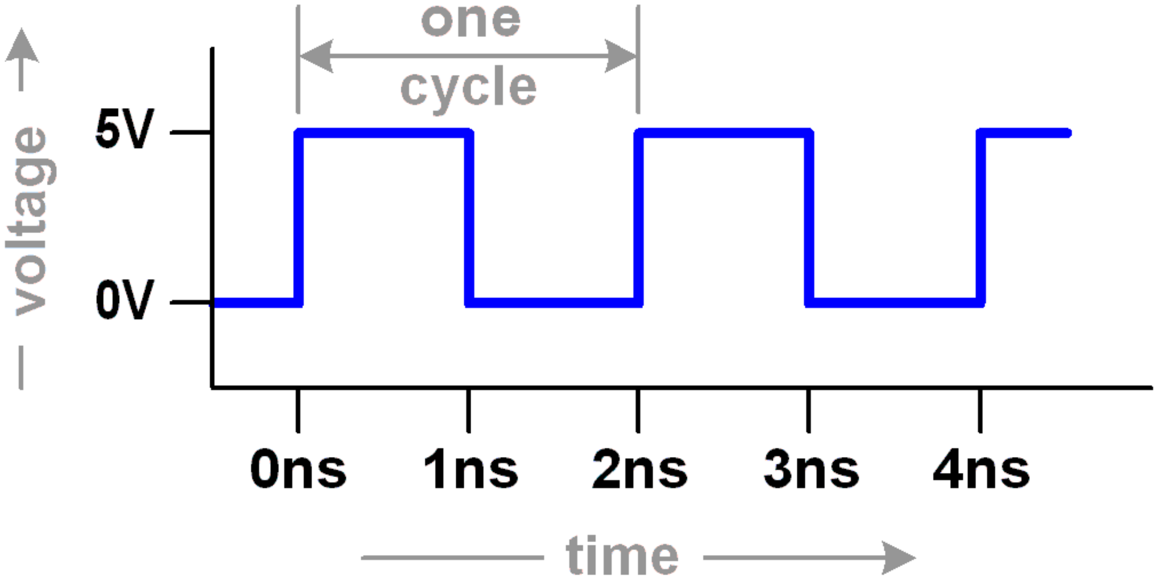
Today’s Topics

- CPU clocks
- Building a “latch” – a 1-bit storage circuit
- Improving the latch
- Edge-triggered vs. level-triggered storage circuits
- Putting flip-flops together: Registers
- Joining outputs using Tri-State Buffers
- Creating a memory circuit using flip-flops
- Organization of memory chips
- Memory types – RAM vs. ROM
- Memory types – Static vs. Dynamic RAM
- Types of ROM memory

A CPU Clock

The “clock” signal used by a CPU has nothing to do with the time of day. It is a master “drumbeat” that’s used to synchronize all of the circuits.

The “clock” signal is a signal that simply turns on and off, on and off over and over again. It has a Boolean value of TRUE, FALSE, TRUE, FALSE.   We can what the clock signal looks like using a timing diagram:

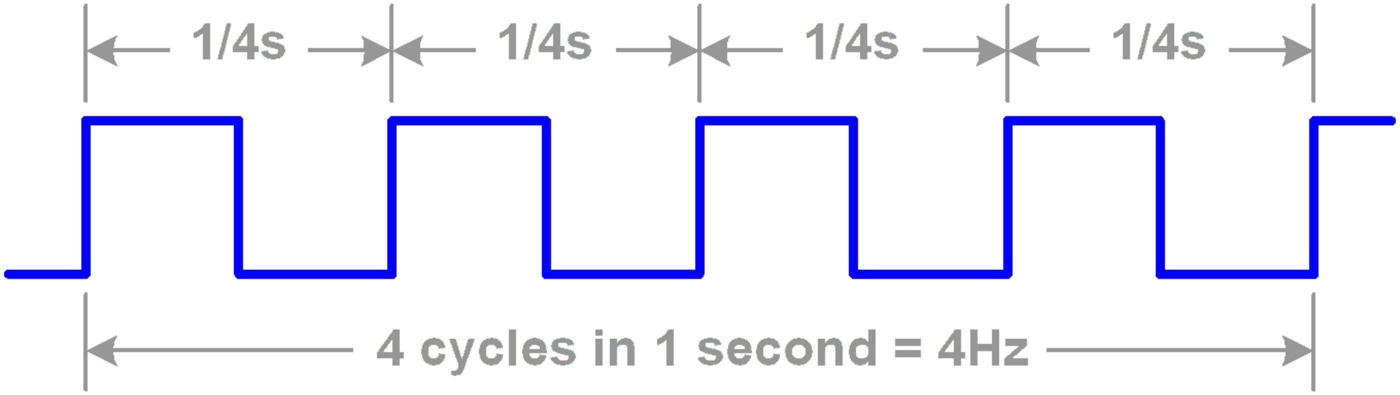


The timing diagram shows the voltage of the signal as time changes: 0V = FALSE, 5V = TRUE (some systems use lower voltages such as 3.3V or 1.8V for TRUE).   In practice, voltages are not labeled on a timing diagram.

As the diagram shows, a clock cycle includes one complete TRUE and FALSE state. The sample clock above has cycles that last 2 nanoseconds.

Clock Speed vs. Cycle Time

The speed of a clock can be described by stating the cycle time (how long a cycle takes) or the frequency (how many clock cycles occur every second). These two measurements have an inverse relationship. For example, consider the following clock:



This clock has cycle time of ¼ second (0.25 seconds). The frequency is 4Hz, because there are four clock cycles every second. Use these formulas:

cycle time = 1 / frequency

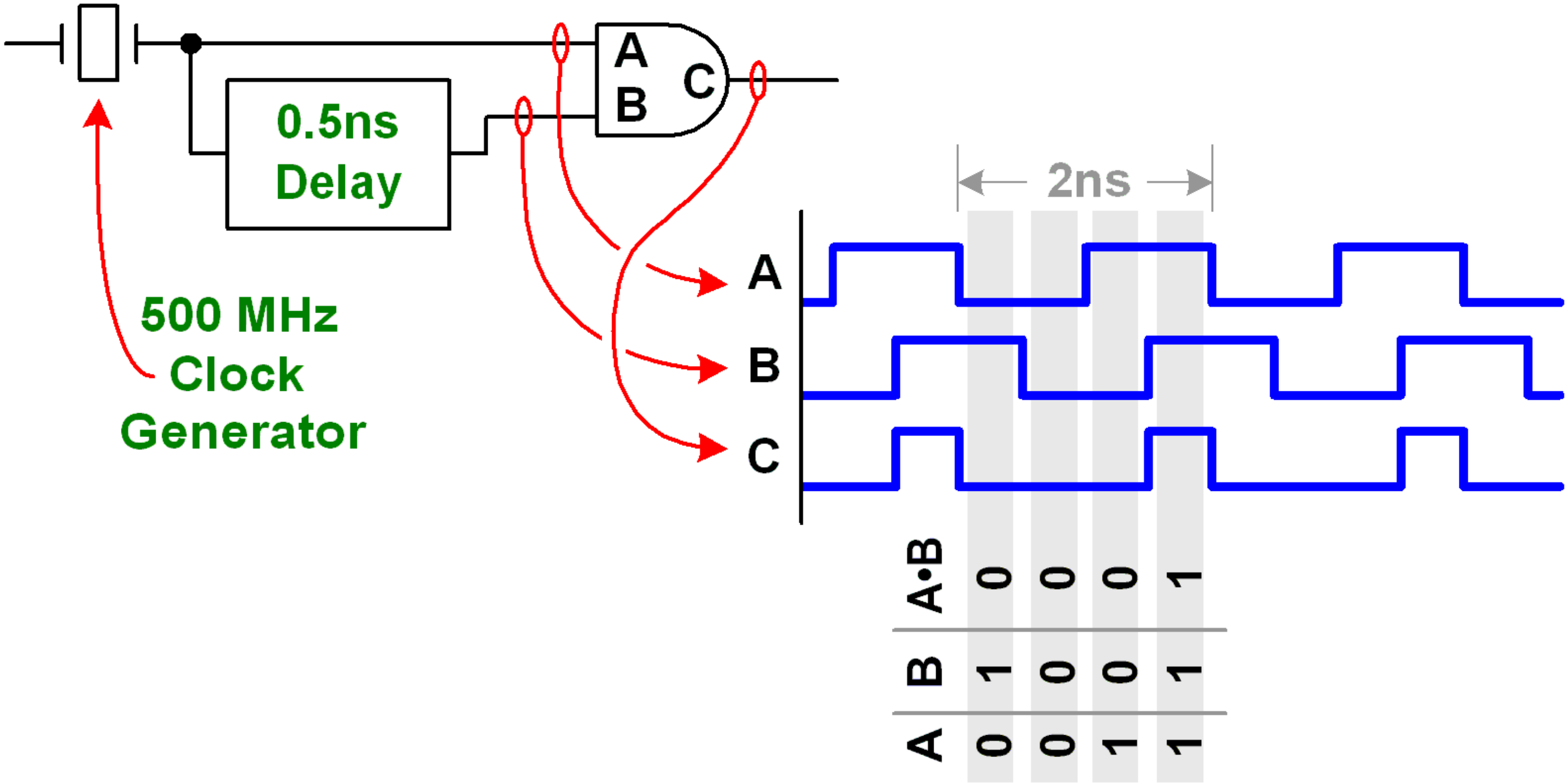
frequency = 1 / cycle time

The clock on the previous page has a 2ns (.000 000 002s) cycle time, so it's frequency is 500MHz (1 / .000 000 002).

In general, a CPU will run faster or slower as the clock frequency is increased or decreased. But there is a maximum speed beyond which the circuits cannot keep up – if the frequency is increased beyond this limit the CPU will stop working correctly.

An Asymmetric Clock

Most modern computers use a simple clock as described above, but some systems (older ones, in particular) may use an asymmetric clock where the "on" time of the clock is different than the "off" time. This can be done using a delay circuit and an AND gate as shown below:

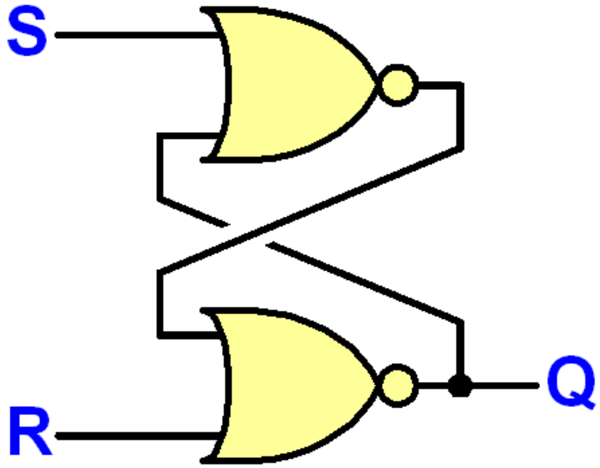


A	B	A•B
0	1	0
0	0	0
1	0	0
1	1	1

This example shows the output of a 500MHz clock generator being fed through a 0.5ns delay circuit. The undelayed signal and the delayed signal are fed into an AND gate. This gate produces a TRUE output only when both inputs are true, which occurs only ¼ of the time.

Latches

A latch is a one-bit memory circuit. It uses two cross-connected NOR gates, each of which feeds back into the other's input as shown below.



Unlike the sequential logic circuits we studied earlier, this feedback circuit can't be analyzed using a truth table or Karnaugh diagram.

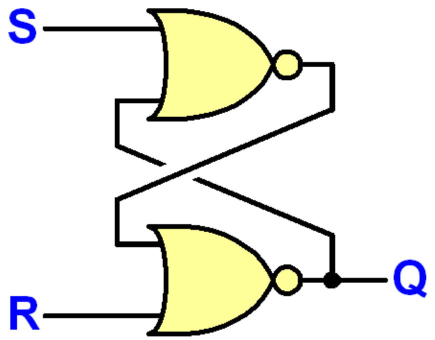
The latch has three connections:

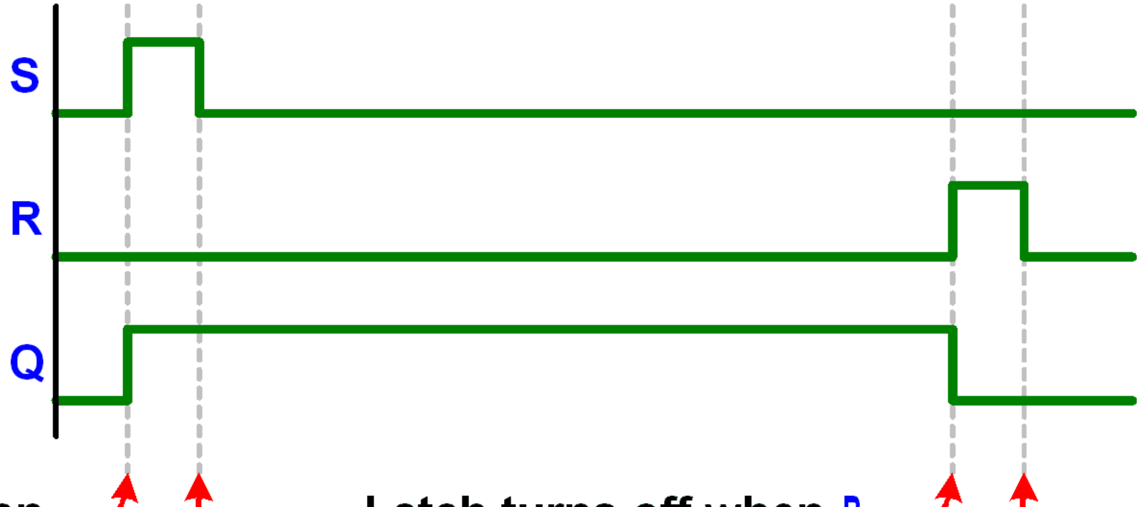
S	**The "set" input – when TRUE this input turns the latch "ON"**
R	**The "reset" input – when TRUE this input turns the latch "OFF"**
Q	**The output – TRUE when the latch is "ON" and FALSE when it is "off"**

This type of latch is called an "S-R Latch" after the types of inputs that control it. Note that it's not valid to try to set both the S and R inputs to TRUE at the same time.

S-R Latch Operation

The timing diagram below shows how an S-R latch operates:





Latch turns on when S becomes TRUE

Latch stays on <u>even after</u> S becomes FALSE

Latch turns off when R becomes TRUE

Latch stays off <u>even after</u> R becomes FALSE

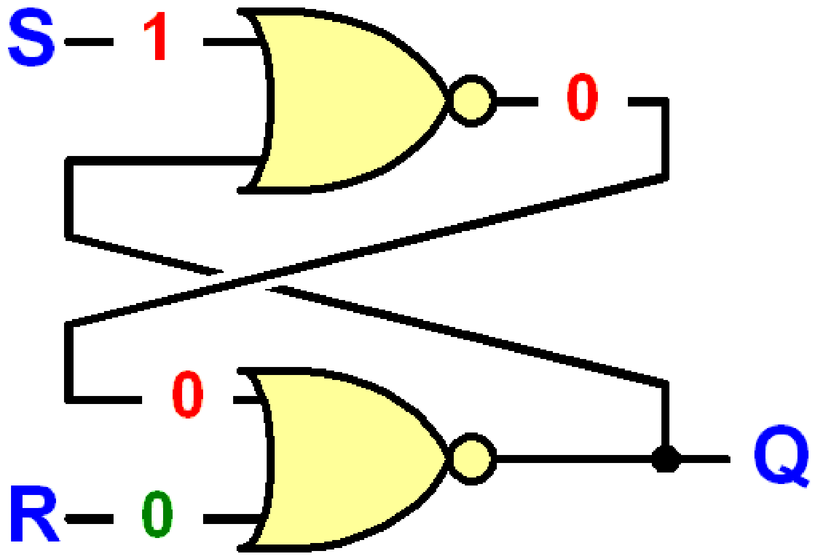
Note that when S=0 and R=0, the output of the latch can be in one of two different states. This is different from the sequential logic circuits we studied earlier – in those circuits if the inputs had a particular value then the output would always be the same.

Because of this, <u>you can't tell what state the latch is in just by looking at the inputs</u>. You have to know whether the last thing that was done to the latch was to SET it or RESET it.

Turning the Latch On – Step 1

3.3.1

Let's see how the latch works. To turn the latch on, we set the S input to TRUE while the R input is FALSE.



A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

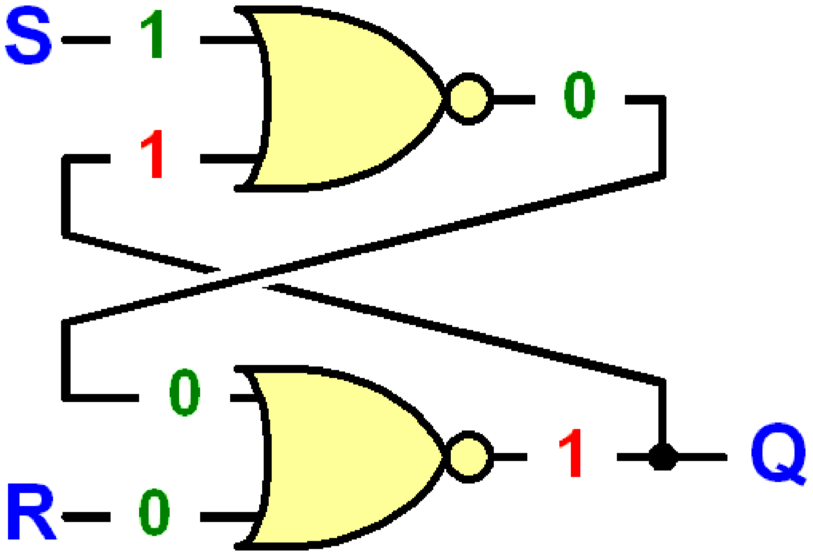
We don't know what the other input to the top NOR gate is, and so it's tricky to tell what the output would be. But look at the truth table for the NOR function. You can read the truth table as:

The output is FALSE if either input is TRUE

Since one input of the top NOR gate is TRUE, this means it's output must be FALSE. That FALSE input is fed back into the bottom NOR gate as shown above.

Turning the Latch On – Step 2

According to the NOR Truth Table, when both inputs to the bottom NOR gate are FALSE, the output must be TRUE:

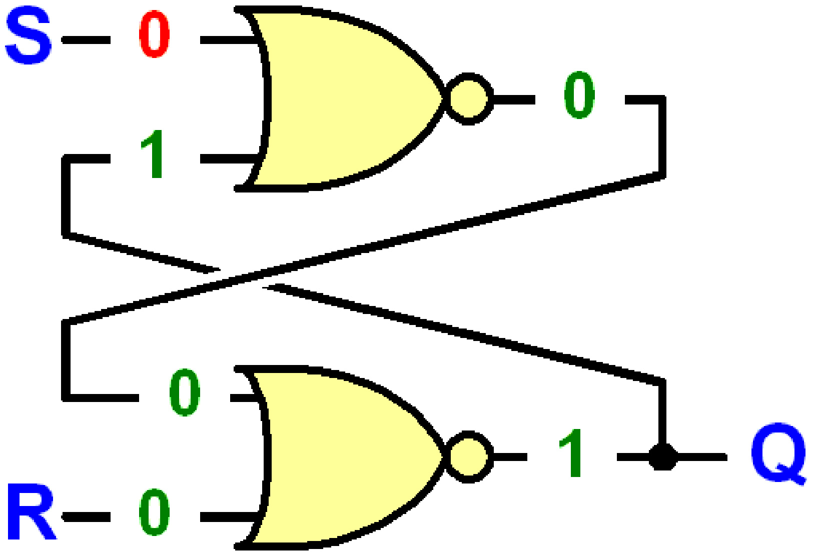


A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

This TRUE output becomes the Q output of the latch, and it's also fed back into the top NOR gate as shown above.

Turning the Latch On – Step 3

Finally, we set the S input back to FALSE again. Since both inputs to the top NOR gate are TRUE, the output stays FALSE:



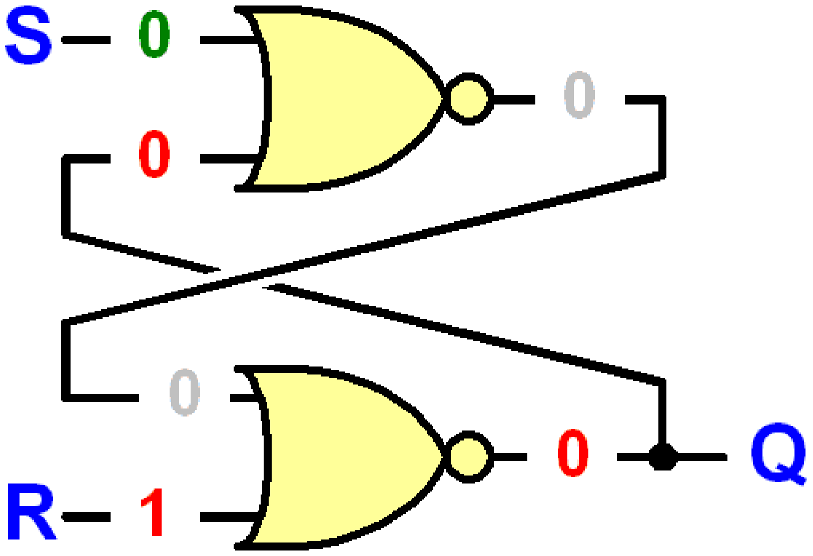
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

The FALSE output of the top NOR gate is fed back into the bottom NOR gate, which continues to have two FALSE inputs and a TRUE output (which is the Q output of the latch).

And, the TRUE output of the bottom NOR gate feeds back into the top NOR gate to maintain it’s state.

Turning the Latch Off – Step 1

To turn the latch off again, we now set the R input to TRUE while keeping the S input set to FALSE:

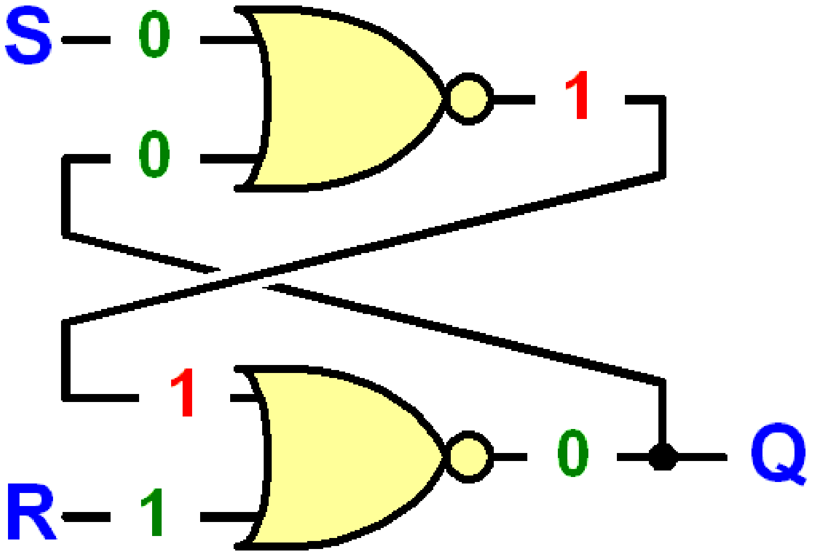


A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

The NOR truth table tells us that when either input is TRUE the output is FALSE – so the output of the bottom NOR gate now changes to FALSE. This is the latch's output, which is also fed back as an input to the top NOR gate.

Turning the Latch Off – Step 2

To turn the latch off again, we now set the R input to TRUE while keeping the S input set to FALSE:

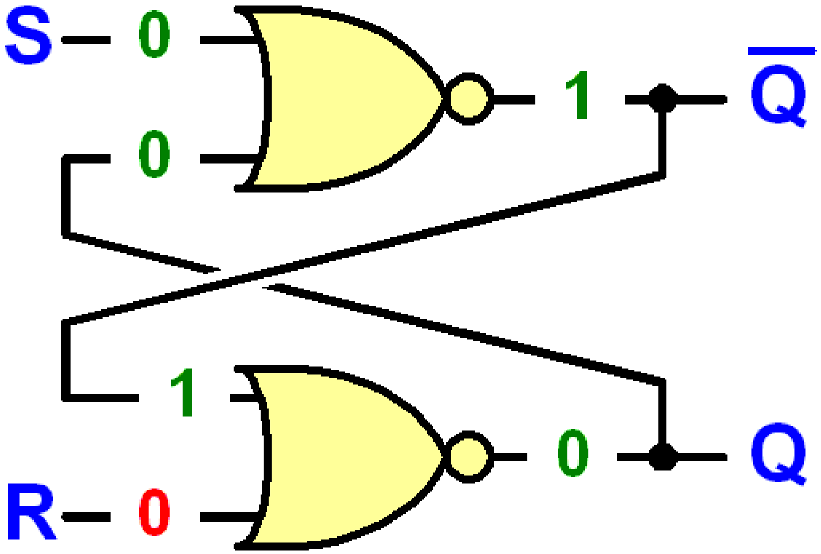


A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Now that the top NOR gate has two FALSE inputs, it's output changes to TRUE and is fed back into the bottom NOR gate. This makes the bottom NOR gate's two inputs TRUE as well, and it continues to output a FALSE value.

Turning the Latch Off – Step 3

Finally, we set the R input back to FALSE again. Since both inputs to the bottom NOR gate are TRUE, the output stays FALSE:



A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

The FALSE output of the bottom NOR gate (which is the Q output of the latch) is fed back into the top NOR gate, which continues to have two FALSE inputs and a TRUE output.

And, the TRUE output of the top NOR gate feeds back into the bottom NOR gate to maintain it's state.

You may have noticed that the output of the top NOR gate is always the opposite from the bottom one. This gate's output is sometimes used and called "Not-Q"

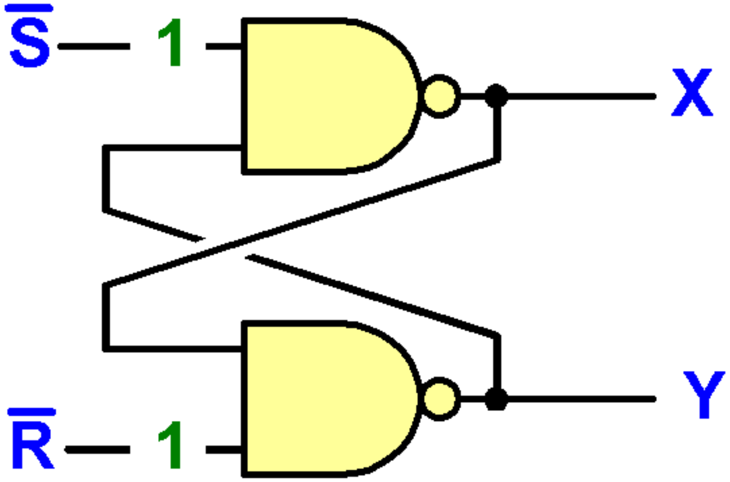
NAND Latch

3.3.1

Latches can also be built from NAND gates. A latch built this way has negative logic inputs:

Not-S **The "Set" input. When FALSE this turns the latch "ON".**
Not-R **The "Reset" input. When FALSE this turns the latch "OFF".**

For a NAND latch, the "normal" state (when we're not trying to turn the latch ON or OFF) is that both of it's inputs are TRUE:

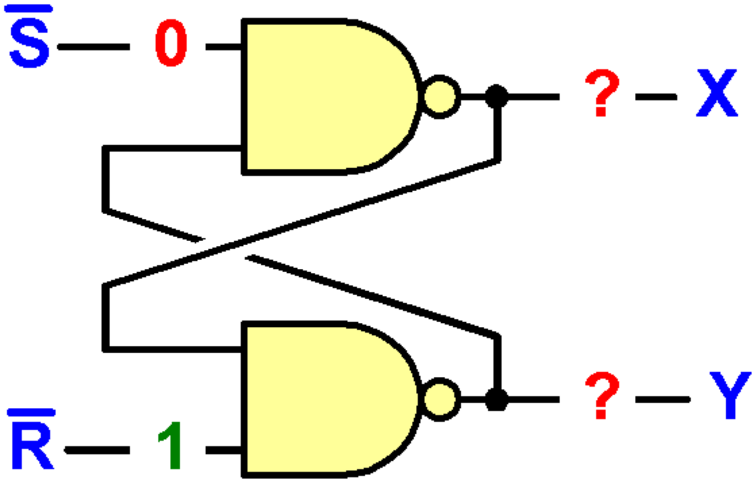


A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

In the above diagram, we can't tell what the outputs of the latch are because we can't tell what the outputs of the NAND gates are. When we only know that one NAND input is TRUE, the output could be TRUE or FALSE depending on the other input.

Exercise 1 – NAND Latch

We'll turn the NAND latch ON by setting the Not-S input to FALSE:



A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

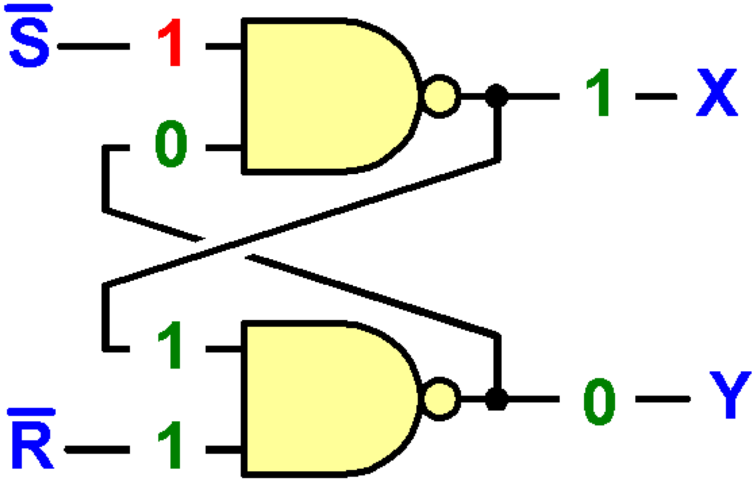
1. **What is the value of the X output at this point? Is it TRUE (A) or FALSE (B)?**

Since the X output is also connected to the bottom NAND gate, we now know what both it's inputs are and therefore we should be able to tell what the Y output is as well.

2. **What is the value of the Y output at this point? Is it TRUE (A) or FALSE (B)?**

NAND Latch

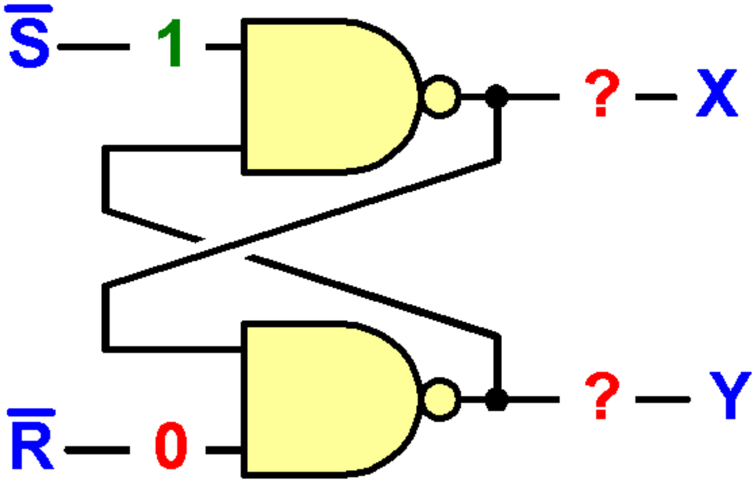
Notice that the outputs of the latch stay the same even after we return the Not-S input back to it’s normal FALSE state:



A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

Exercise 2 – NAND Latch

We'll turn the NAND latch OFF by setting the Not-R input to FALSE:



A	B	$\overline{A \bullet B}$
0	0	1
0	1	1
1	0	1
1	1	0

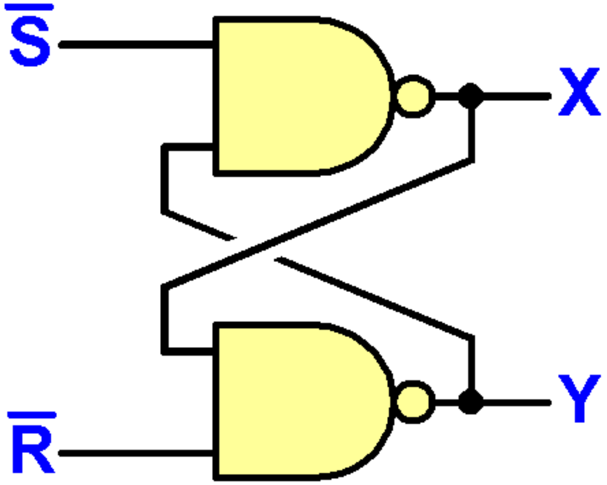
1. **What is the value of the Y output at this point?   Is it TRUE (A) or FALSE (B)?**

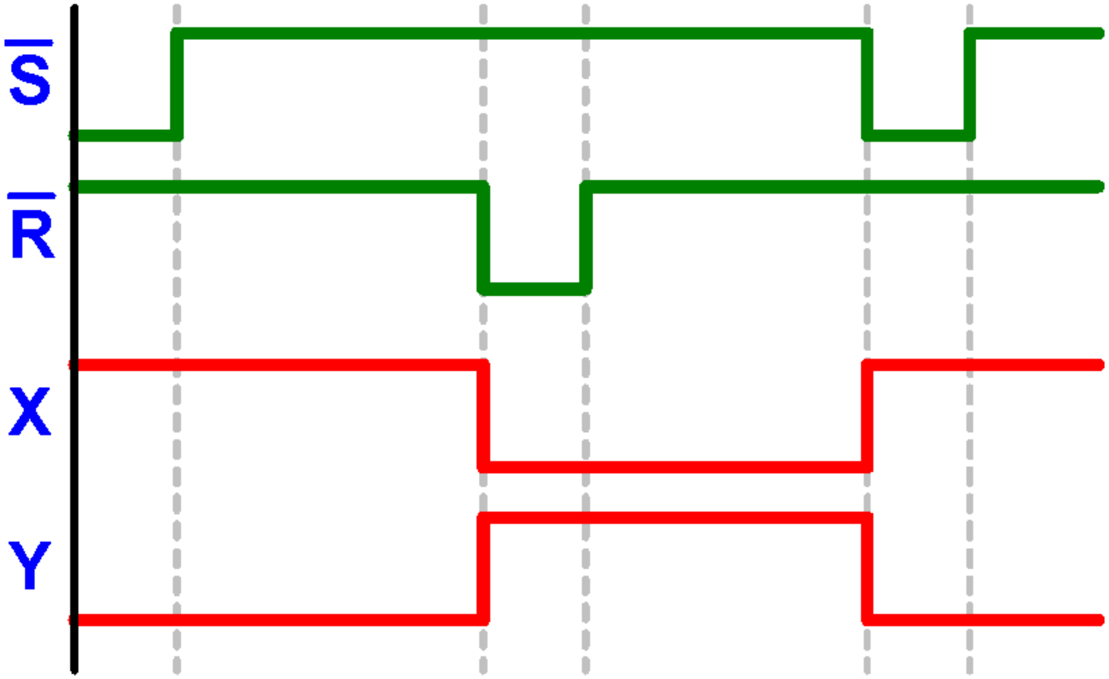
Since the Y output is also connected to the top NAND gate, can check to see if it's output has changed as well.

2. **What is the value of the X output at this point?   Is it TRUE (A) or FALSE (B)?**

NAND Latch

By tracing the changes to the latch, we can see that the outputs follow this pattern:





Since the X output is the one which becomes TRUE when the latch is "Set", it's the one that should be labeled "Q". The Y output would therefore be labeled "Not-Q".

“Asserting” A Signal

It can be confusing to talk about negative logic signals. For example, in the NAND gate flip/flop with the negative-logic inputs, if we say that the NOT-S input is TRUE, do we mean:

a) that the signal is at the “TRUE” (5 volt) logic level and therefore the latch is NOT being “set”? or

b) that the logical “SET” action is occurring and therefore the signal is at the FALSE (0 volt) logic level?

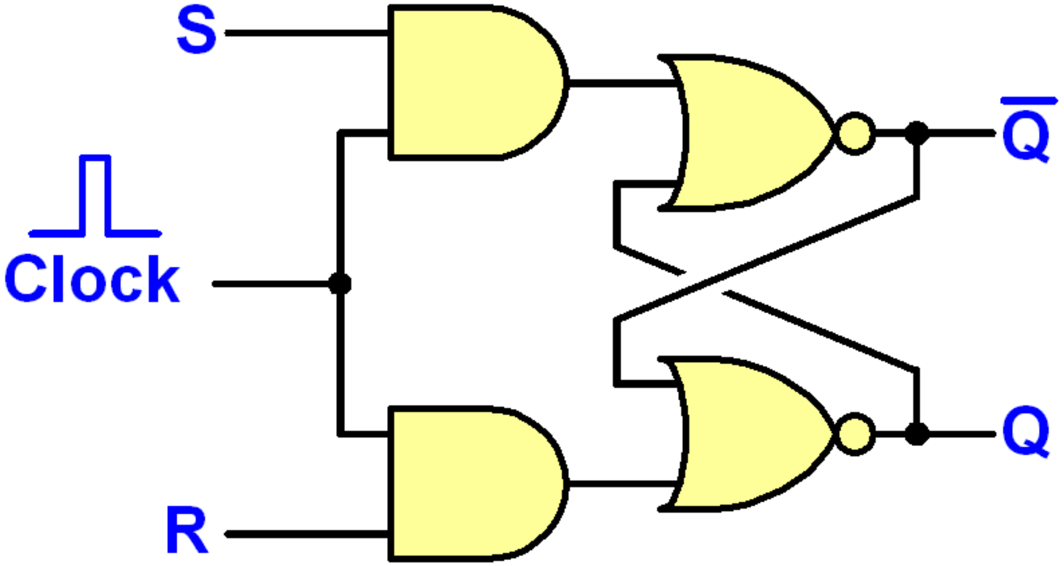
To avoid this confusion we use the term “assert”. When a signal is “asserted”, it means that the signal is set to the level required to perform whatever logical function the signal is for. So in the example of the SET signal for a latch:

In the case of the positive-logic NOR gate latch with “S” and “R” inputs, “SET Asserted” means that the signal is at the TRUE (5 volt) level.

In the case of the negative-logic NAND gate latch with “Not-S” and “Not-R” inputs, “SET Asserted” means that the signal is at the FALSE (0 volt) level.

A Clocked S-R Latch

We can add AND gates to the S and R inputs of an S-R latch to act as a “gatekeeper” to the latch and prevent it from being changed except when we want it to:



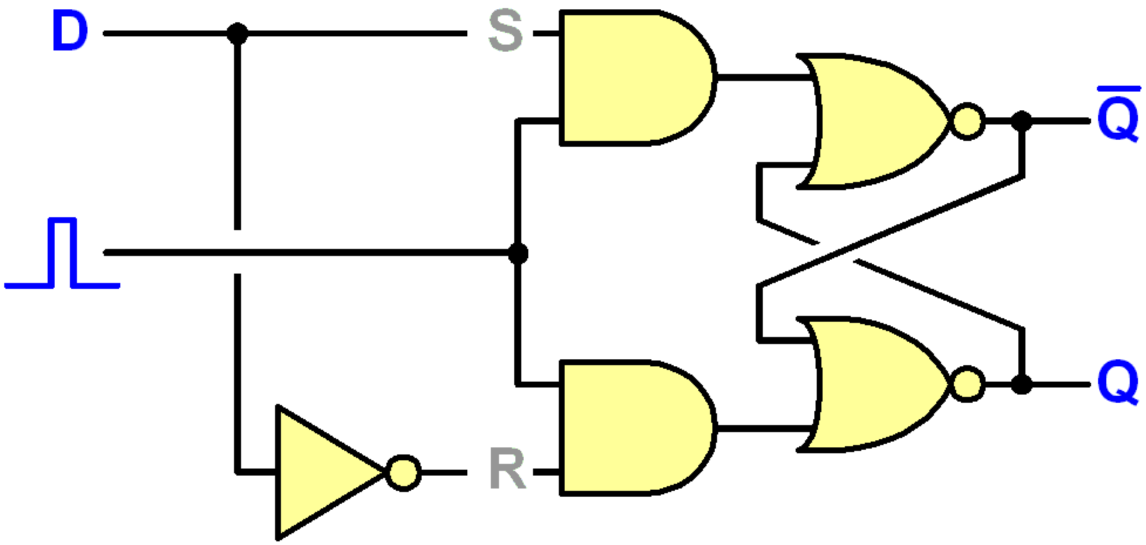
The right half of this circuit is a standard S-R latch. The two AND gates prevent the S or R signals from reaching the latch unless the “Clock” signal is also TRUE.

The clock signal provides a way to control <u>when</u> the latch may be changed.

A Clocked D Latch

The separate "S" and "R" inputs of an S-R latch are a little awkward to deal with. We usually have a single TRUE / FALSE Boolean signal which we want to store in memory. If the signal is TRUE we want to "Set" the latch, and if the signal is FALSE we want to "Reset" it.

We can use an inverter to convert a single input data signal into the "S" and "R" inputs of a clocked S-R latch:



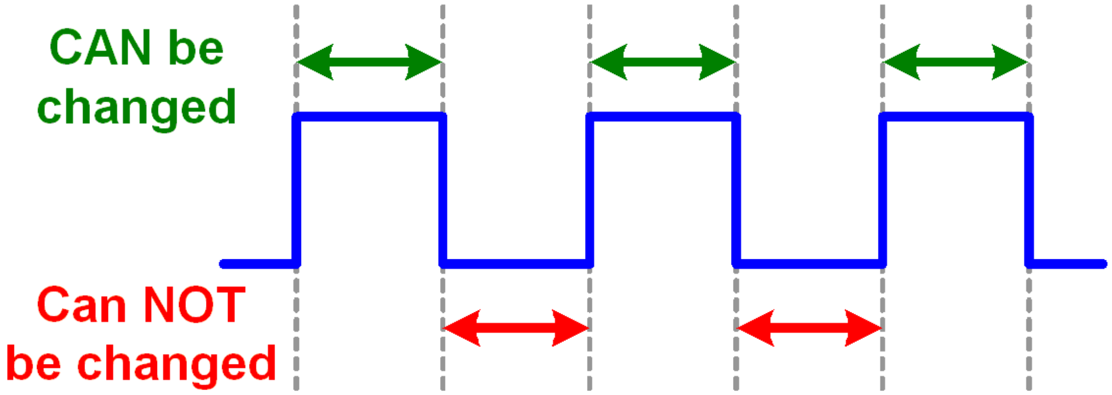
When the "D" input is TRUE, S=1 and R=0. When the "D" input is FALSE, S=0 and R=1.

This is where the clocked input is really needed. The "D" signal is always converted into either a SET or RESET signal, but the AND gates prevent those signals from changing the latch <u>except</u> when the CLOCK input is TRUE.

Level Triggering

3.3.1

Latches are known as “level triggered” devices because it’s the level of the clock that governs when their contents can be changed:

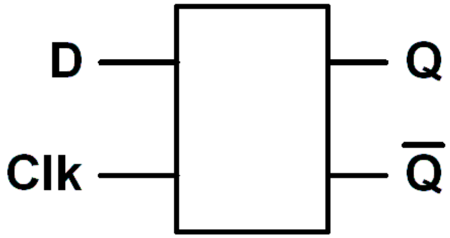


CAN be changed

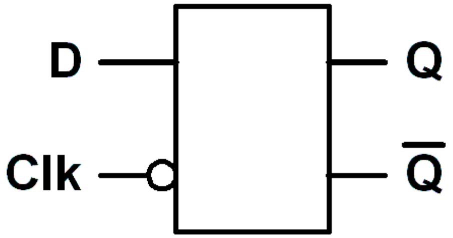
Can NOT be changed

The latch can be changed at any time while the clock signal is TRUE

It’s also possible to build latches that can change only when the clock is FALSE (negative level triggered latches). These are the diagramming symbols used to represent latches:



Positive Level Triggered Latch



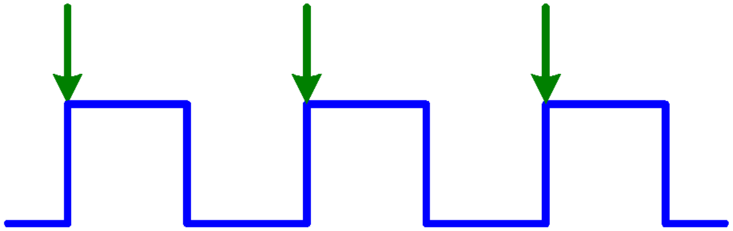
Negative Level Triggered Latch

Edge Triggering

Edge triggering can be used to more precisely control <u>when</u> the memory contents can be changed:

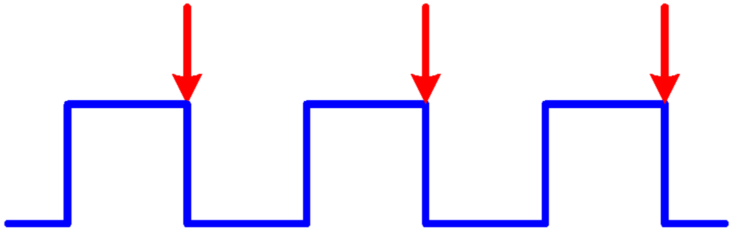
Leading Edge Triggered:

Can only be changed at the INSTANT the clock goes from FALSE to TRUE

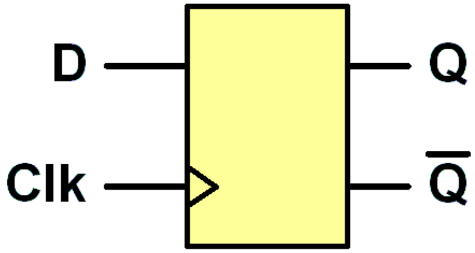


Trailing Edge Triggered:

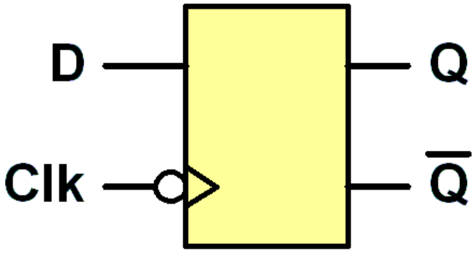
Can be only be changed at the INSTANT the clock goes from TRUE to FALSE



Memory circuits that use edge triggering are called <u>Flip-Flops</u>. Here are the diagramming symbols used for flip-flops:



Positive Edge Triggered F/F

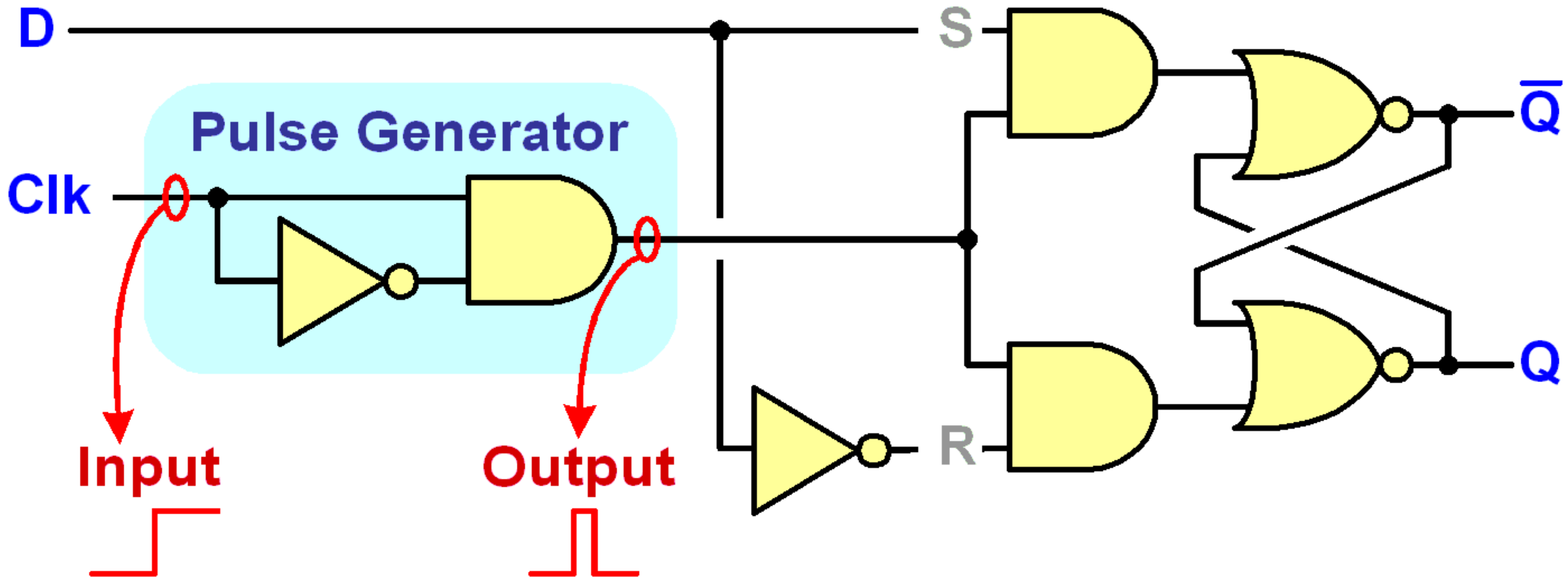


Negative Edge Triggered F/F

You can tell these are flip-flops and not latches because of the triangle at the “Clk” input.

Making an Edge Triggered Circuit

To convert an ordinary D latch into an edge-triggered flip-flop, we add some logic called a "Pulse Generator" to the input clock signal. The Pulse Generator generates a very brief, positive spike whenever the input clock signal goes from low to high. It's only during this brief spike that the contents of the flip-flop can be changed.

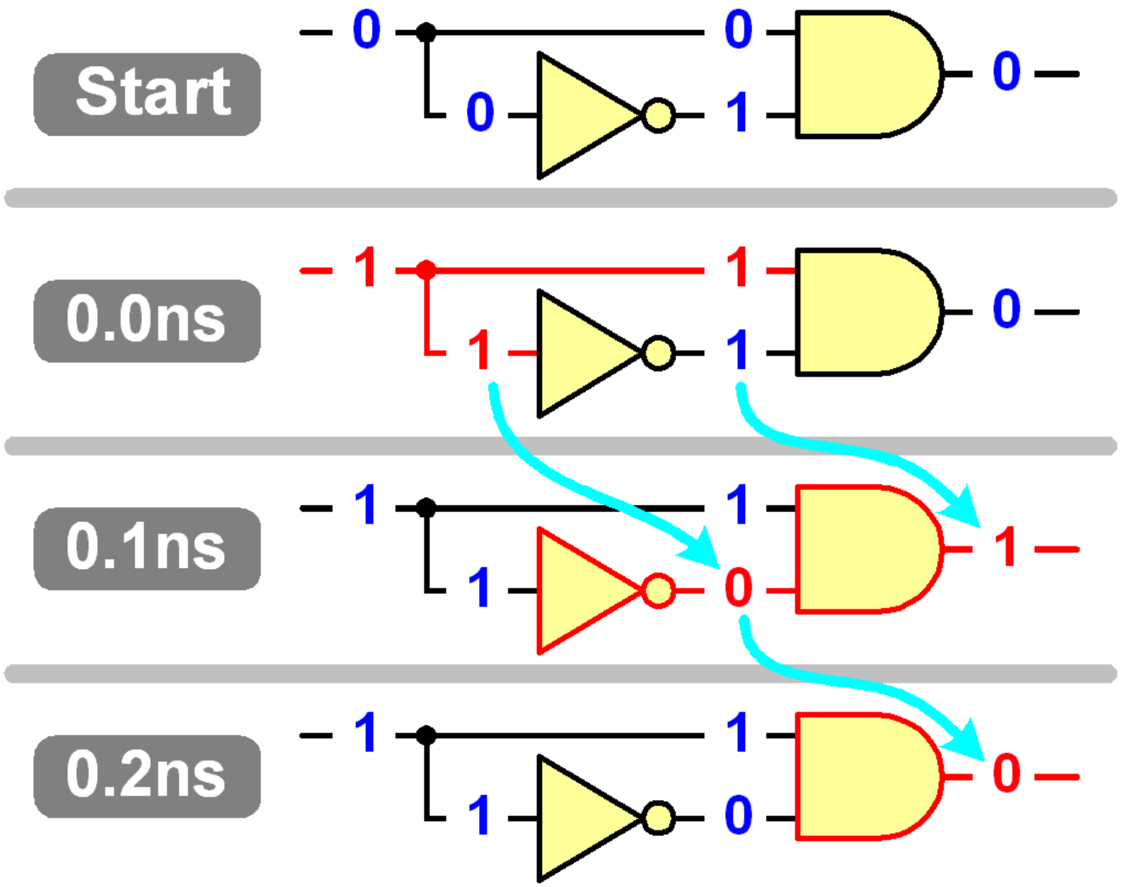


The Pulse Generator consists of a simple inverter (NOT gate) and AND gate as shown above. At first this circuit looks like it will always produce a false output, because any value ANDed with it's opposite always gives a zero result.

The secret of the Pulse Generator lies in the fact that it takes a small but finite time for signals to pass through a logic element such as an inverter or an AND gate.

How the Pulse Generator Works

To see how the pulse generator works, let's start with the clock input at the FALSE state and watch what happens when it changes to TRUE. We'll assume that it takes 0.1ns for a signal to pass through a gate and no time at all to travel along the wires from one gate to the next:



At 0.0ns the clock input goes from FALSE to TRUE. This input goes directly to the AND gate with no delay. But there is a 0.1ns delay before the signal goes through the inverter. During that 0.1ns, the AND gate has two TRUE inputs.

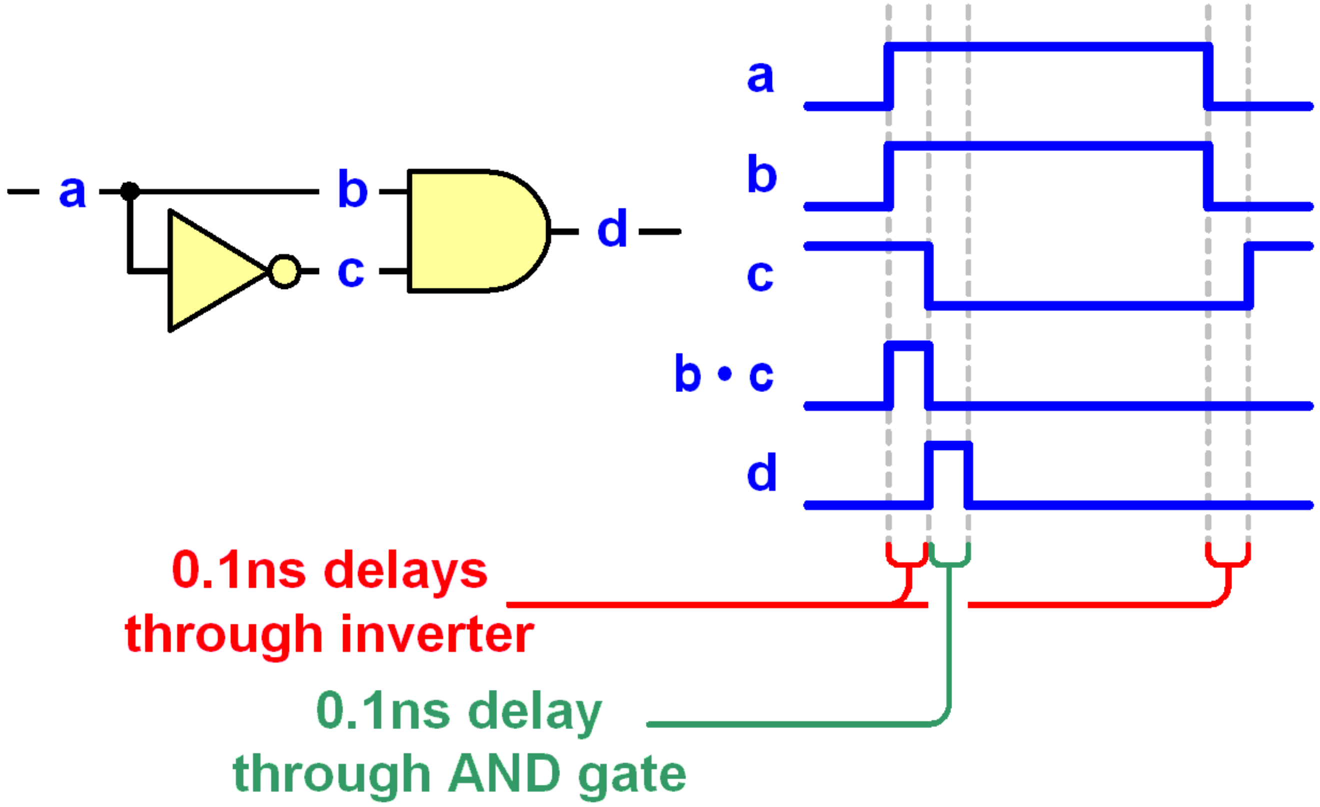
At 0.1ns, the two TRUE inputs have passed through the AND gate and produced a TRUE output from the pulse generator. At the same time, the inverter output goes FALSE and the AND gate no longer has two TRUE inputs.

At 0.2ns, the different inputs have passed through the AND gate and produced a FALSE output from the pulse generator.

For a brief 0.1ns interval, the pulse generator produces a TRUE output.

Pulse Generator Timing Diagram

Here is a timing diagram showing the relationship of the signals flowing through the pulse generator:



0.1ns delays through inverter

0.1ns delay through AND gate

This diagram shows why an output pulse only occurs at the <u>rising</u> edge of the clock. At the falling edge of the clock the delay through the inverter causes both “b” and “c” signals to become FALSE for a brief period of time – the output of the AND gate stays FALSE when this happens.

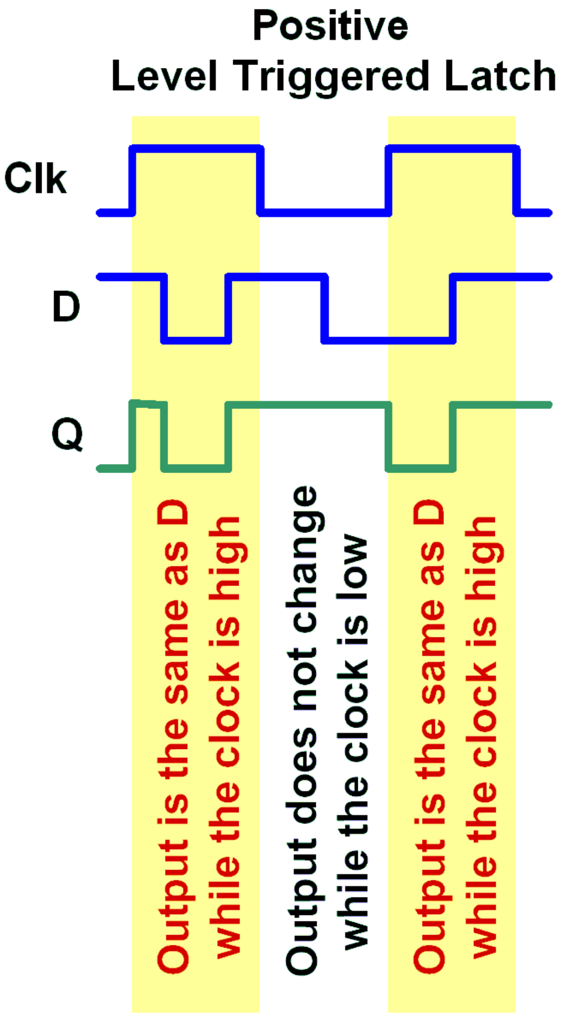
Level Triggering vs. Edge Triggering

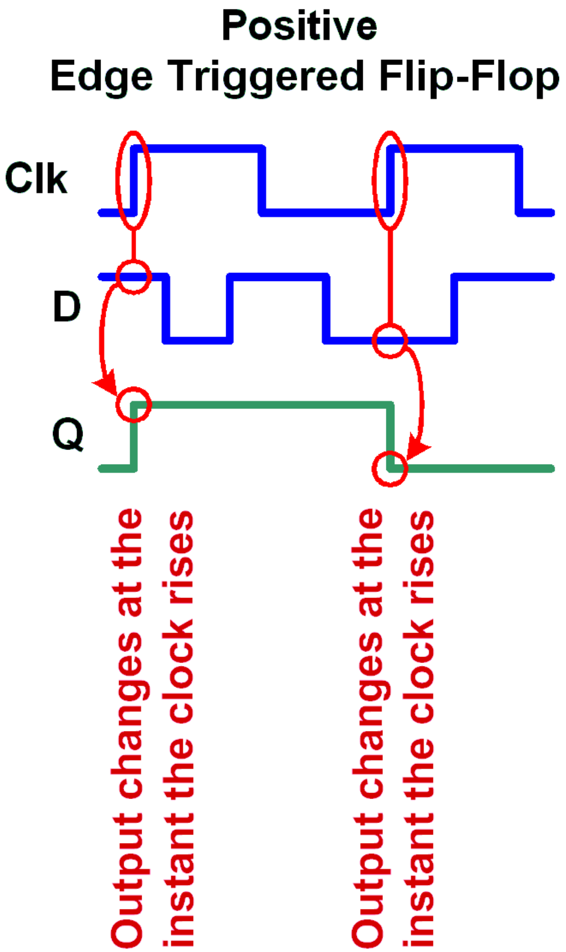
These diagrams show the difference in operation between a level triggered latch and an edge triggered flip-flop:

Positive level triggered circuits load the input “D” signal into the flip-flop (and thus to the Q output) as long as the clock is high. Positive edge triggered circuits load the input only at the instant the clock rises.

Negative triggered circuits work the same except that the polarity of the clock signal is reversed:

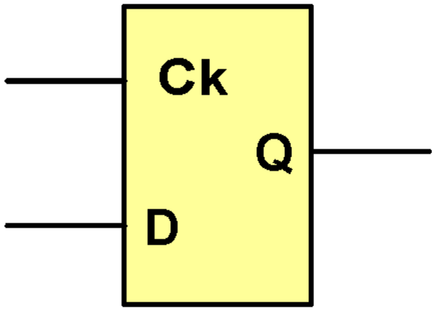
- Low clock level for negative level triggering
- The instant the clock falls for negative edge triggering.

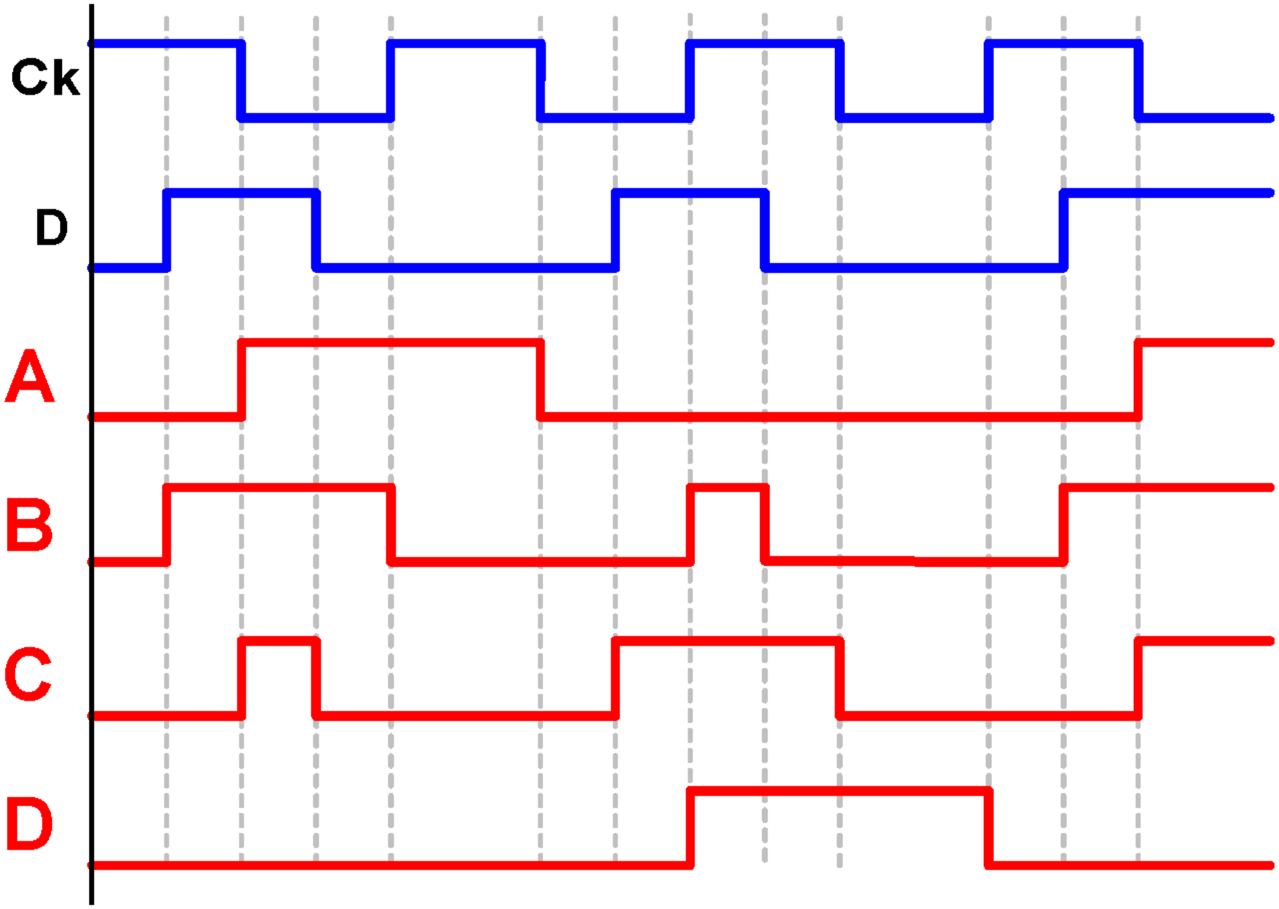

Positive
Level Triggered Latch
Clk
D
Q
Output is the same as D while the clock is high
Output does not change while the clock is low
Output is the same as D while the clock is high


Positive
Edge Triggered Flip-Flop
Clk
D
Q
Output changes at the instant the clock rises
Output changes at the instant the clock rises

Exercise 3 – Edge vs. Level Triggering

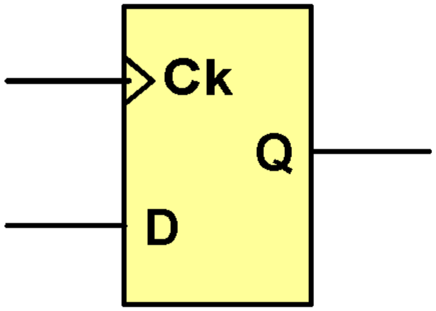
For a positive level-triggered latch, which of the red lines on the timing diagram represents what the Q output would look like for the given Clock and Data inputs?

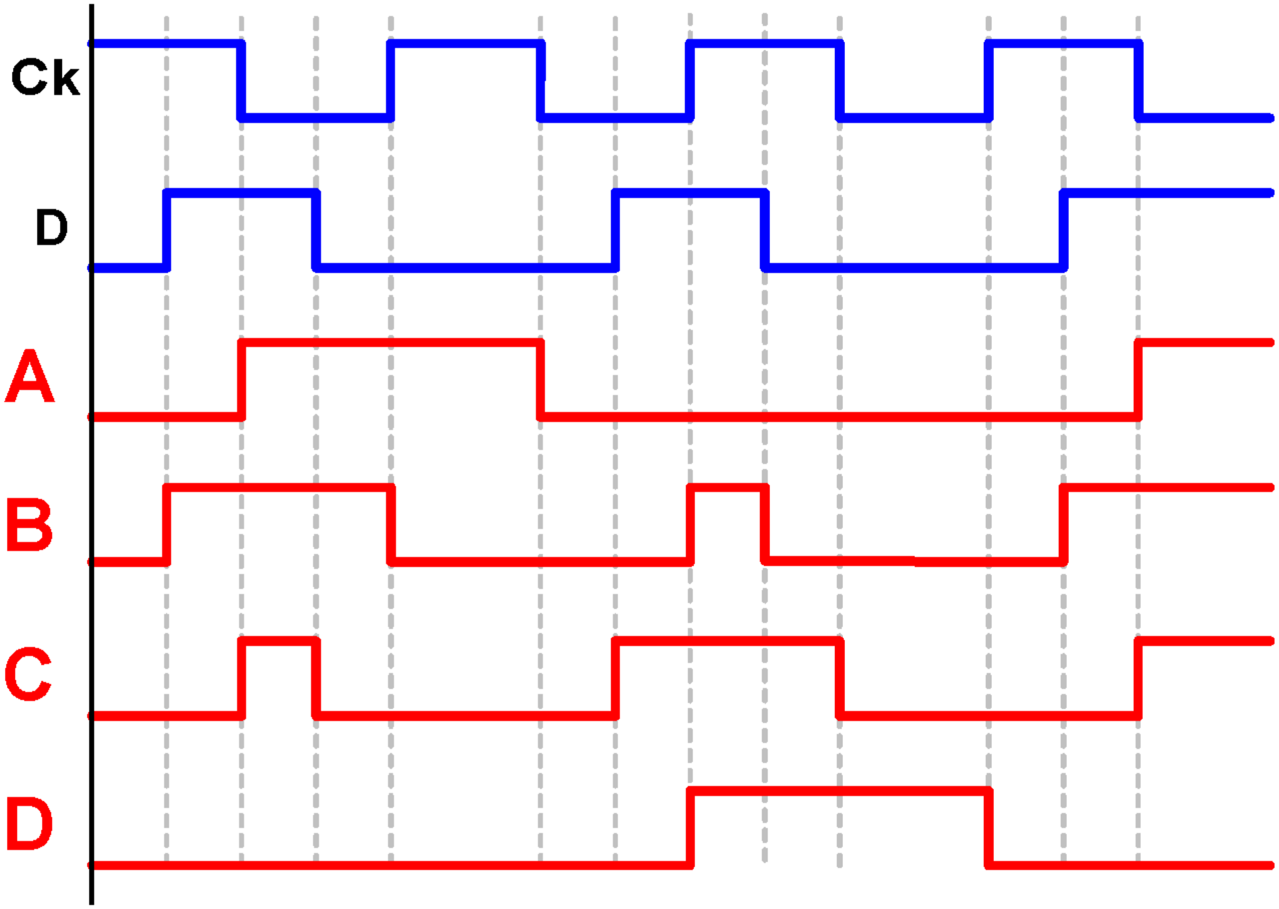




Exercise 4 – Edge vs. Level Triggering

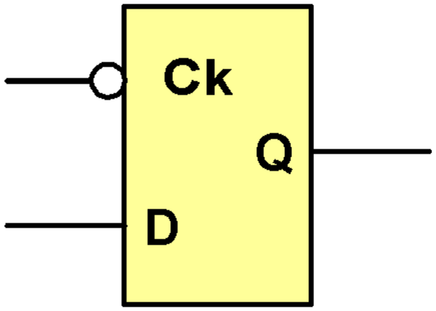
For a positive edge-triggered flip-flop, which of the red lines on the timing diagram represents what the Q output would look like for the given Clock and Data inputs?

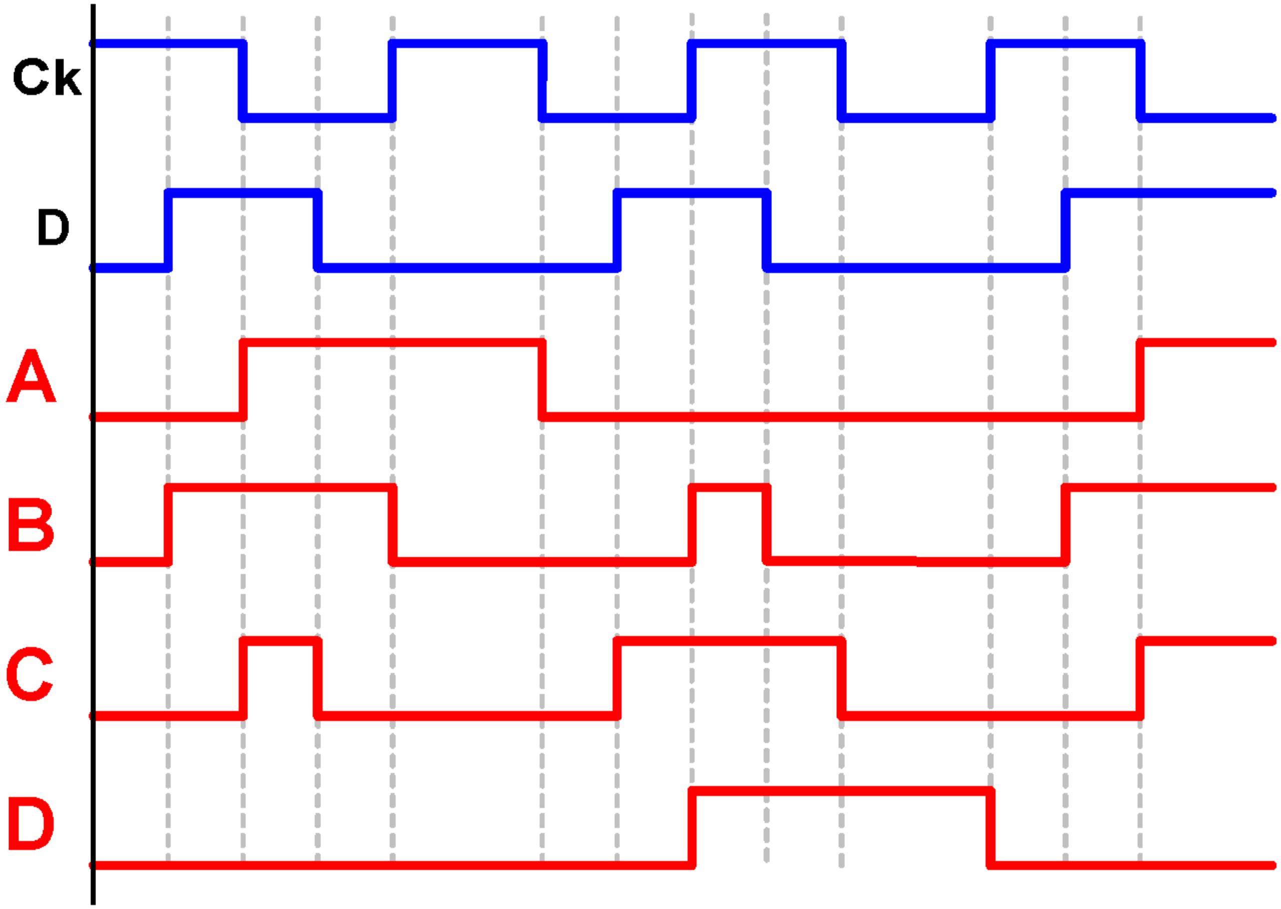




Exercise 5 – Edge vs. Level Triggering

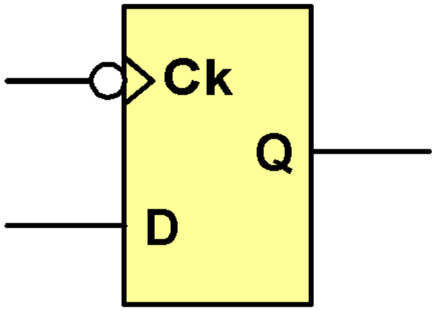
For a negative level-triggered-triggered latch, which of the red lines on the timing diagram represents what the Q output would look like for the given Clock and Data inputs?

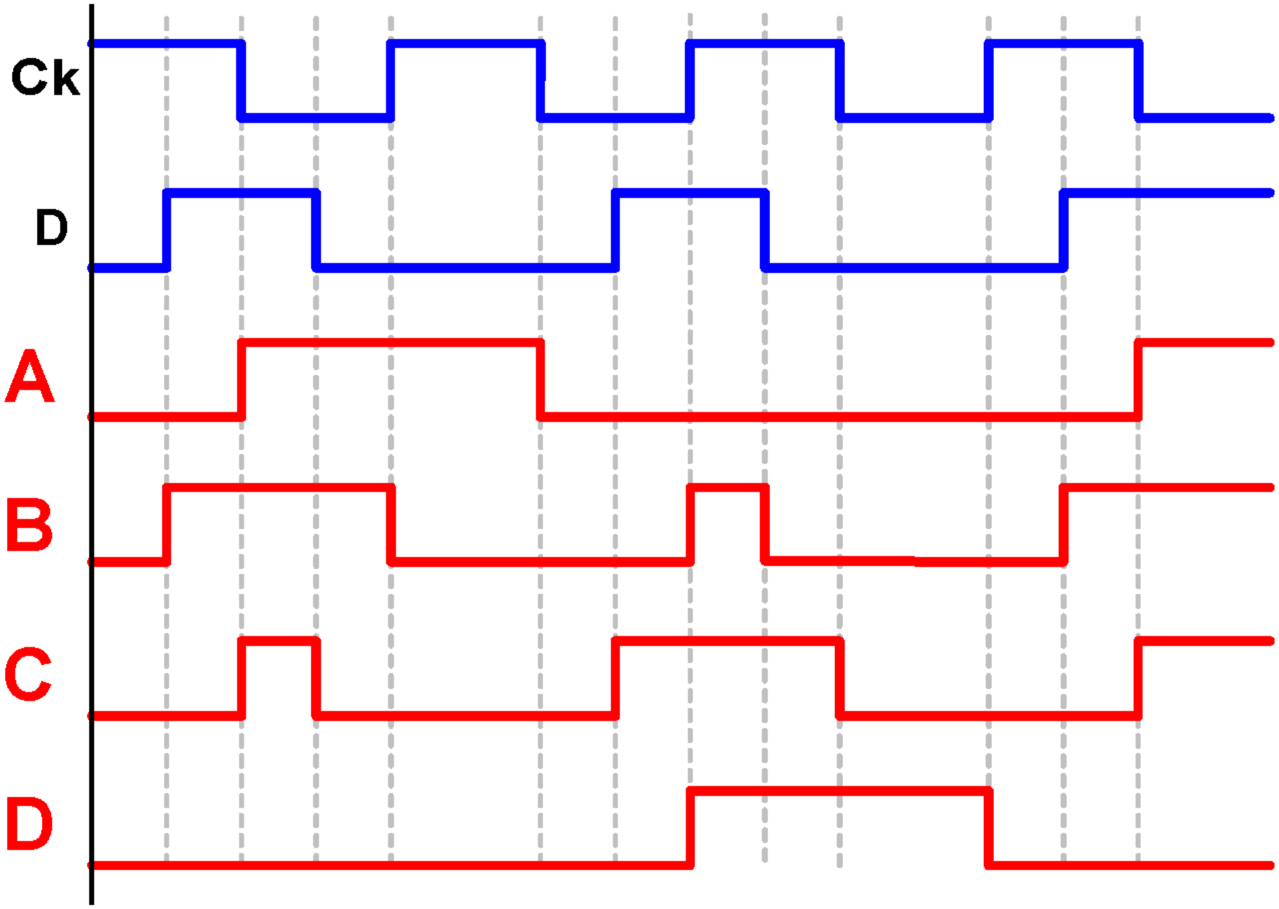




Exercise 6 – Edge vs. Level Triggering

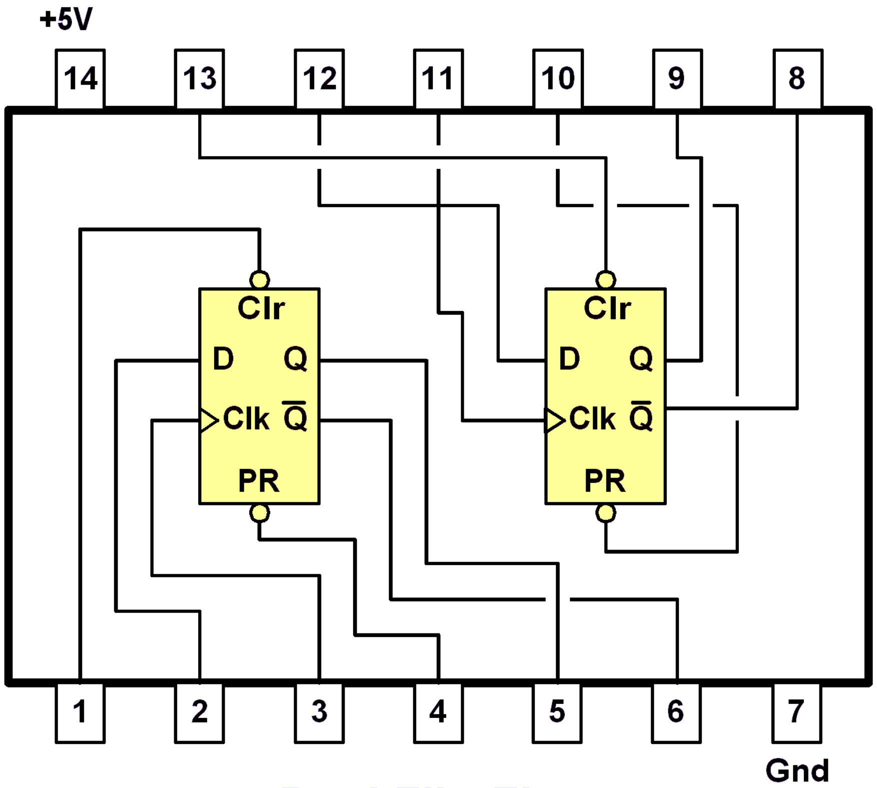
For a negative edge-triggered flip-flop, which of the red lines on the timing diagram represents what the Q output would look like for the given Clock and Data inputs?



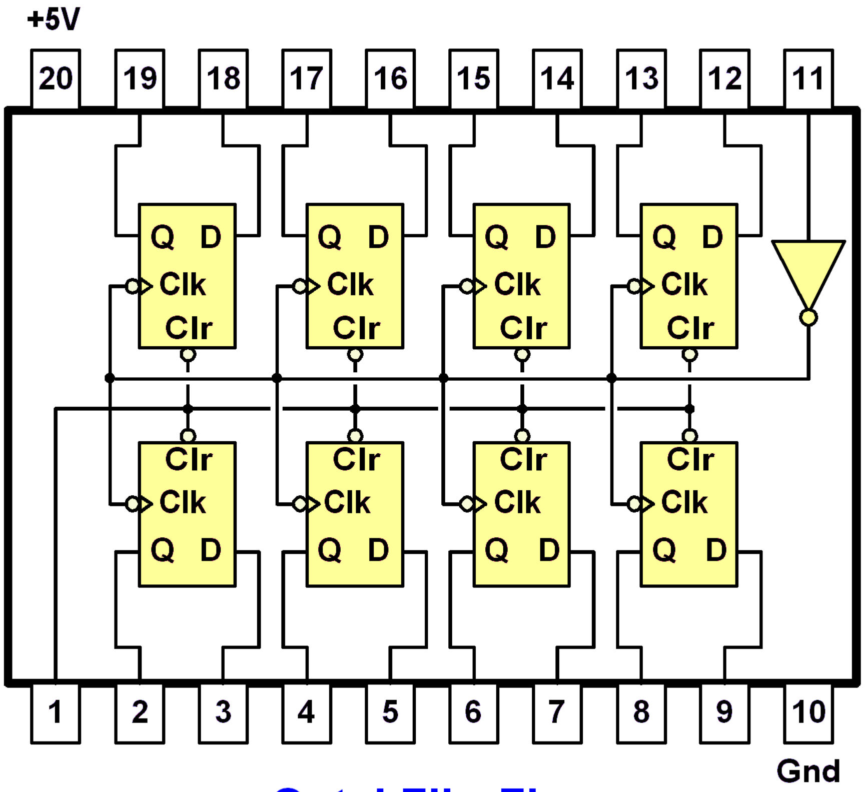


TTL Flip-Flop Chips

Here are a couple of examples of flip-flop TTL chips:



Dual Flip-Flop



Octal Flip-Flop

The dual chip has two independent flip-flops, while the octal chip has eight flip flops that all load data at the same time – they act as an 8-bit register.

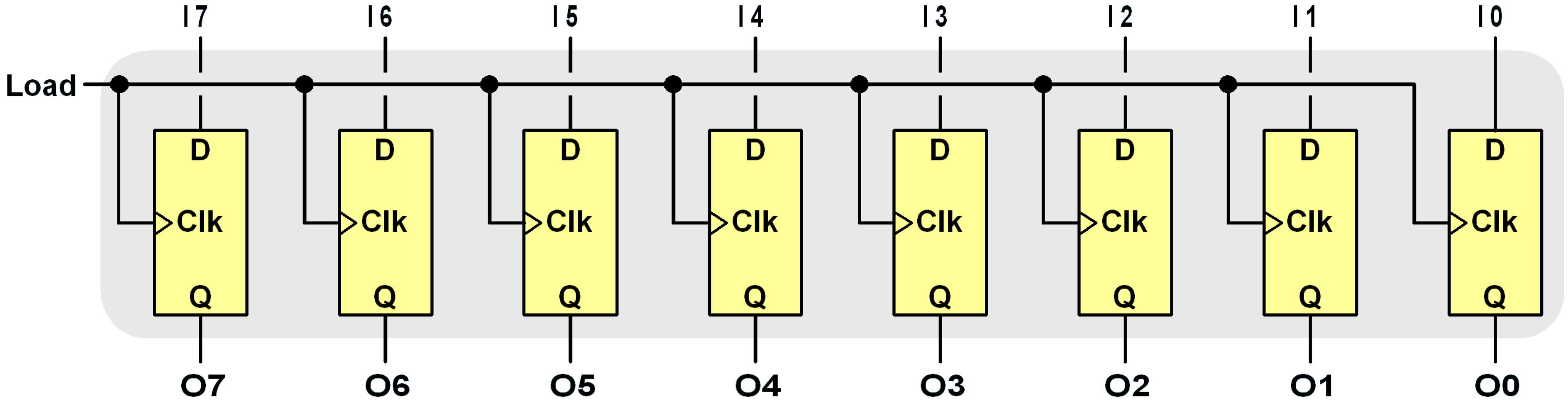
These flip-flops have two new inputs that we haven't seen yet:

Clr - ("Clear") forces the contents of the flip-flop to FALSE (like "R" in an S-R latch)

PR - ("Preset") forces the contents of the flip-flop to TRUE (like "S" in an S-R latch)

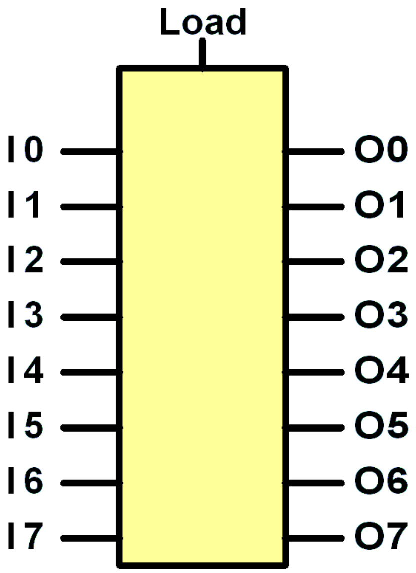
An 8-bit Register

The flip-flops we learned about last week can be used as a "register" – a group of bits used to store a number inside a CPU. Here's an example of 8 flip-flops put together as an 8-bit register:



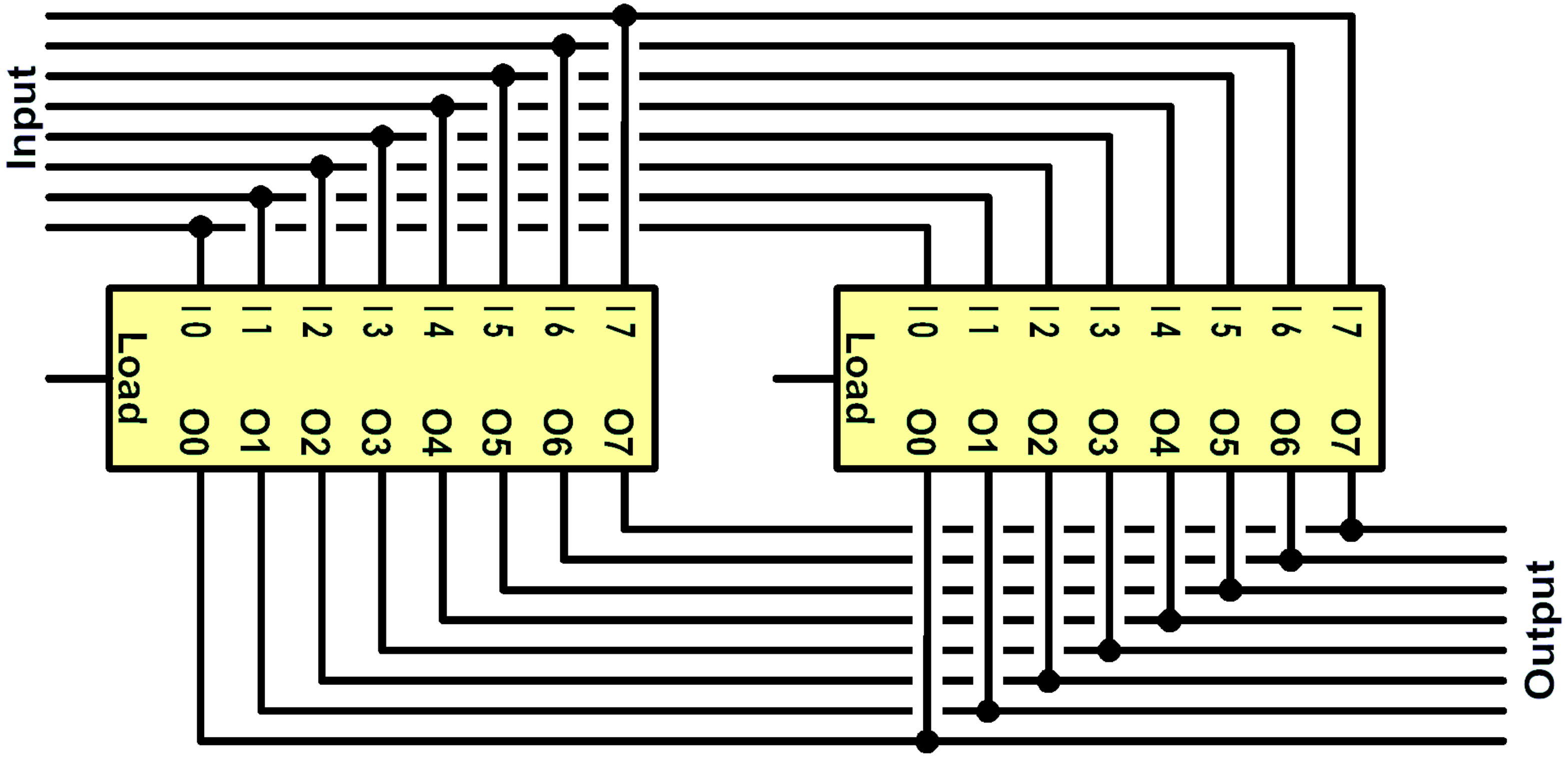
Data to be loaded into the register is sent to the "I0-I7" input lines and the "Write" signal is activated to actually load the data into the flip-flops. The information stored in the register is always present on the "O0-O7" output lines.

This entire register can be diagrammed as shown at the right:



The Problem with Hardwired Outputs

The register we just described has a problem – the contents of each of the flip-flops are always sent to the output connections.   This poses a problem if we want to connect the registers to a bus:

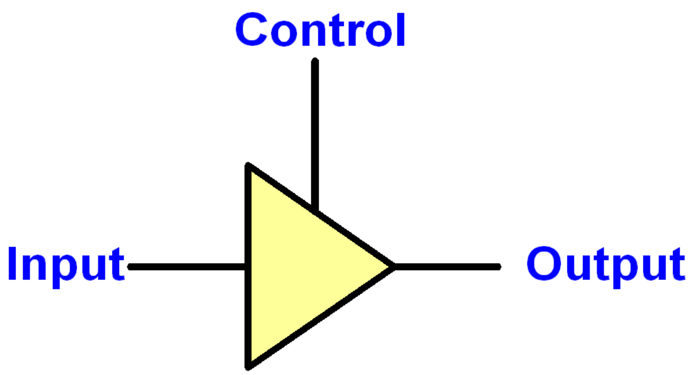


There's no problem connecting the inputs together because the voltages that drive the TRUE or FALSE signals come from somewhere else.

But having the outputs wired together is a problem.   What happens if the left O7 line is set to FALSE and the right O7 line is set to TRUE?   The two signals are wired together and they would both try to force the line to different values.   That won't work.

Solving the Output Problem: The Tri-State Buffer

To solve the problem we will introduce a new type of circuit: the Tri-State Buffer. Here is the diagram for this circuit:



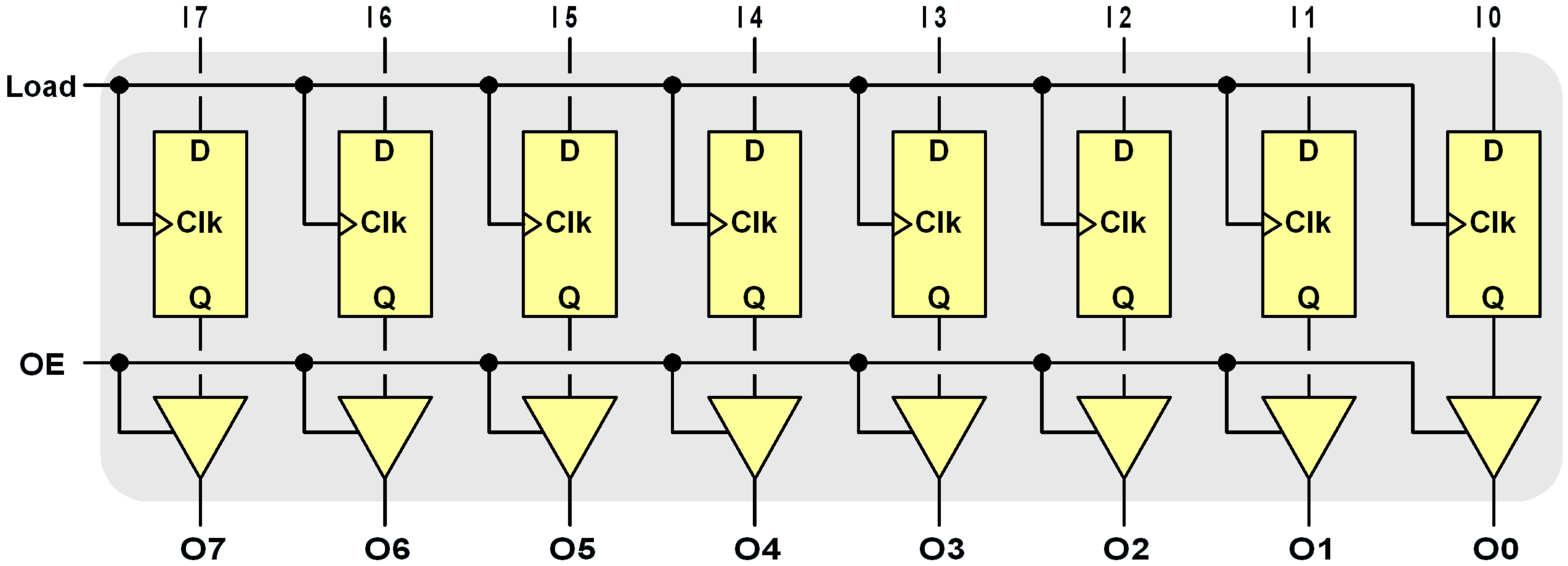
The Tri-State Buffer looks a lot like an inverter, except it doesn't have a circle at the output end and it has a third connection labeled "control". Here's the truth table for the circuit:

Control	Input	Output
0	x	Disconnected
1	0	0
1	1	1

The "third state" of a Tri-State Buffer is the "disconnected" state. In this state the buffer acts as if the output wire had been cut so that there is no connection. We can use this to build a register that lets us disconnect the outputs so we can connect multiple registers to a bus.

Register with Output Enable

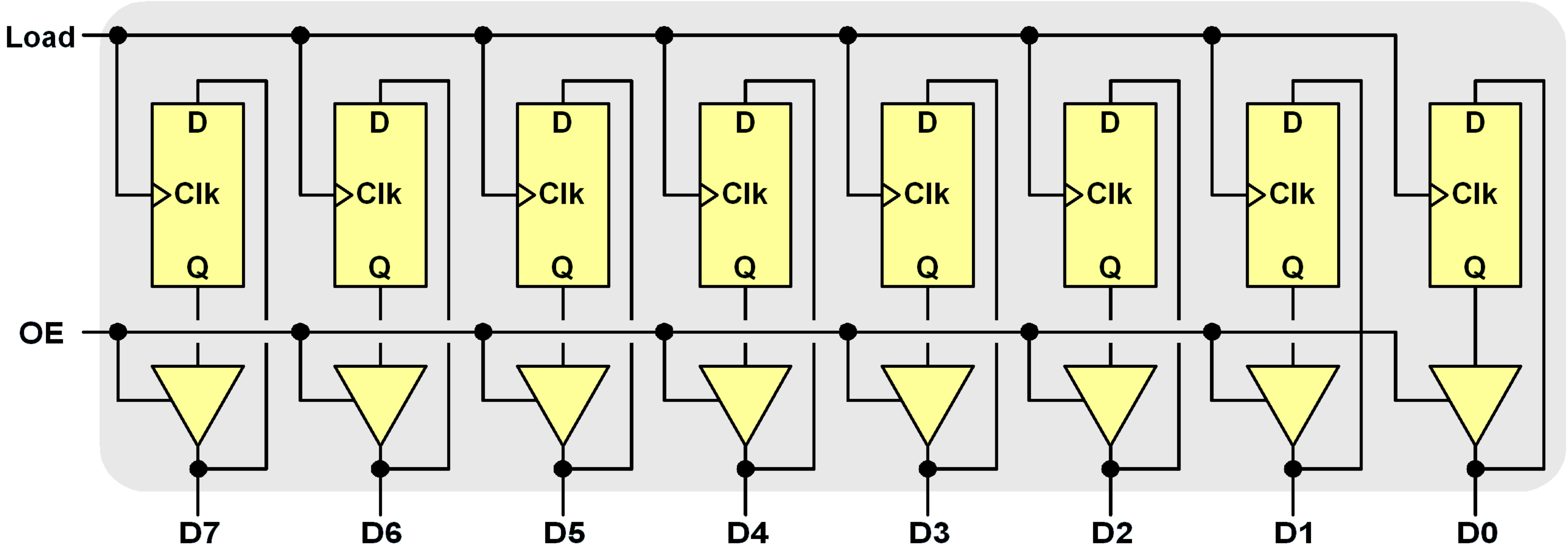
By adding tri-state buffers to the output of our register, we can control whether the register contents are delivered to the output pins or not:



Now we have a new control line labeled "OE" (Output Enable) which allows us to disconnect the output lines from the flip-flops. With this improvement we're able to connect different registers to the same bus lines as long as we only enable the outputs for one register at a time.

Register with Bidirectional Data Lines

With the addition of tri-state buffers to enable or disable the output, we can now eliminate the need for separate input and output data connections:



Now we only have one set of data lines which can be used to load data into the register or read data from the register:

Reading: **The “OE” signal is asserted and the contents of the flip-flops are connected to the data lines. These signals are sent to other devices in the system which use the information.**

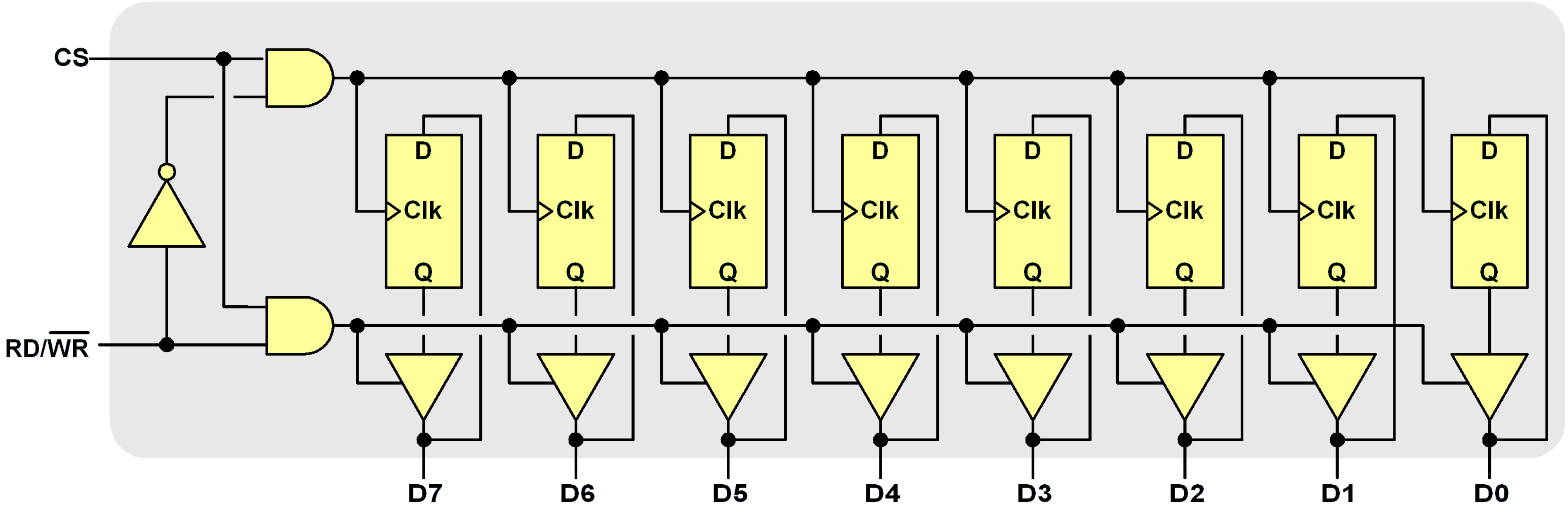
Writing: **The “OE” signal is not asserted and the “Load” signal is. Data is sent from other devices in the system through the data lines and is loaded into the flip-flops.**

Register with CS and RD/$\overline{\text{WR}}$ Lines

It's easier to connect a register to the system if, instead of LOAD and OE signals, we use CS ("Chip Select") and RD/$\overline{\text{WR}}$ control signals:

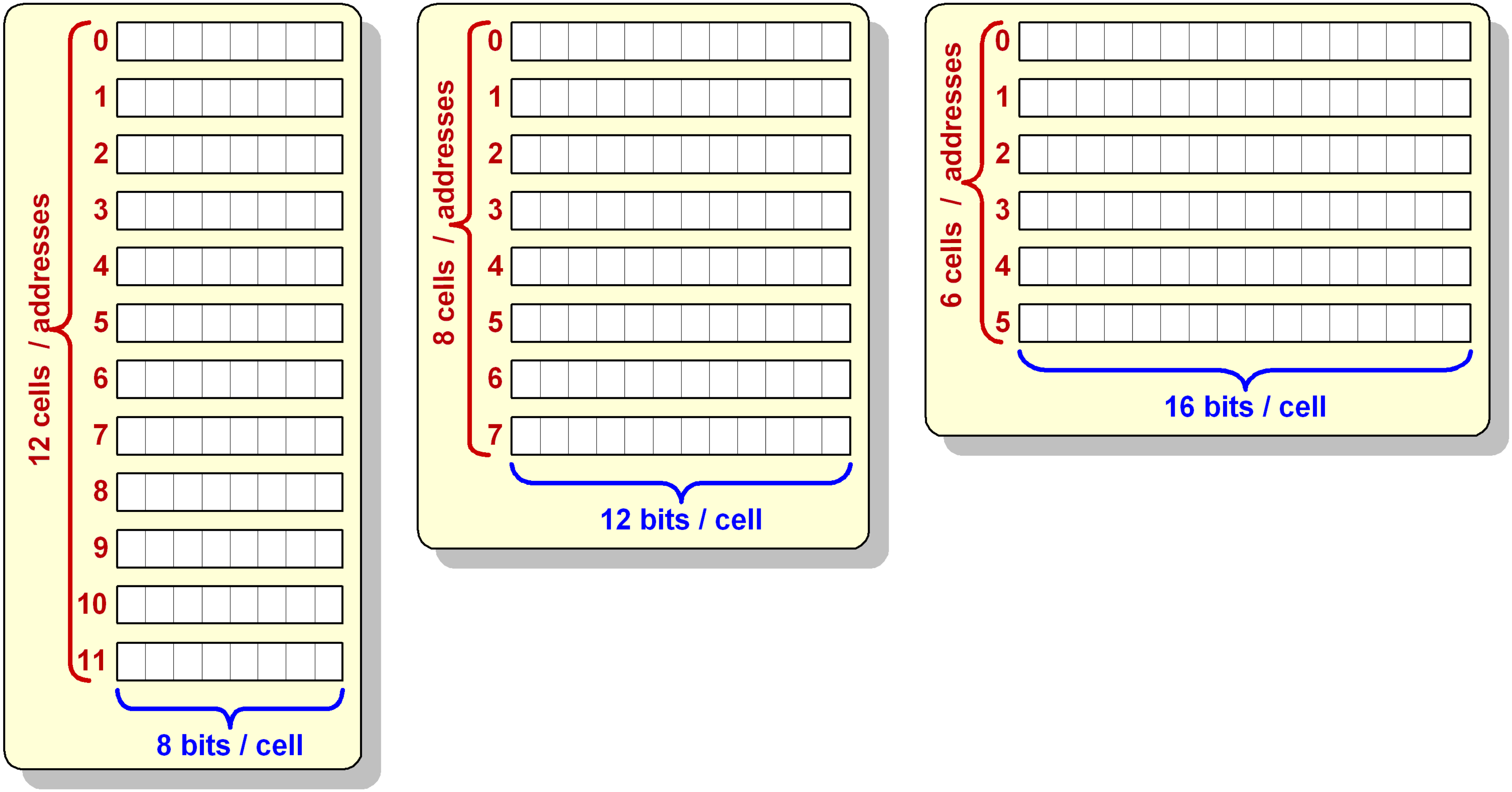
CS	RD/$\overline{\text{WR}}$	Action
0	–	Do nothing
1	0	Write data into register
1	1	Read data from register

In a system with more than one register, this arrangement allows us to use one signal to decide which register to use, and a second signal to indicate which direction the data goes.



Memory Circuits

Remember that a memory circuit is a device that holds one or more cell of data, and each cell can in turn hold one or more bits.   These are three ways to organize 96-bits of memory :



A Register is a simple memory that holds only one cell.   To build a multi-cell memory circuit we need to include addressing circuitry to select which cell to access.

Memory Circuits

This is a 4 x 3 memory circuit (4 cells containing 3 bits each). It’s similar to the figure 3-29 on page 166 of the textbook, except that:

- It uses a RD/$\overline{\text{WR}}$ signal instead of RD and OE (Output Enable)
- It uses a single set of data connections for reading and writing

These connections are the same as are used on actual memory chips.

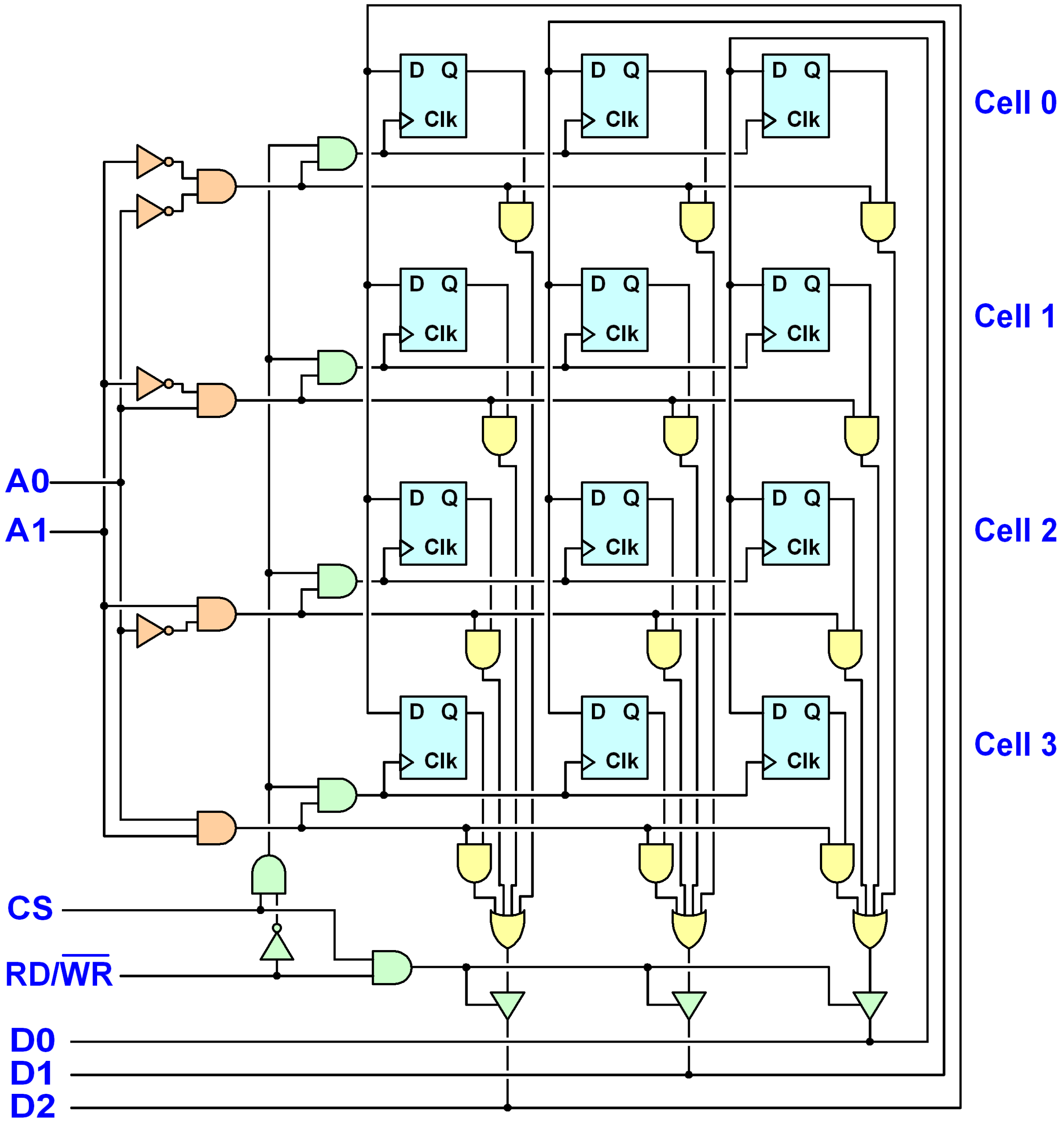
Legend:

Flip-flops containing data

Addressing Logic

Read/Write Logic

Read data path



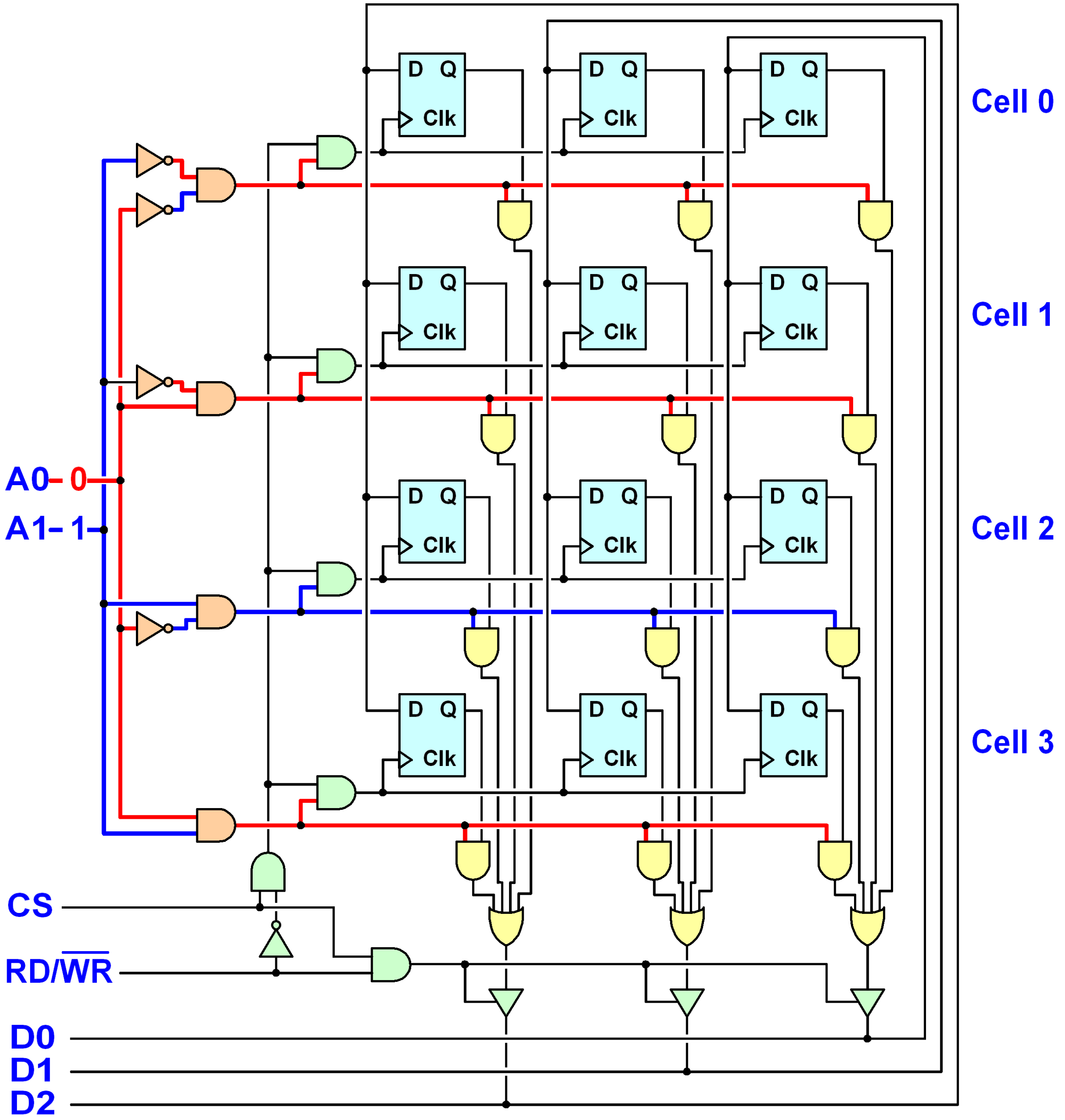
Memory Circuit - Addressing

The memory is addressed by selecting one of the four cells via the A0 and A1 input signals as follows:

A1	A0	Selects
0	0	Cell 0
0	1	Cell 1
1	0	Cell 2
1	1	Cell 3

The diagram shows FALSE signals in red and TRUE signals in blue to show how the A0 and A1 signals are decoded to activate one of the four "word select" lines (long horizontal red and blue lines).

The activated word select line can then be used by the read or write logic to load data into the flip-flops for that word or to read data from them.

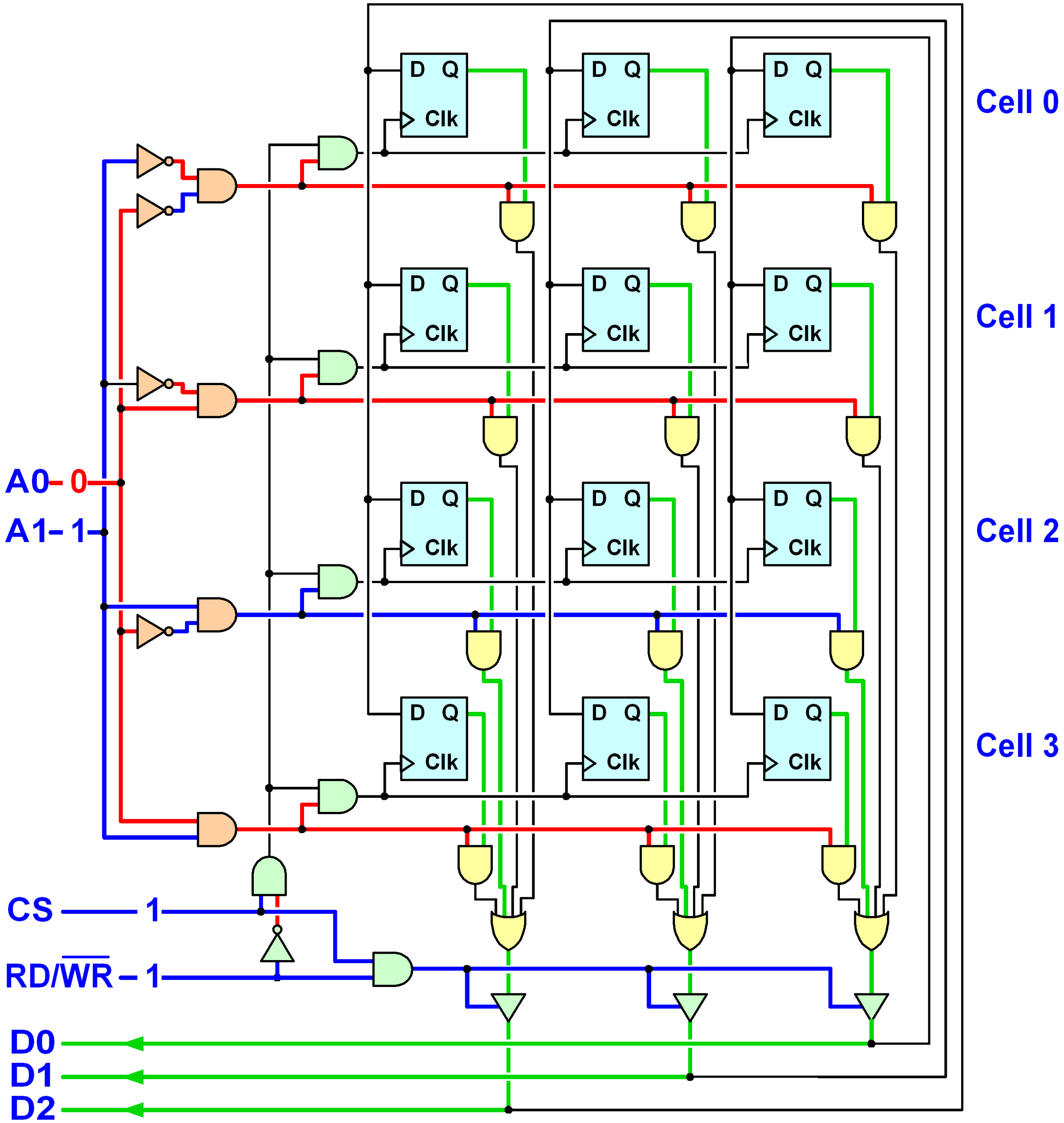


Memory Circuit - Reading

This diagram shows how the memory is read. The A0 and A1 lines select one of the cells (Cell 2 in this example) as described before.

All of the flip-flops send their data contents (shown as green lines) out over their Q lines, but these outputs are gated by an AND gate connected to each word select line. Since Cell 2's word select line is the only one set to TRUE, only Cell 2's Q outputs are passed through to the tri-state output buffers.

The tri-state output buffers are enabled by the combination of the CS and RD/$\overline{\text{WR}}$ signals, so the data from Cell 2 is connected to the D0, D1 and D2 lines and passes out of the memory circuit to other devices in the system.



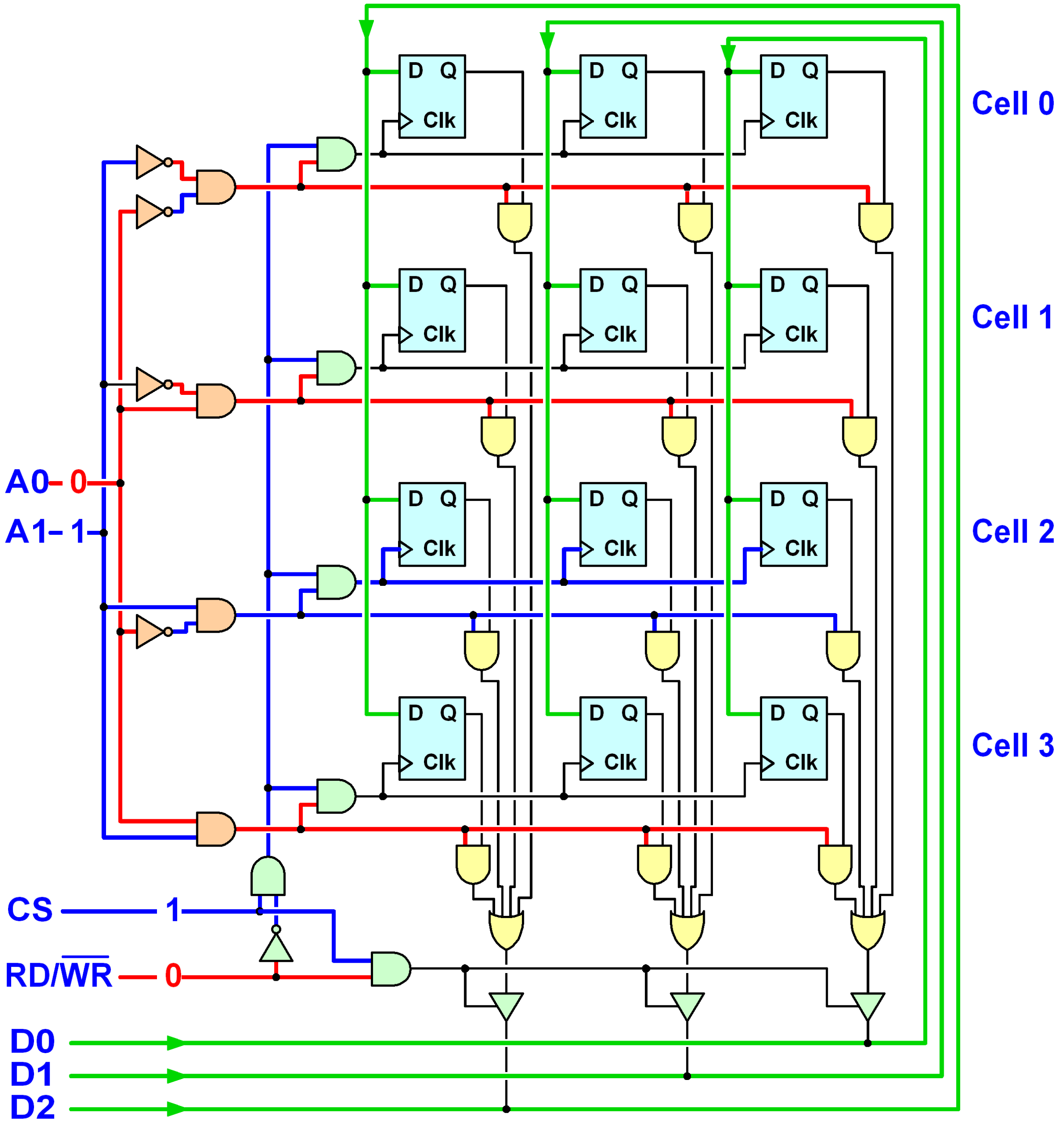
Memory Circuit - Writing

This diagram shows how data is written into the memory. The A0 and A1 lines are again selecting cell 2.

Data supplied by another device in the system enters the circuit through the D0, D1 and D2 connections and is delivered to the D inputs of every flip-flop.

The CS and RD/$\overline{\text{WR}}$ signals enable the four "write gates" attached to each of the word select lines, but only the gate for Cell 2 actually turns on because only it's word select line is true.

The output of the Cell 2 write gate is connected to the Clk inputs for each of the Cell 2 flip-flops, and so only those flip/flops actually load the data.



Exercise 7 – Memory Circuit

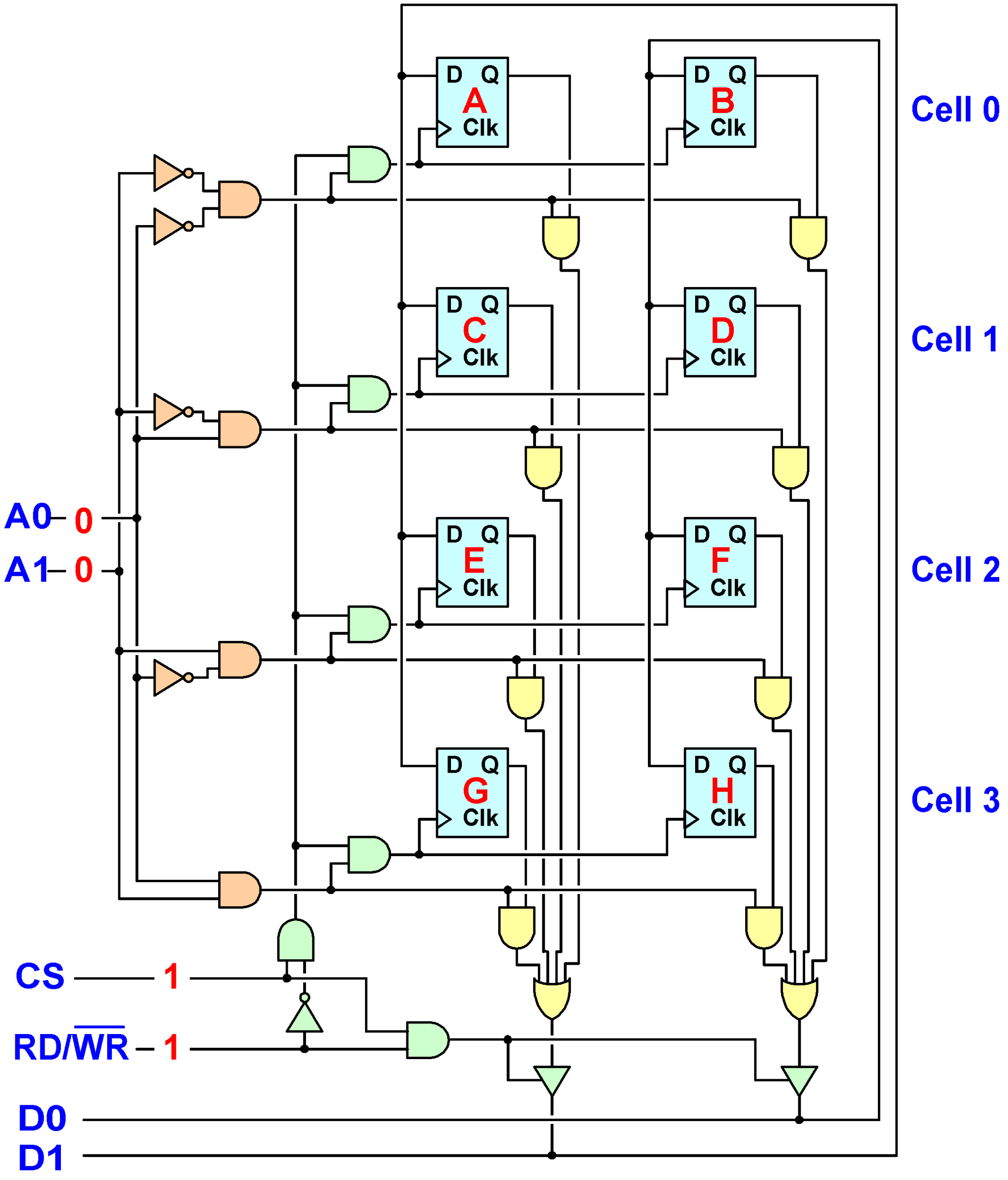
Here's a memory circuit very similar to the previous one, except that it holds only two bits per cell.

1. **Given the state of the input signals, what is the memory chip doing?**

   A reading
   B writing
   C neither

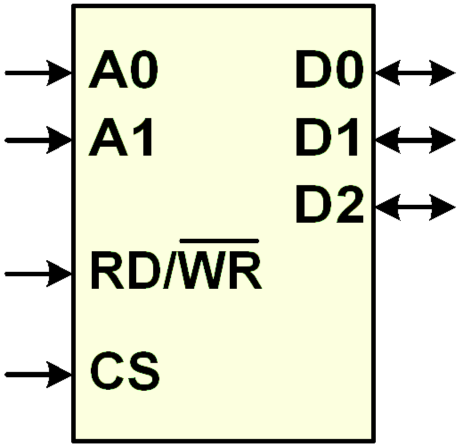
2. **With the selected address, which flip/flop is being used with the data on the "D0" connection?**

   (Select one of A thru H based on the letter in each flip-flop)



Memory Connections

The memory circuit we just examined had the connections as shown here. All memory circuits have similar connections:



Chip Select (CS) Activates the chip and requests a read or write operation. In most real memory chips this is a negative logic signal. To avoid confusion we say that the Chip Select signal is being "asserted" when the chip is activated. This signal always comes from some other device in the system.

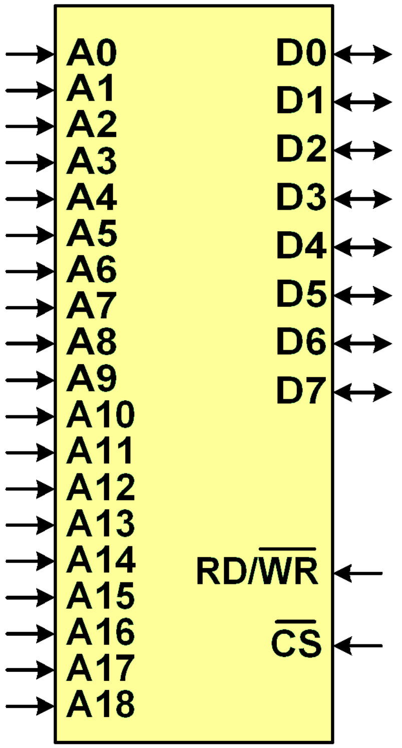
Read / Not-Write (RD/$\overline{WR}$) Used to select a write (new data is stored in the chip) or read (previously stored data is returned out of the chip) operation. This signal always comes from some other device in the system.

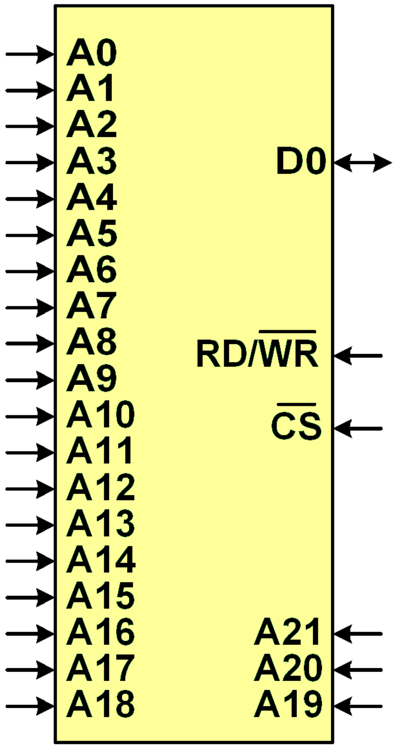
Address Lines Used to select which cell the memory chip will access. The number of address lines determines the number of cells in the chip via the 2^n rule (i.e., a chip with two address lines has $2^2 = 4$ cells). Addresses always come from other device in the system.

Data Lines Carry data into the chip when writing or out of the chip when reading. The number of data lines is the same as the number of bits in each cell. The data lines are bi-directional – data comes from some other device when writing, or from the memory chip itself when reading.

Memory Organization

As we saw, different memories with the same overall number of bits can be organized in different ways. This means they will have different connections ("pins"):





19 address lines: 2^{19} = 512K cells
8 data lines = 8 bits / cell
Memory organization is 512K x 8
512K x 8 = 4M bits total capacity

22 address lines: 2^{22} = 4M cells
1 data line = 1 bit / cell
Memory organization is 4M x 1
4M x 1 = 4M bits total capacity

A Note About Memory K’s and M’s

Memories always have capacities that are a power of 2, because memory addressing uses binary numbers. But quoted memory sizes are rounded to the nearest K (1000), M (1 000 000) or G (1 000 000 000) for convenience. For example:

No. of Address Bits	No. of Memory Cells	Rounds to:
10	1 024	1K
20	1 048 576	1M
30	1 073 741 824	1G

To round the memory capacity, repeatedly divide the capacity by 1024 and increase the metric unit you have a number less than 1024. For example:

For a memory chip with 28 address lines:

2^{28} = 268 435 456 cells

268 435 456 / 1024 = 262 144 K cells

262 144 / 1024 = 256 M cells

For this course, this rounding of powers-of-two should only be done for memory capacities, *not* for other measures such as data rates or disk sizes.

Row / Column Addressing

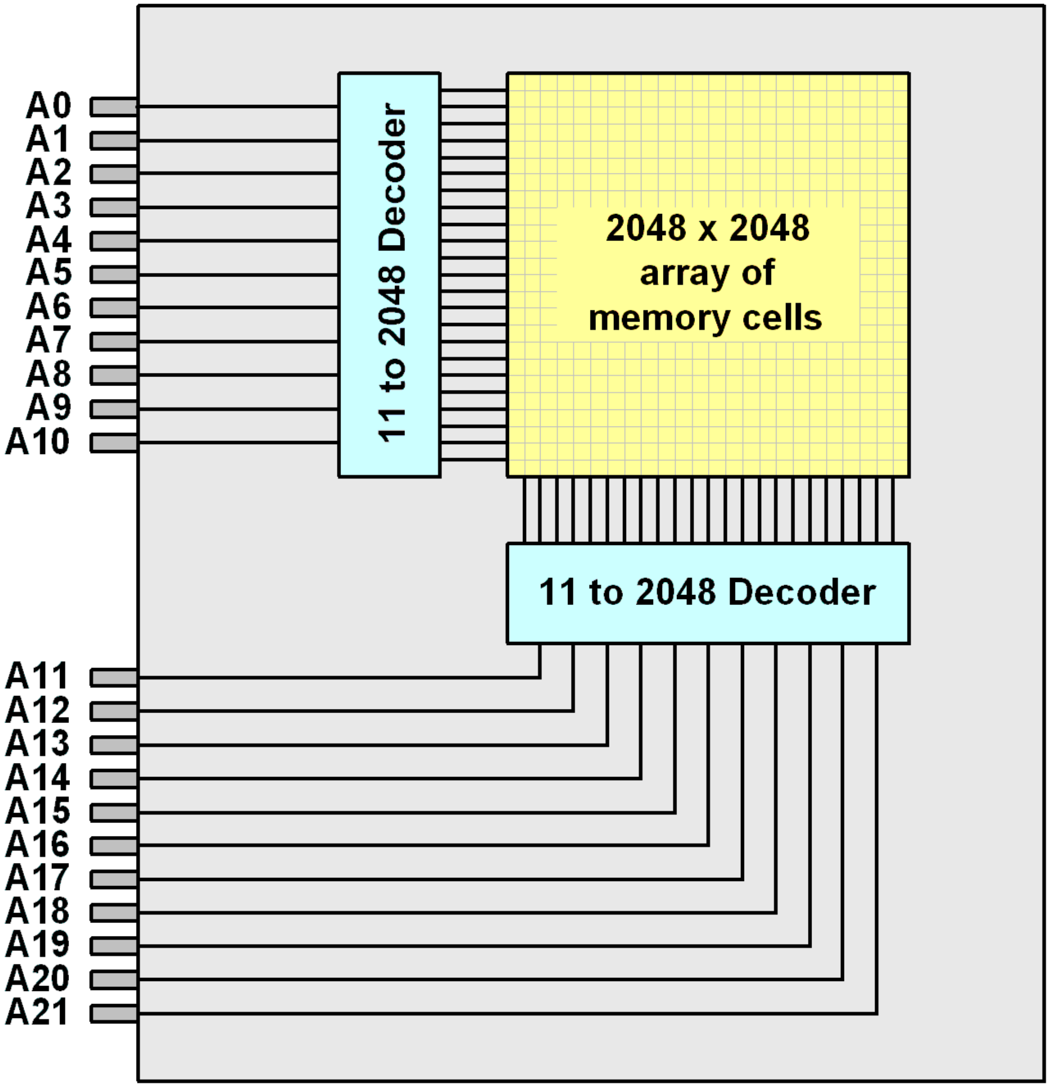
Modern high-density memory chips have hundreds of millions of addresses, and this means they need lots and lots of pins for addressing. To reduce the pin count, many chips use "Row/Column" addressing.

Row/Column addressing is used when the memory cells are arranged in a rectangular grid on the chip. The diagram at right is an example of an ordinary 4M-bit chip that has it's cells arranged in a grid of 2048 rows by 2048 columns (data and control connections are not shown).

A 4M-bit chip requires 22 address lines (because 2^{22} = 4M). In this chip, half of the address lines (11 of them) are decoded to select one of the 2048 rows that the addressed memory bit lies in (note that 2^{11} = 2048).

The other 11 address lines are decoded to select one of the 2048 columns that the addressed bit lies in.

The addressed bit lies where the selected row and column intersect.



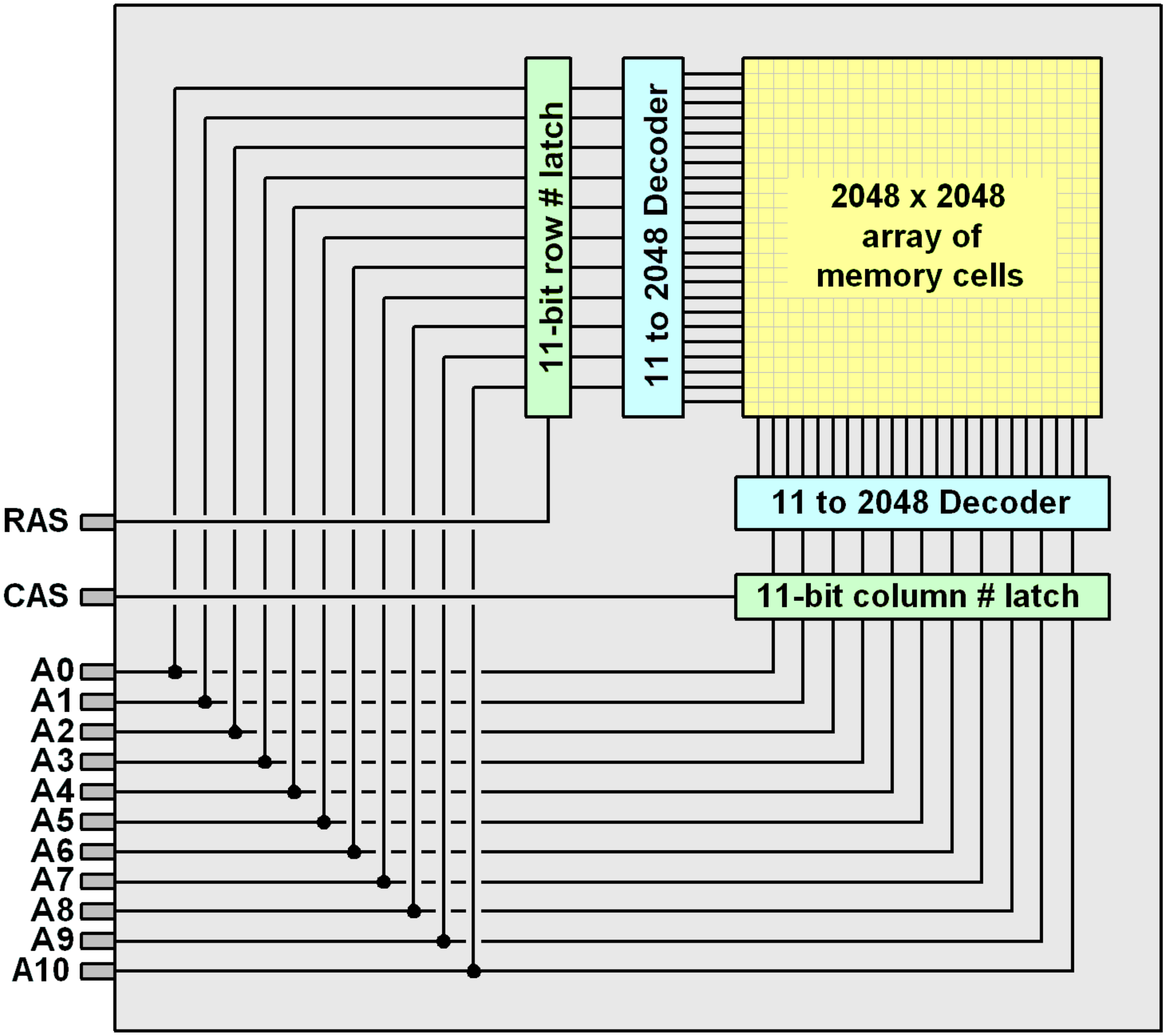
Row / Column Addressing

Chip that use "Row/Column" addressing, have a single set of pins is to load the Row address and the Column address, and they also have two additional signals that are not used on a normal memory chip:

RAS – Row Address Strobe
When triggered, signifies that the Row address is being loaded

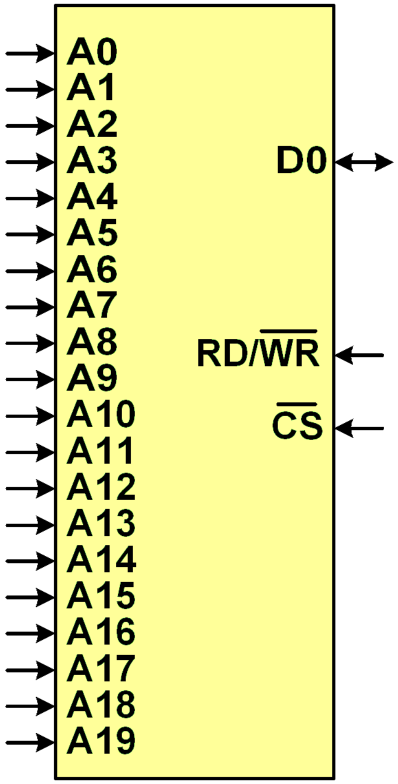
CAS – Column Address Strobe
When TRUE, signifies that the Column address is being loaded

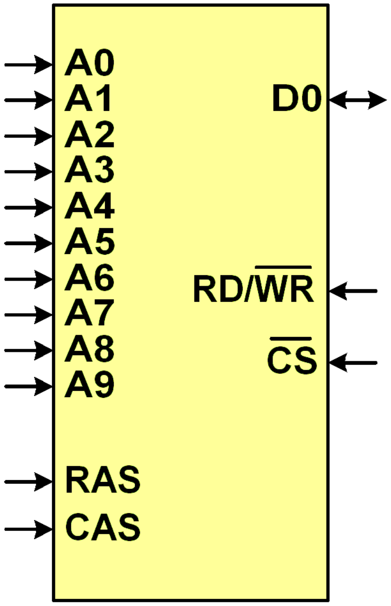
With RAS/CAS, two addressing cycles are required to load in a full 22-bit address over the 11 address pins. This cuts the number of address lines in half.



RAS / CAS Pinouts

Here's an example of the connections for two 1M x 1 memory chips, one that uses "normal" addressing and one that uses RAS/CAS addressing:





Normal Addressing:
20 address lines: 2^{20} = 1M cells

RAS/CAS Addressing:
10 address lines mean a total of 10 x 2 = 20 address bits
2^{20} = 1M cells

You can see that the RAS/CAS chip request a lot less connections, so the chip can put into a smaller package, and this means more memory can be put onto a DIMM module.

Row / Column Addressing - Performance

Accessing data in a memory chip that uses RAS/CAS addressing is somewhat slower because the address must be loaded in two pieces. But it's almost as fast as normal memory chips, because:

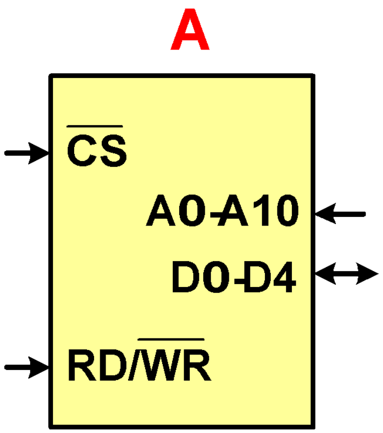
- **After the row address is loaded the row decoding takes place at the same time that the column address is being loaded. This is time that has to be spent anyway, even in a normal memory chip where the entire address is loaded at the same time.**
- **Most chips that use Row / Column addressing support "burst mode" reading, which is designed to rapidly return information from consecutive memory addresses. It works like this:**
    - **The Row address for the first memory location is loaded**
    - **The Column address for the first memory location is loaded and the data is returned**
    - **The next memory location requires a new Column address but not a new Row address. So you can skip the Row address load for the second and subsequent locations.**

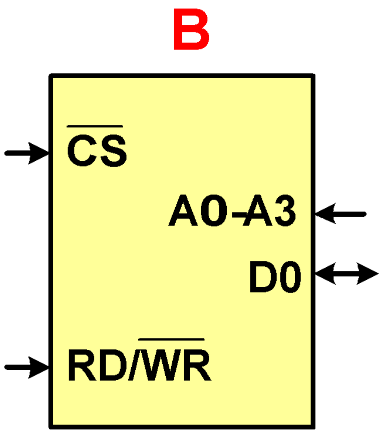
Most systems include a cache which can use burst mode to fill up an entire cache line with data read from memory.

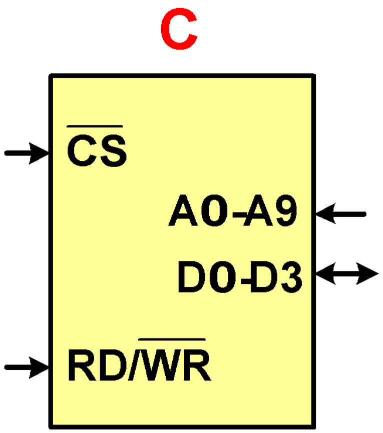
Exercise 8 – Memory Pinouts

1. Which of the memory pinouts at the right shows a memory organized as:

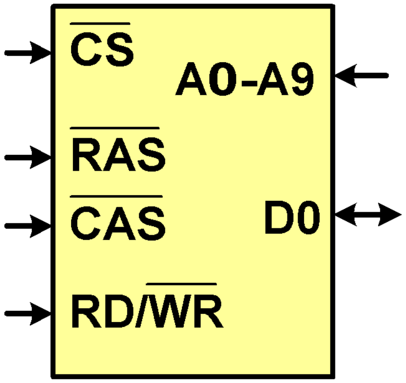
   1K x 4







2. What is the organization of the memory chip shown at the right?



A – 1K x 1

B – 1M x 1

C – 1G x 1

Types of Memory

There are two major types of main memory:

RAM – Random Access Memory

This is the type of memory most used in computer systems, and the type of memory that we've just studied. RAM memory has these characteristics:

- **Can be accessed randomly (the access time for any memory cell is (pretty much) the same as for any other cell**
- **Information can be read from or written to a memory cell**
- **Information stored in RAM memory is lost if the power is turned off.**

ROM – Read Only Memory

Most computer systems have some ROM memory, but generally in very small quantities compared to RAM memory. ROM memory has these characteristics:

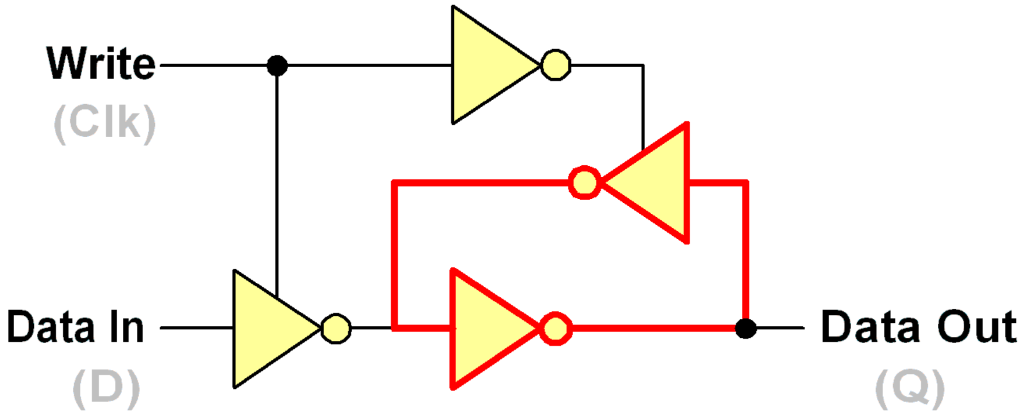
- **Can be accessed randomly (the access time for any memory cell is (pretty much) the same as for any other cell**
- **Information in a memory cell is fixed and cannot be overwritten.**
- **Information stored in ROM memory is kept even if the power is turned off.**

Note that the acronym "RAM" is a bit of a misnomer because ROM memory can also be accessed randomly.

Types of RAM Memory – Static RAM

There are two major types of RAM memory SRAM (Static RAM) and DRAM (Dynamic RAM)

Static memory is the same type of memory that we examined earlier. The storage circuits in a static memory chip are made from flip-flops. Unlike the D flip-flops we used in our memory circuits, the basic memory cell in a typical SRAM chip is built from a much simpler bistable circuit that typically requires only six transistors as shown in the diagram at the right. The equivalent connections for a standard flip-flop are shown in grey lettering.



The two inverters outlined in red form the heart of the flip-flop – they form a bistable circuit which can be on one of two states (with the data out connection being TRUE or FALSE).

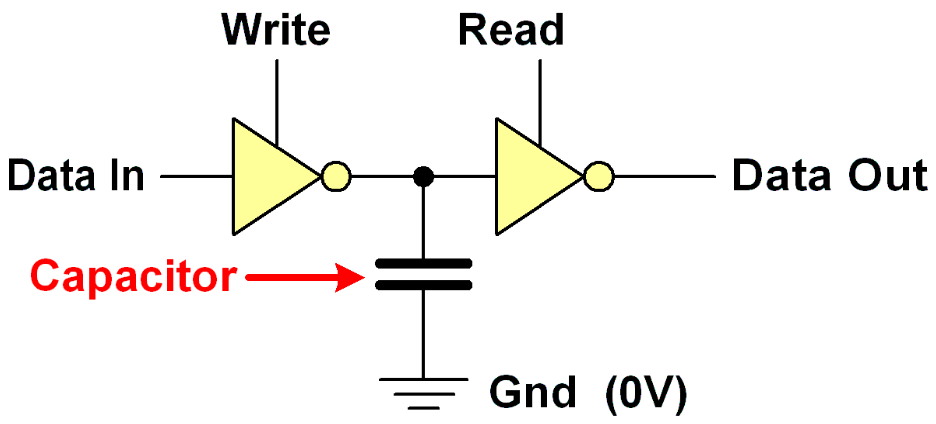
The Write signal disables the output from the top inverter and simultaneously injects the input data into the bottom inverter so as to replace the old contents with new data.

Static RAM is very fast because the contents of the flip-flop are always available at the output without delay. For this reason Static RAM is used wherever very fast access speed is important, such as in cache memory chips.

Types of RAM Memory – Dynamic RAM

Dynamic memory chips have an even simpler memory cell design which uses a capacitor as the basic storage circuit. A capacitor acts like a very, very tiny rechargeable battery. If the capacitor (battery) is "charged" then the bit contains a TRUE or "1" value, and if it's "discharged" then the bit contains a FALSE or "0" value.

This storage circuit is simpler than the flip-flop used in SRAM chips – it requires only a couple of transistors and the capacitor. This takes much less space on the chip, and the result is the a DRAM chip can hold much more information than a SRAM chip can.



Since the cost for a chip of given dimensions is roughly the same, a DRAM chip with more bits on it has a lower cost-per-bit than a same-sized SRAM chip. Thus, storage built from DRAM chips is much cheaper than storage built out of SRAM.

The problem with DRAM chips is that a "charged" capacitor which stores a "1" bit will discharge just like a rechargeable battery if it's left too long. Since the capacitors are very tiny, the discharge time is only about 15 microseconds.

To avoid loosing data, DRAM chips must be refreshed hundreds of times per second. The memory controller circuitry does this by reading the contents of every memory cell and rewriting the data back into the cell again (this is done a row at a time). If the CPU tries to access memory during a refresh cycle it is forced to wait until the cycle completes. Because of this, DRAM chips are much slower than SRAM chips are.

Types of Dynamic RAM Memory

As technology has matured there have been a number of generations of DRAM memory which have taken different approaches to improving performance. Each type uses the same basic memory cell but has different control circuitry to reduce access times:

Fast Page Mode (FPM) DRAM

Automatically increments the column address and returns consecutive bits without requiring additional address cycles.

Extended Data Out (EDO) DRAM

Continues to deliver data from the last cycle as the next cycle begins – allows the next cycle to be started quicker while the data is still being transferred from the last cycle.

Synchronous DRAM (SDRAM) [Note – SDRAM and SRAM (Static RAM) are *different*!]

Synchronizes the operation of the memory to an external clock and allows different operations (such as reads and writes) to be pipelined.

Double Data Rate (DDR) SDRAM

Faster SDRAM that synchronizes on the leading *and* trailing edges of the clock signal, effectively doubling the data rate. Has gone through various versions including DDR, DDR-2 and DDR-3. The different versions have changed the voltage levels and some of the interface protocols between the memory and the memory controller. DDR-2 or DDR-3 SDRAM is used in most current computer systems.

RAMBUS DRAM (RDRAM)

Similar to DDR SDRAM but uses a serial protocol to communicate with the memory controller. Licensed by Rambus Inc. and used by some Intel systems. Was to have provided better performance but DDR SDRAM is essentially just as fast and cheaper.

Types of Memory – ROM Memory

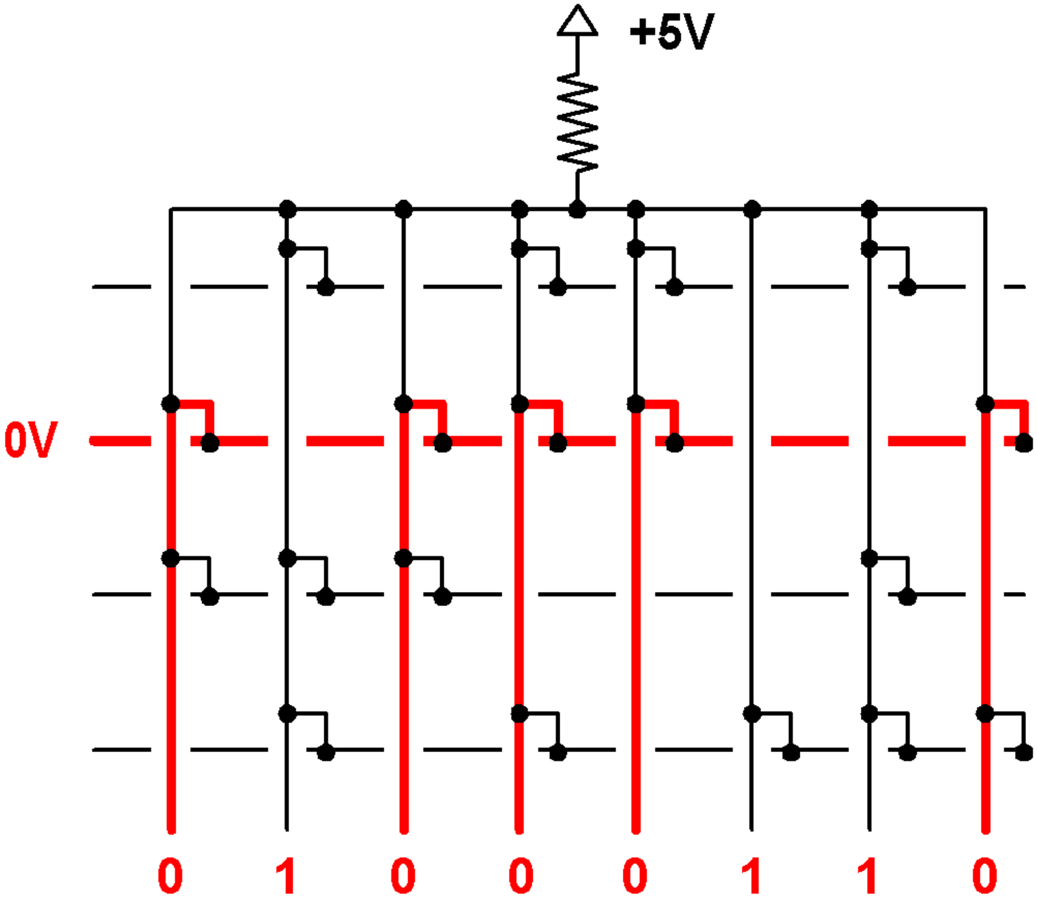
The problem with RAM memory is that it's volatile – it looses it's contents when the power is turned off. This presents a problem when a computer system is first turned on because it means that the program that the CPU is trying to execute can't be stored in RAM memory.

ROM (Read Only Memory) is used to solve the problem because it's contents are non-volatile – they are preserved even when the power is turned off. So most computer systems have at least some ROM memory to hold the program that's executed when power is first turned on.

ROM memory can't be written to, only read from. So how does the data get loaded into it?

For standard ROM chips, the data is put into the chip when it's made by the semiconductor manufacturer. The diagram at the right shows how some of the intersections in a grid of rows and columns be wired together to deliver pre-determined data when a given row is selected (it's actually just a little more complicated than this diagram suggests).

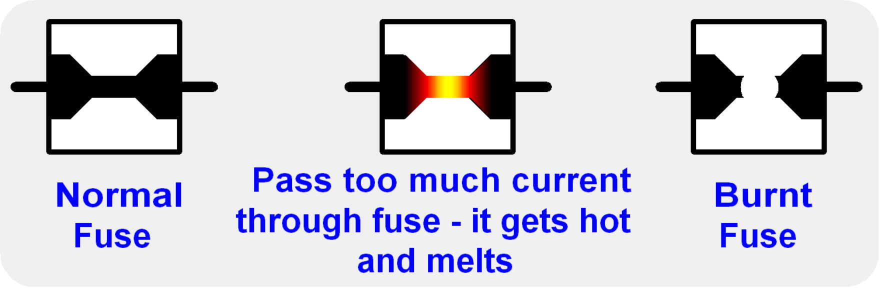
The semiconductor manufacturer produces this "wiring" by creating custom masks use during the chip fabrication process.



Types of Memory – PROM Memory

ROM memory has a very low unit cost (cost per chip), but a high setup cost (cost to produce the first chip). Designers don't want to use ROM chips until they are very sure their programs are fully debugged.

For small volumes or prototyping, PROM (Programmable ROM) chips are cheaper. "Programmable" in this case means the same thing as in a PLA (Programmable Logic Array) chip – it means the wiring of the chip can be altered by blowing little fuses inside the circuit. This is done by putting the PROM chip into a special "burner" circuit which can apply higher than normal voltages to the chip (typically 25 volts instead of 5 volts).

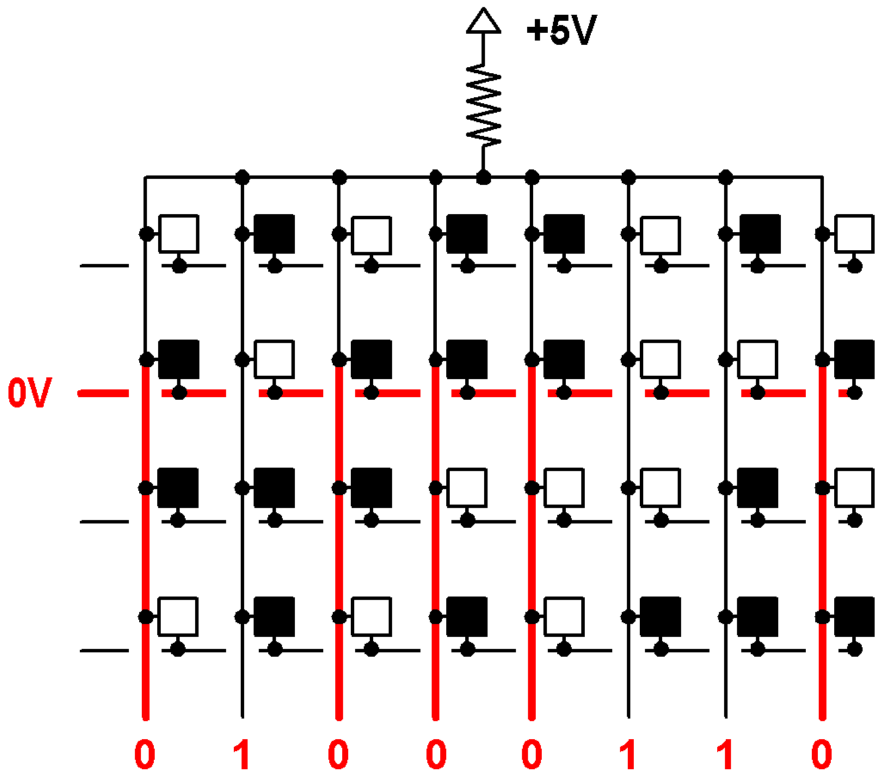


Normal Fuse

Pass too much current through fuse - it gets hot and melts

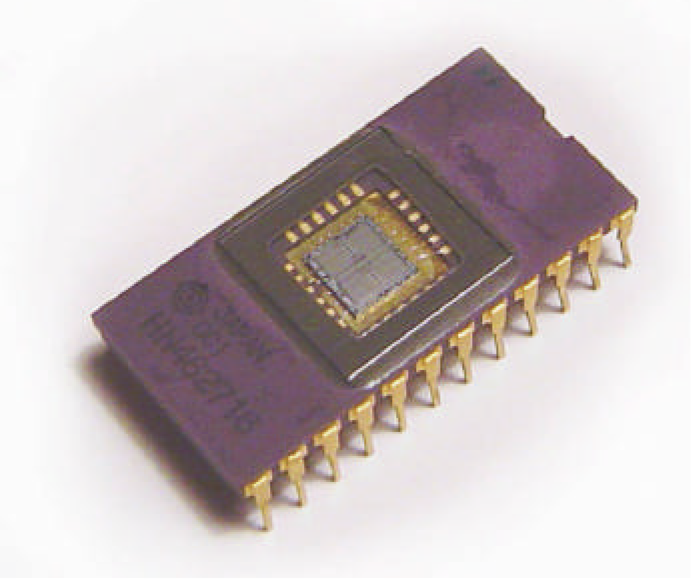
Burnt Fuse

The fuses take the place of the wired intersections of the ROM chip (shown at right using black squares for connected fuses and white squares for "burnt" fuses). Now instead of having to design a custom mask and chip, the user can buy a standard chip at a slightly higher unit cost, but with no up-front setup cost (other than for the cost of the PROM burner).



But once burnt, a PROM chip can't be changed.

Types of Memory – EPROM and EEROM Memory



EPROM (Erasable PROM) chips are a reusable alternative to PROMs. EPROM chips store 1 and 0 bits in using an electrostatic charge which can be broken down by exposure to ultraviolet light. Information is stored in an EPROM chip using a special programming circuit much like a PROM. But after the chip is "burned" it can be erased by putting it under a strong ultraviolet lamp for 10-20 minutes. EPROM chips have a quartz "window" to allow the ultraviolet light to reach the chip.

EPROM chips are more expensive than PROM chips, but they are very useful during the design and debugging stage of programs that have to be stored in read-only memory.

The last type of ROM is known as EEROM (Electrically Erasable ROM) or EAROM (Electrically Alterable ROM). These are non-volatile ROM chips that have been designed so that their contents can be erased by special control signals while in place in the computer system without having to put them into a special programming circuit. EEROM also permits individual bytes to be erased and reprogrammed, unlike other PROM memory types.

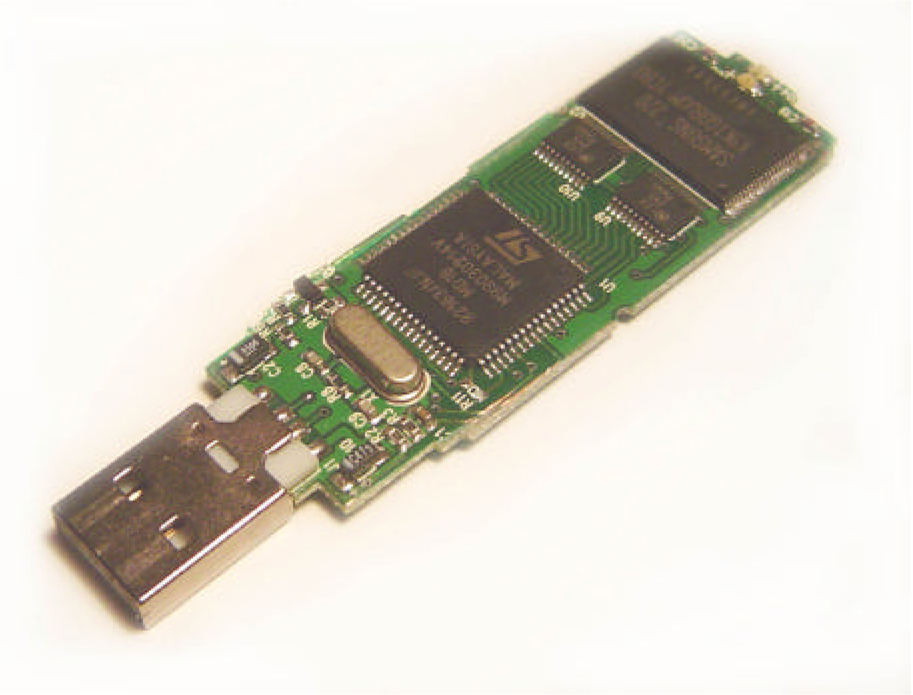
But EEROM is much slower, more expensive, and smaller than RAM and so it's only used when it's ability to retain information with no power is very important. EEROM is often used to store configuration information which doesn't have to be changed very often. As an example, the BIOS configuration settings for computers are often stored in EEROM memory.

Types of Memory – Flash Memory

Flash Memory is a special type of EEROM chip which can write information to an entire block of memory at the same time – this makes it much faster than byte-at-a-time EEROM memory.

Flash memory chips typically contain a block of ordinary RAM memory. In order to write data, the data is first loaded into the block of RAM memory and then the contents of the entire block are "flashed" into the nonvolatile memory circuits.

Flash memory has become very popular for the storage of digital pictures and music files in portable devices. "USB Flash Drives" have replaced the floppy disk as a way to manually transfer files between computers. Flash memory is also replacing ROM chips in many devices where it's important to be able to update the programming ("flash BIOS", for example).

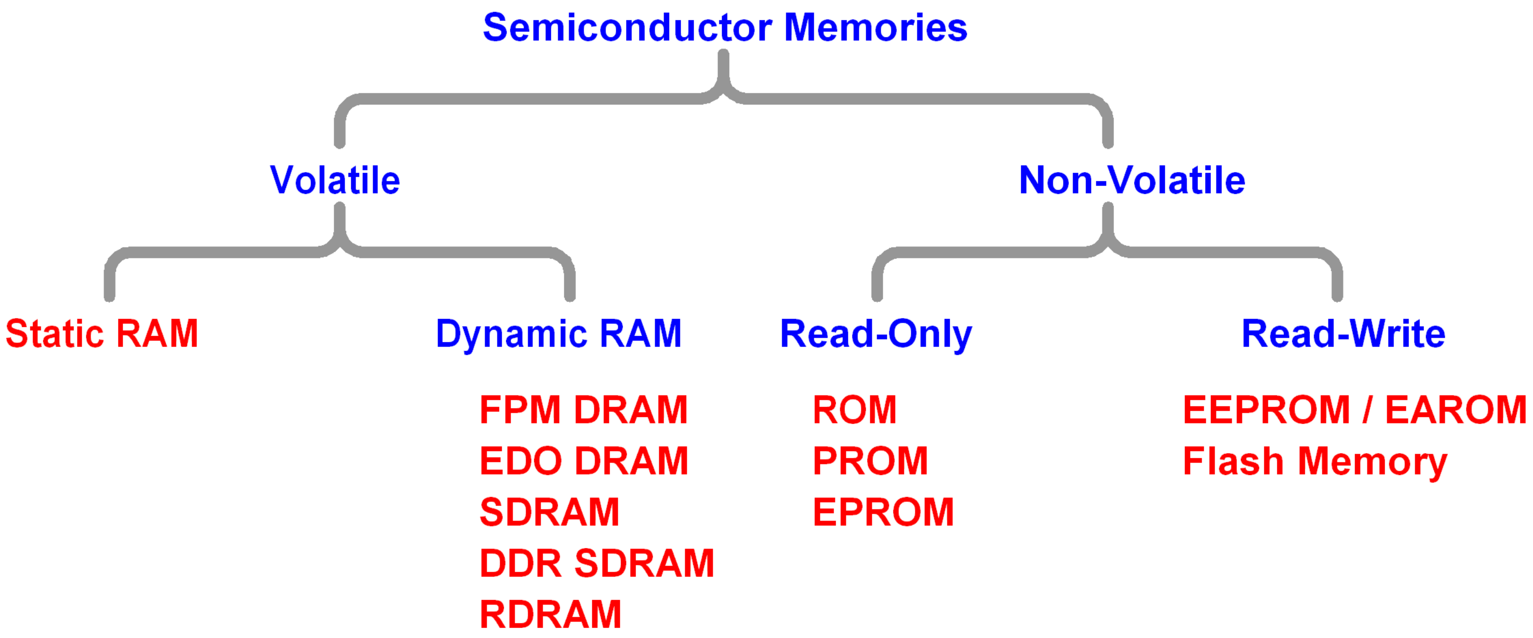


Internal view of USB Flash Drive

Flash memory (and EEPROM memory too) has a limited lifetime – after a certain number of write cycles (typically 10,000 to 1,000,000 writes) the memory will stop accepting new information. Modern flash memory chips use "wear leveling" which spreads writes around to different areas to help reduce this as an issue.

Semiconductor Memory Taxonomy

Here's a "family tree" and summary of the semiconductor memories we've just discussed:



Semiconductor Memories

Volatile

Non-Volatile

Static RAM

Dynamic RAM

FPM DRAM
EDO DRAM
SDRAM
DDR SDRAM
RDRAM

Read-Only

ROM
PROM
EPROM

Read-Write

EEPROM / EAROM
Flash Memory

Type	Category	Erasure	Byte Alterable	Volatile	Typical Use
SRAM	Read / write	Electrical	Yes	Yes	Cache
DRAM	Read / write	Electrical	Yes	Yes	Main Memory
ROM	Read only	Not possible	No	No	High volume
PROM	Read only	Not possible	No	No	Low volume
EPROM	Read mostly	UV Light	No	No	Prototyping
EEPROM	Read mostly	Electrical	Yes	No	Config data
Flash	Read / write	Electrical	No	No	Digital film

Key Concepts

- Construction and operation of a latch
- S-R latches vs. D latches
- Edge vs. level triggering
- How a pulse generator works
- How Tri-State Buffers are used to connect outputs
- Connecting flip-flops to form a memory circuit
- Creating a memory circuit using flip-flops
- Memory chip pin connections
- RAS/CAS addressing
- Memory types – Static vs. Dynamic RAM,   RAM vs. ROM
- Types of ROM memory

What’s Next

- Look at Review Questions for this week
- Sign on to WebCT and do Module 4 – “Display & Printer Technology”
- Study for Quiz 3, which covers:
  - ➢ Week 4 lecture
  - ➢ WebCT module 4
- Complete Assignment 1 for hand-in next week
- Pre-read week 5 material