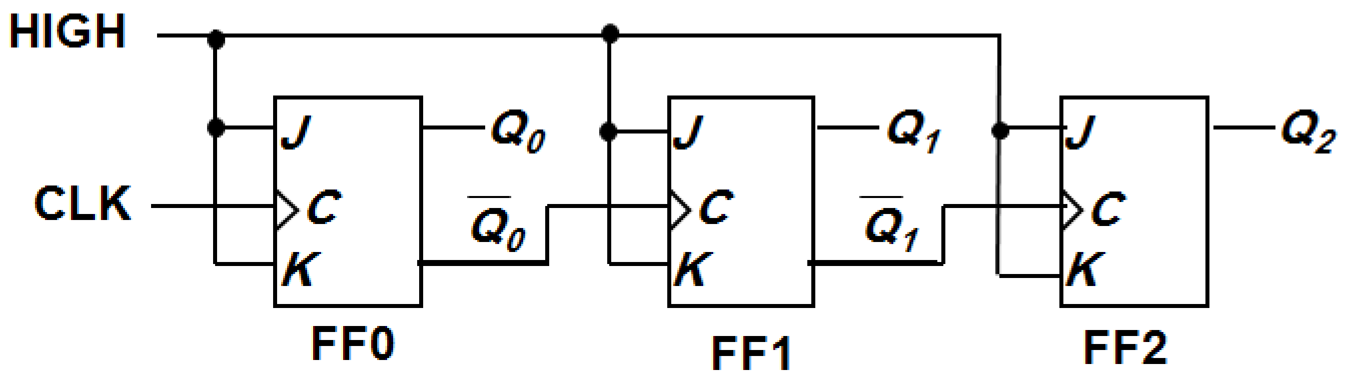


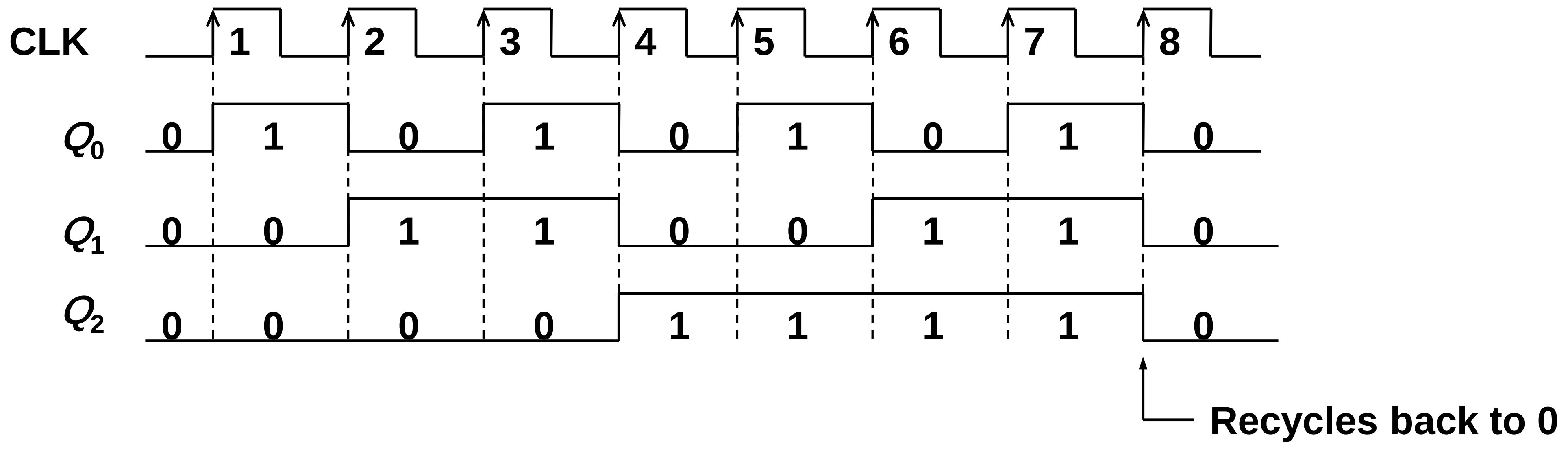
Timing diagram

$00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00 \ldots$

Asynchronous (Ripple) Counters

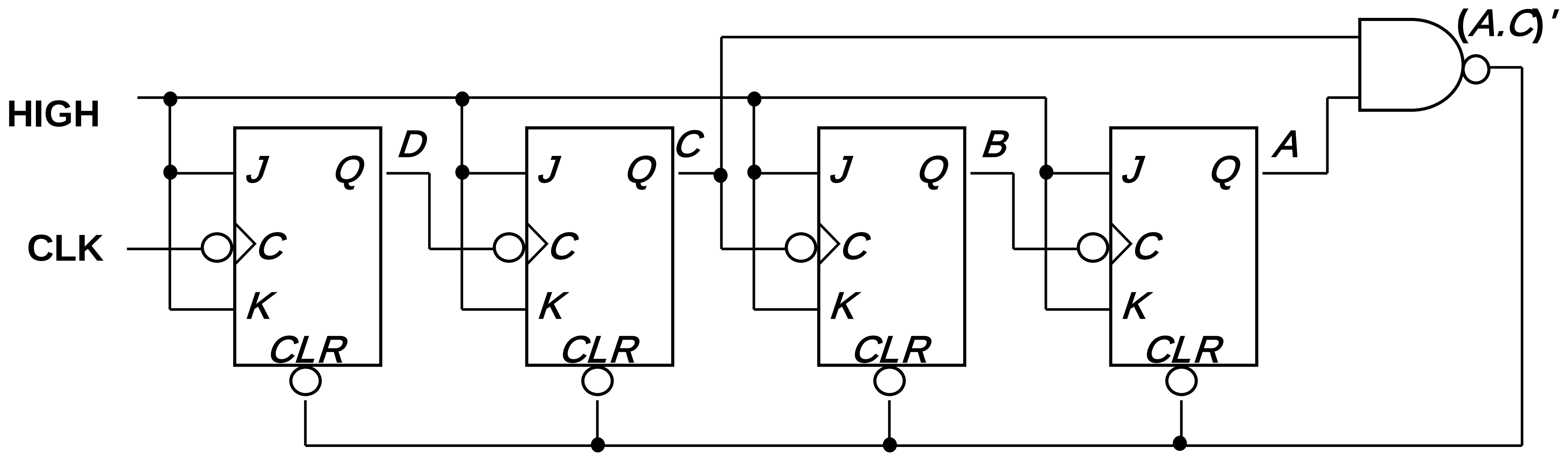
- Example: 3-bit ripple binary counter.





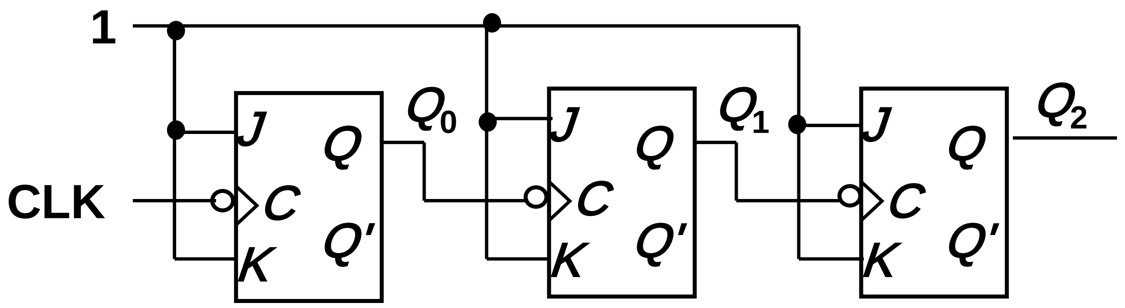
Asyn. Counters with MOD no. < 2^n

- Decade counters (or BCD counters) are counters with 10 states (modulus-10) in their sequence. They are commonly used in daily life (e.g.: utility meters, odometers, etc.).
- Design an asynchronous decade counter.

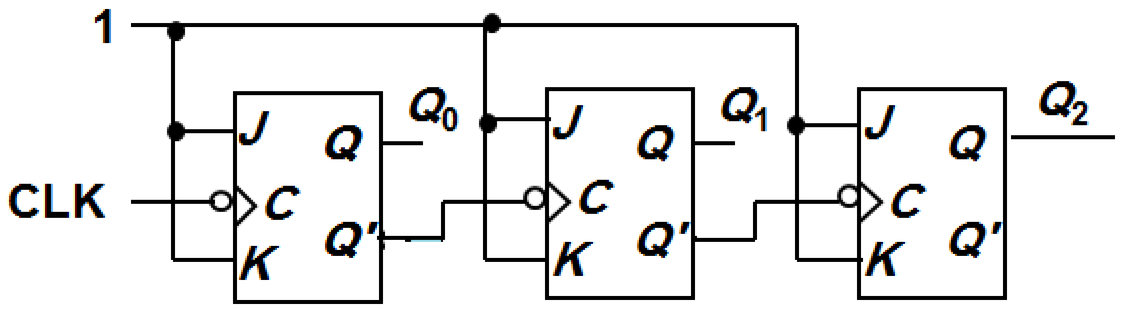


Asynchronous Down Counters

- So far we are dealing with *up counters*. *Down counters*, on the other hand, count downward from a maximum value to zero, and repeat.
- Example: A 3-bit binary (MOD-2^3) down counter.



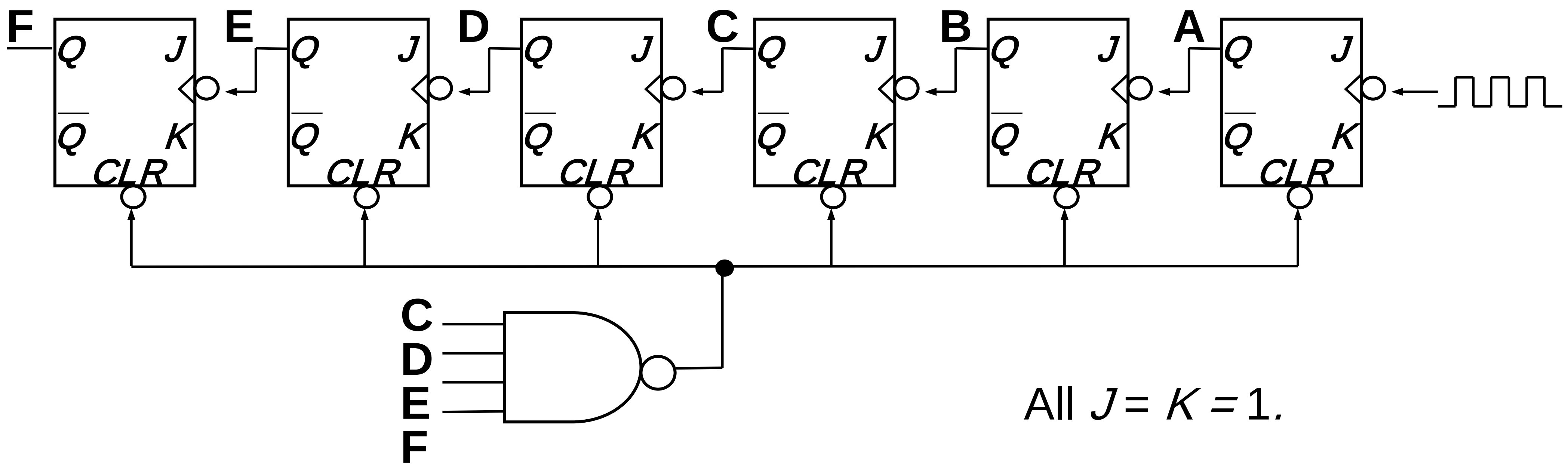
3-bit binary up counter



3-bit binary down counter

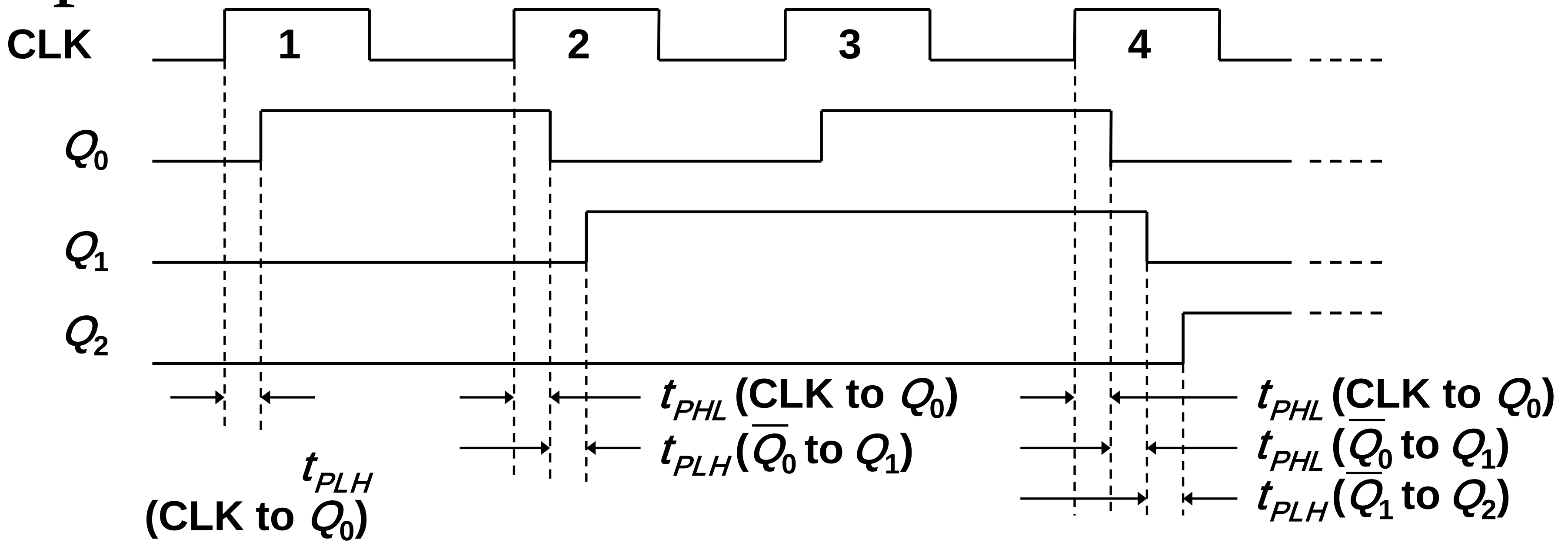
Asyn. Counters with MOD no. < 2^n

- *Exercise:* How to construct an asynchronous MOD-5 counter? MOD-7 counter? MOD-12 counter?
- *Question:* The following is a MOD-? counter?



Asynchronous (Ripple) Counters

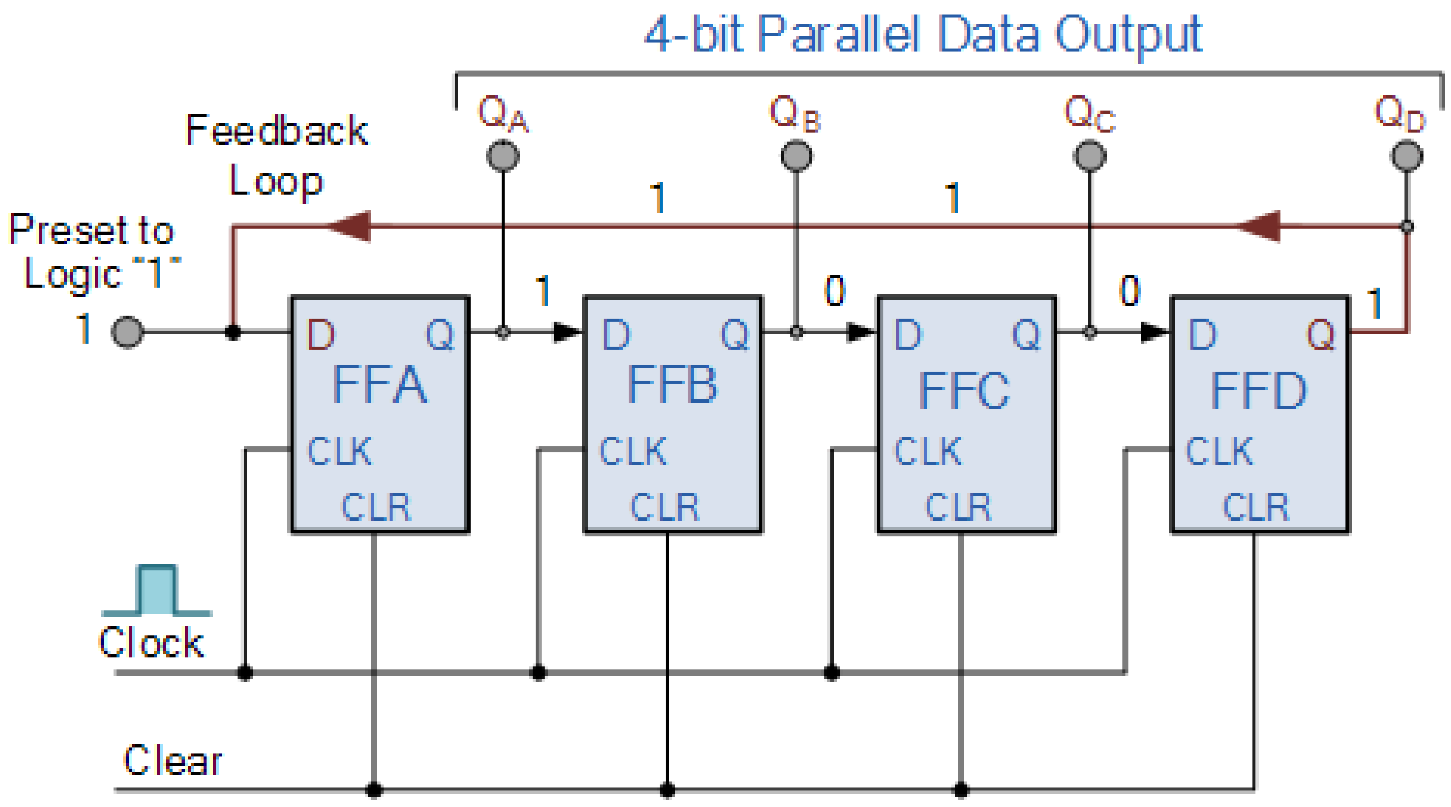
- Propagation delays in an asynchronous (ripple-clocked) binary counter.
- If the accumulated delay is greater than the clock pulse, some counter states may be misrepresented!



CLK
1
2
3
4
Q_0
Q_1
Q_2
t_{PLH} (CLK to Q_0)
t_{PHL} (CLK to Q_0)
t_{PLH} ($\overline{Q_0}$ to Q_1)
t_{PHL} (CLK to Q_0)
t_{PHL} ($\overline{Q_0}$ to Q_1)
t_{PLH} ($\overline{Q_1}$ to Q_2)

4-bit Ring Counter

A "CLEAR" signal is firstly applied to all the flip-flops together in order to "RESET" their outputs to a logic "0" level and then a "PRESET" pulse is applied to the input of the first flip-flop (FFA) before the clock pulses are applied. This then places a single logic "1" value into the circuit of the ring counter.

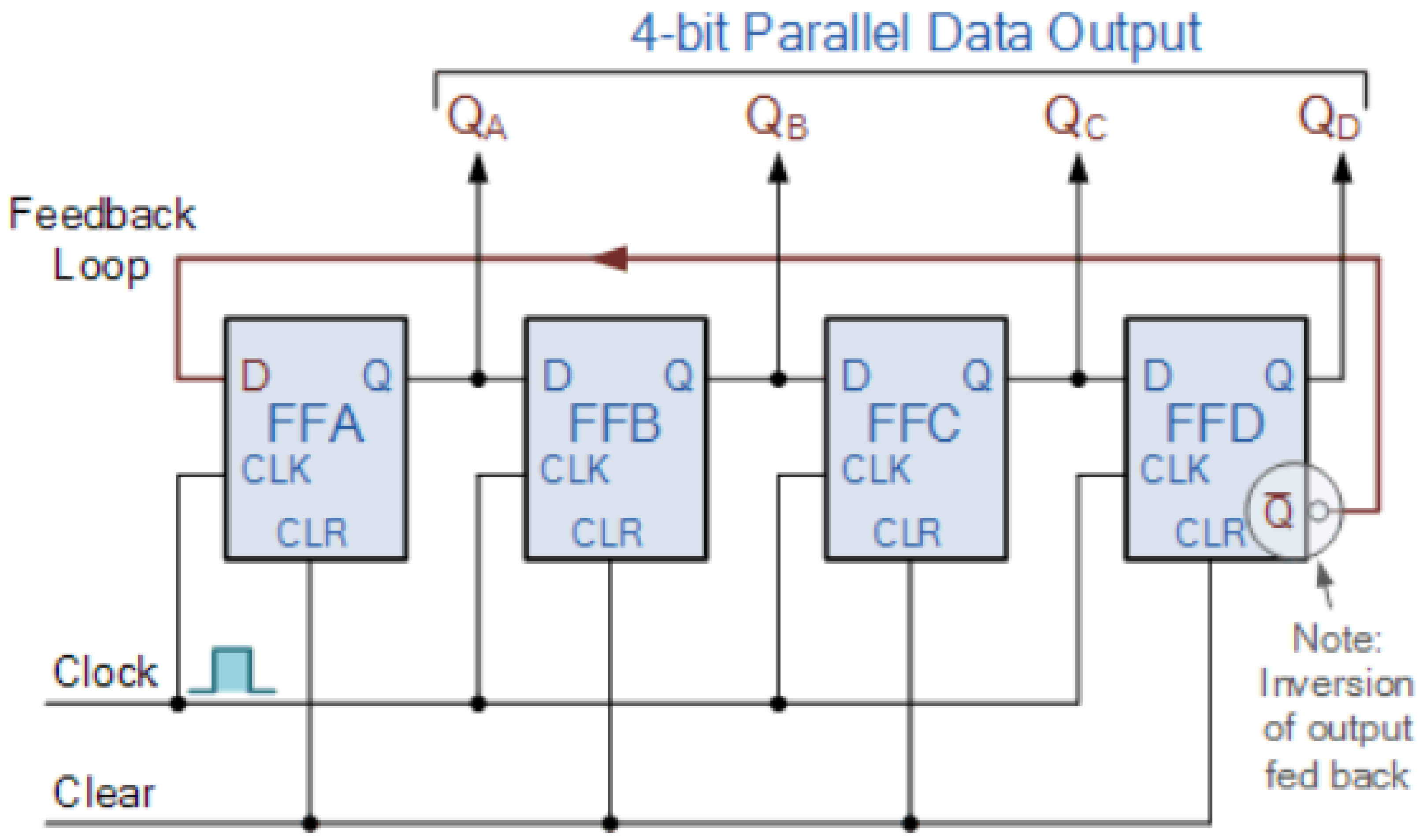


So on each successive clock pulse, the counter circulates the same data bit between the four flip-flops over and over again around the "ring" every fourth clock cycle. But in order to cycle the data correctly around the counter we must first "load" the counter with a suitable data pattern as all logic "0's" or all logic "1's" outputted at each clock cycle would make the ring counter invalid.

Johnson Ring Counter

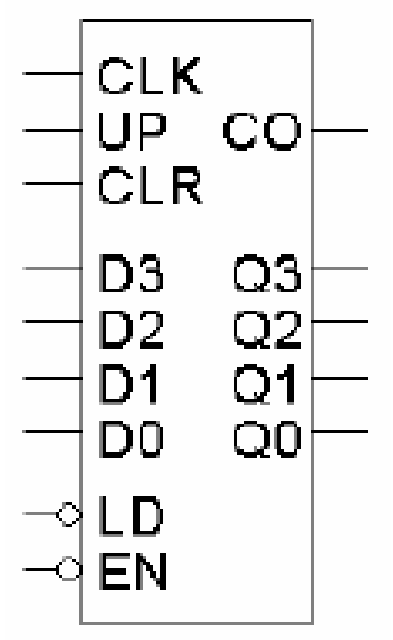
The **Johnson Ring Counter** or “Twisted Ring Counters”, is another shift register with feedback exactly the same as the standard *Ring Counter* above, except that this time the inverted output Q of the last flip-flop is now connected back to the input D of the first flip-flop as shown below.

4-bit Johnson Ring Counter



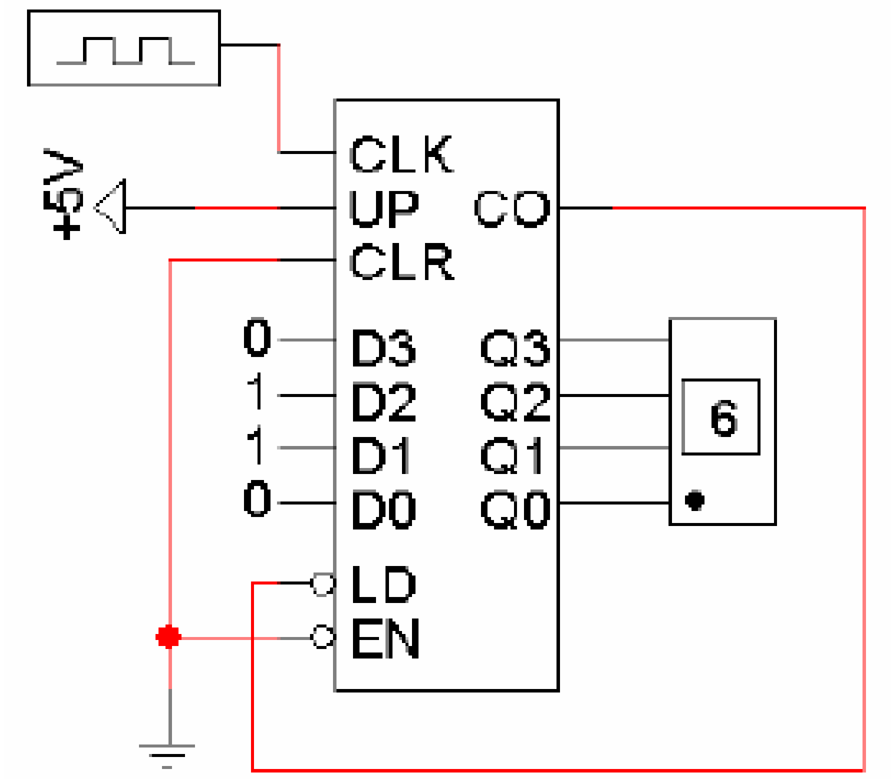
More complex counters

- More complex counters are also possible. The full-featured LogicWorks Counter-4 device below has several functions.
  - It can count up or down, depending on whether the UP input is 1 or 0.
  - You can clear the counter to 0000 asynchronously by setting CLR = 1.
  - You can perform a parallel load of D3-D0 when LD = 0.
  - The active-low EN input enables or disables the counter.
  - The "counter out" CO is normally 1, but becomes 0 when the counter reaches its maximum value of 1111 (if UP = 1) or 0000 (if UP = 0).



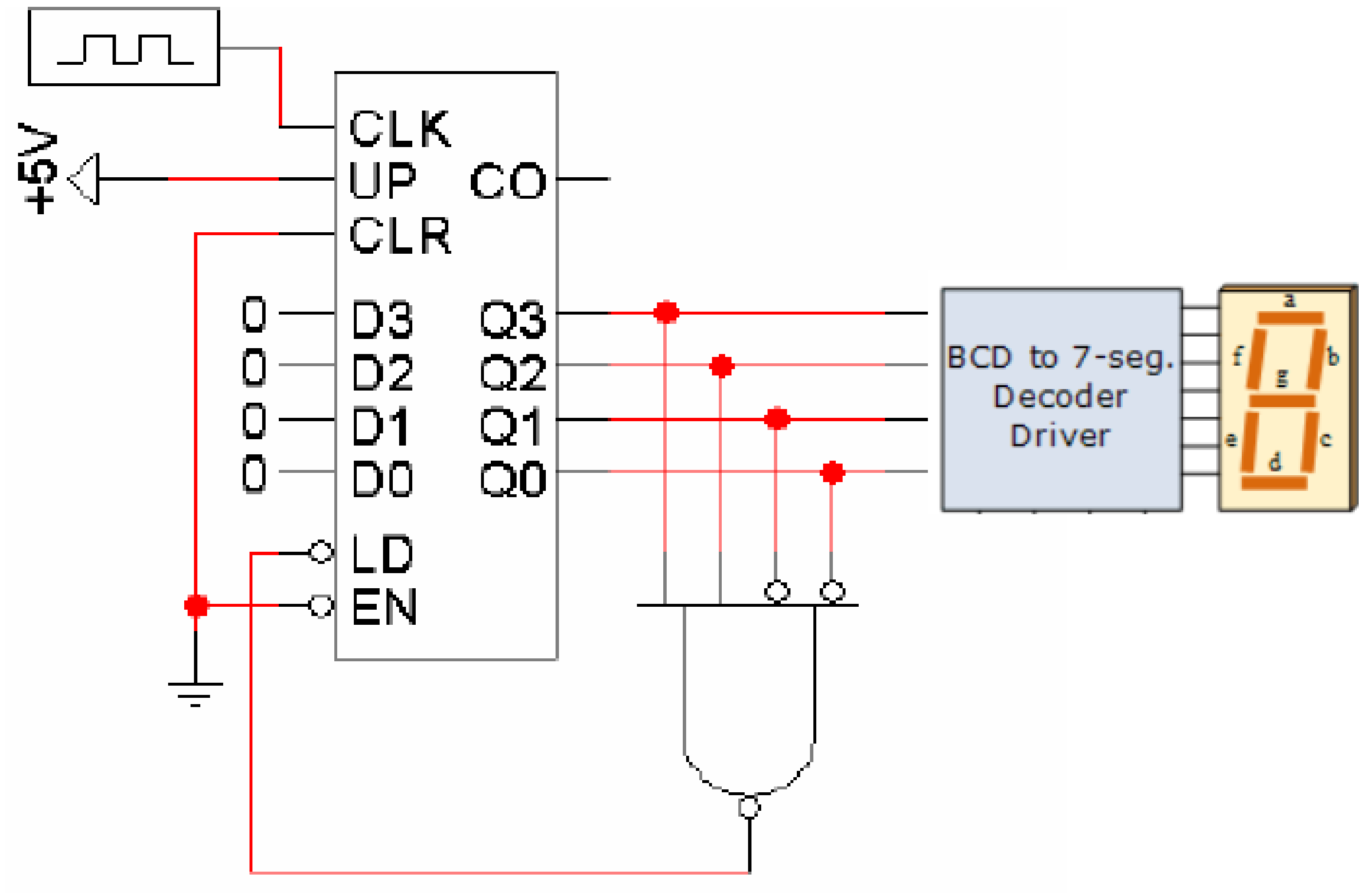
A restricted 4-bit counter

- We can also make a counter that "starts" at some value besides 0000.
- In the diagram below, when CO = 0 the LD signal forces the next state to be loaded from D3-D0.
- The result is this counter wraps from 1111 to 0110 (instead of 0000).



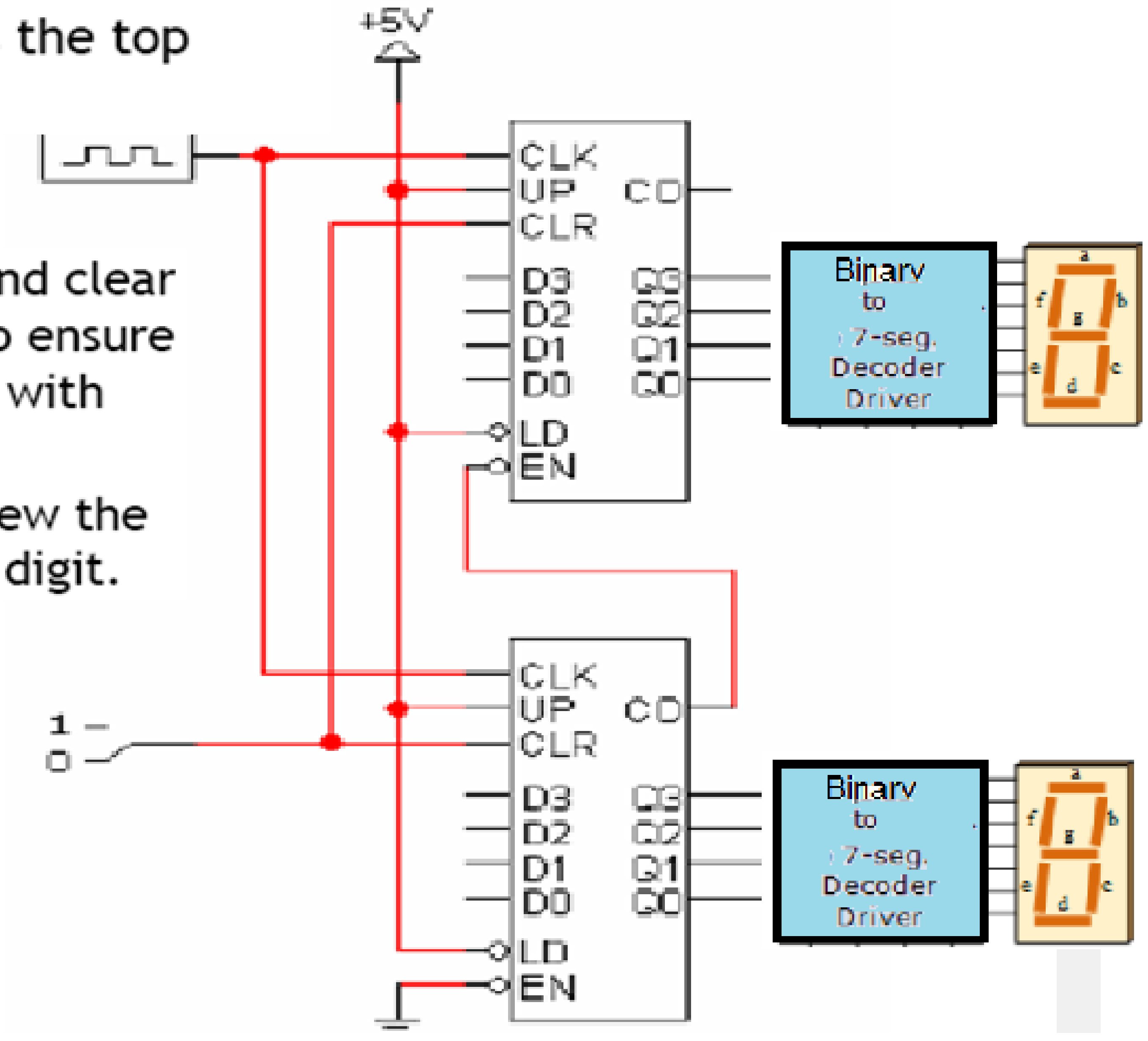
Another restricted counter

- We can also make a circuit that counts up to only 1100, instead of 1111.
- Here, when the counter output reaches 1100, the NAND gate forces the counter to load, so the next state becomes 0000.



Here is an 8-bit counter made from two 4-bit

- The bottom device represents the least significant four bits, while the top counter represents the most significant four bits.
- When the bottom counter reaches 1111 (i.e., when CO = 0), it enables the top counter for one cycle.



The two four-bit counters share clock and clear inputs. Sharing the clock is important to ensure that the two counters are synchronized with respect to each other.

We've used Hex Display units here to view the four-bit output as a single hexadecimal digit.

Introduction to RAM

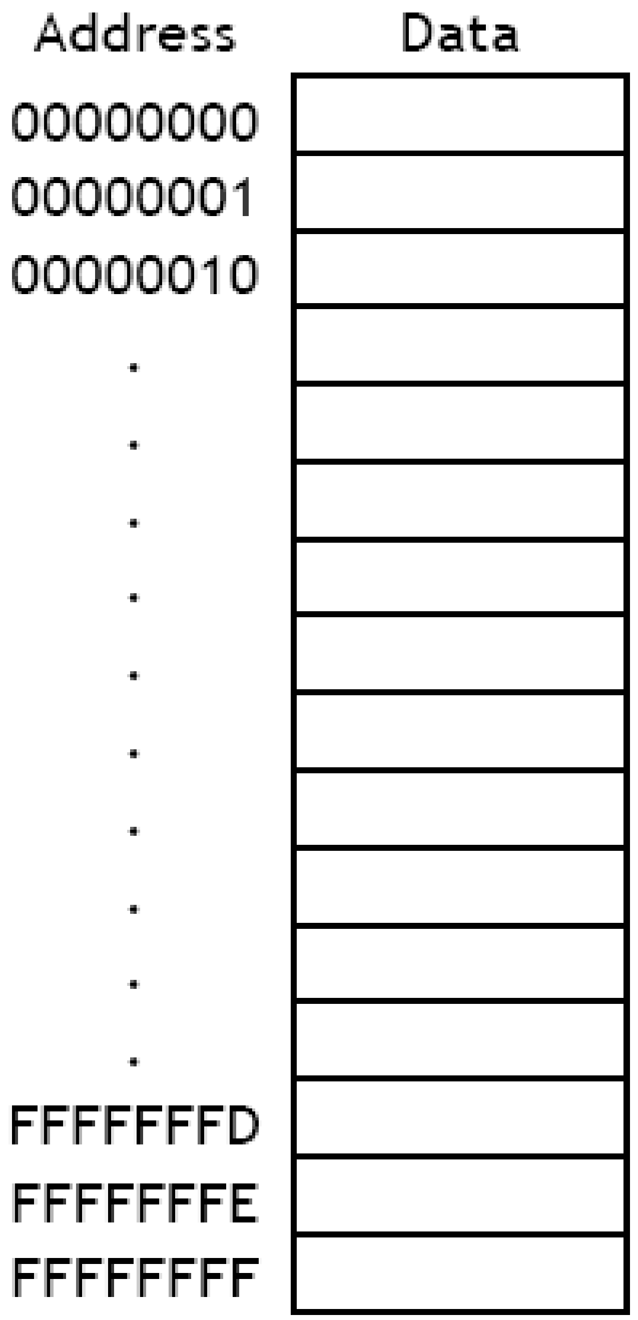
- Random-access memory, or RAM, provides large quantities of temporary storage in a computer system.
- Remember the basic capabilities of a memory.
  - It should be able to store a value.
  - You should be able to read the value that was saved.
  - You should be able to change the stored value.
- A RAM is similar, except that it can store *many* values.
  - An address will specify which memory value we're interested in.
  - Each value can be a multiple-bit word (e.g., 32 bits).
- We'll refine the memory properties as follows.

A RAM should be able to:

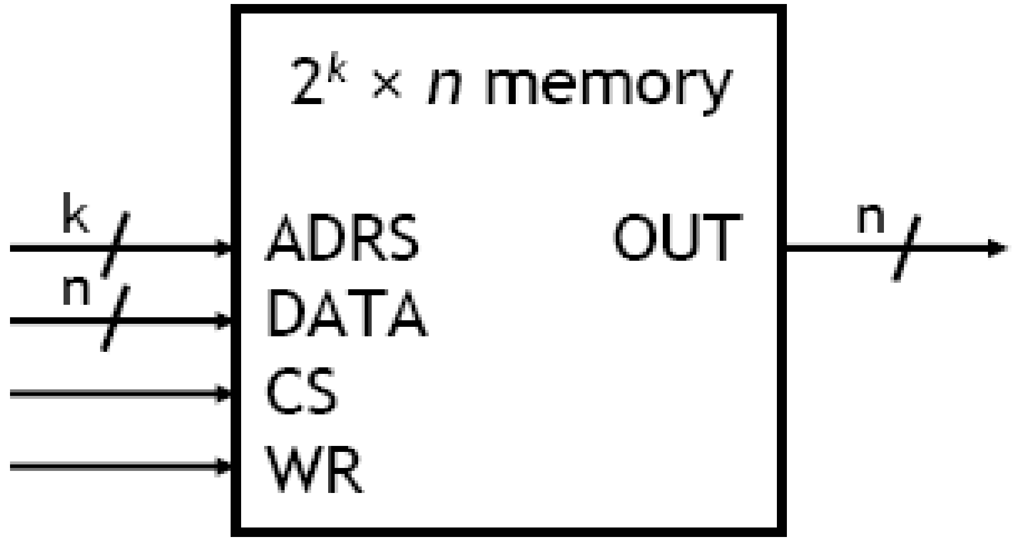
1. Store many words, one per address
2. Read the word that was saved at a particular address
3. Change the word that's saved at a particular address

Picture of memory

- You can think of computer memory as being one big array of data.
  - The address serves as an array index.
  - Each address refers to one word of data.
- You can read or modify the data at any given memory address, just like you can read or modify the contents of an array at any given index.
- If you've worked with pointers in C or C++, then you've already worked with memory addresses.



Block diagram of RAM

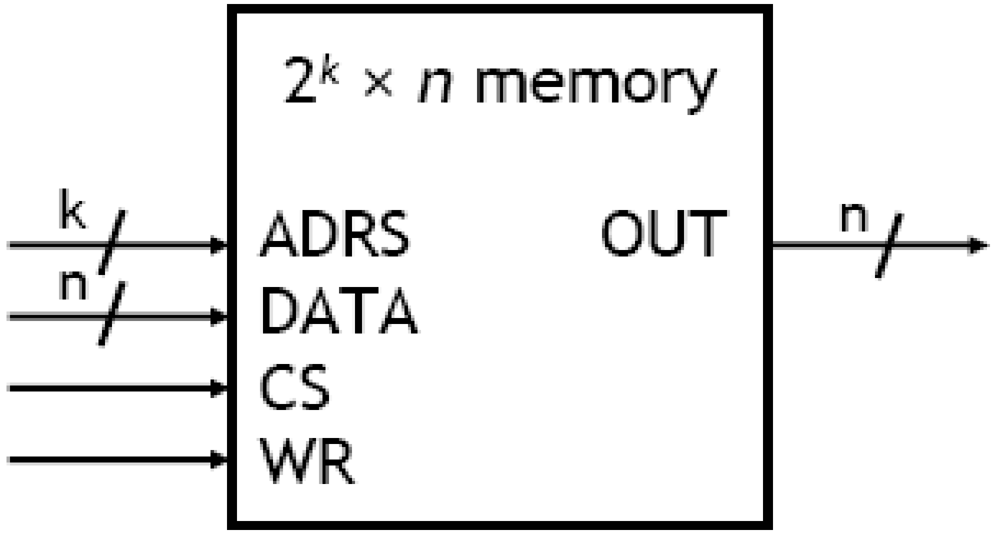


CS	WR	Memory operation
0	x	None
1	0	Read from ADRS
1	1	Write DATA to ADRS

- This block diagram introduces the main interface to RAM.
  - A Chip Select, CS, enables or disables the RAM.
  - ADRS specifies the address or location to read from or write to.
  - WR selects between reading from or writing to the memory.
    - To read from memory, WR should be set to 0.
      OUT will be the *n*-bit value stored at ADRS.
    - To write to memory, we set WR = 1.
      DATA is the *n*-bit value to save in memory.
- This interface makes it easy to combine RAMs together, as we'll see.

Memory sizes

- We refer to this as a $2^k \times n$ memory.
  - There are k address lines, which can specify one of 2^k addresses.
  - Each address contains an n-bit word.



- For example, a $2^{24} \times 16$ RAM contains 2^{24} = 16M words, each 16 bits long.
  - The RAM would need 24 address lines.
  - The total storage capacity is $2^{24} \times 16 = 2^{28}$ bits.

Size matters!

- Memory sizes are usually specified in numbers of bytes (8 bits).
- The 2^{28}-bit memory on the previous page has a capacity of 2^{25} bytes.

$$2^{28} \text{ bits } / \text{ 8 bits per byte} = 2^{25} \text{ bytes}$$

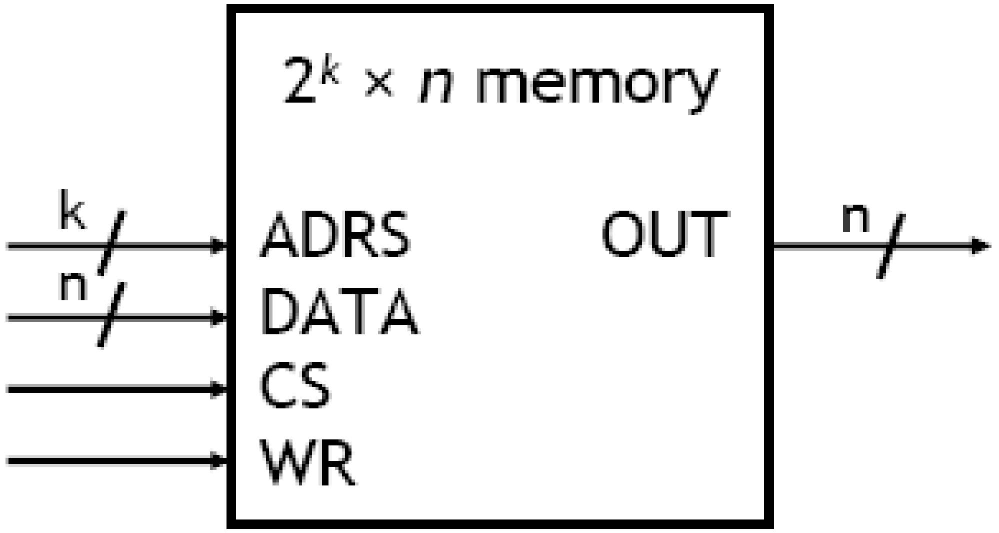
- With the abbreviations below, this is equivalent to 32 megabytes.

$$2^{25} \text{ bytes} = 2^{5} \times 2^{20} \text{ bytes} = 32 \text{ MB}$$

	Prefix	Base 2	Base 10
K	Kilo	2^{10} = 1,024	10^{3} = 1,000
M	Mega	2^{20} = 1,048,576	10^{6} = 1,000,000
G	Giga	2^{30} = 1,073,741,824	10^{9} = 1,000,000,000

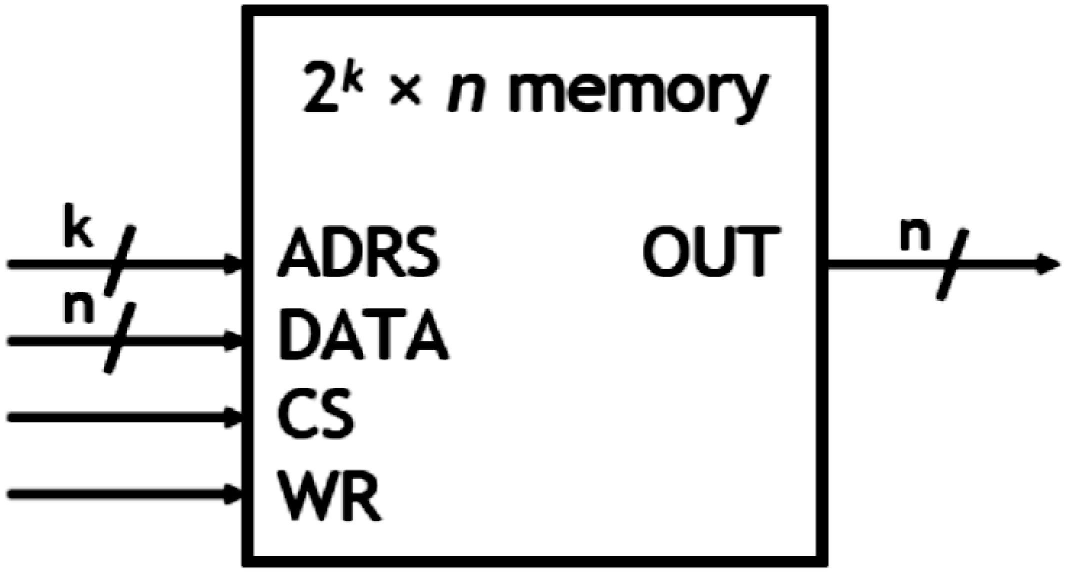
Reading RAM

- To *read* from a RAM, the controlling circuit must take several steps.
  1. Enable the chip by ensuring CS = 1.
  2. Select the read operation, by setting WR = 0.
  3. Send the desired address to the ADRS input.
  4. The contents of that address appear on OUT after a little while.
- Notice that the DATA input is unused for read operations.



Writing RAM

- To *write* to this RAM, you need to do the following tasks.
  1. Enable the chip by setting CS = 1.
  2. Select the write operation, by setting WR = 1.
  3. Send the desired address to the ADRS input.
  4. Send the word to store to the DATA input.
- The output OUT is not needed for memory write operations.


$2^k \times n$ memory
k
n
ADRS
DATA
CS
WR
OUT
n