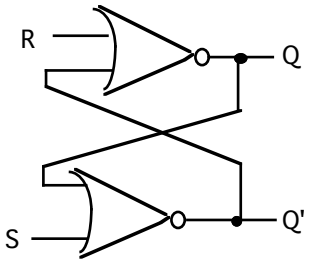
# CS240 Laboratory 5
# Sequential/Memory Circuits

**Basic Memory Circuits**

**Latch** Single-bit memory, level-triggered

**Flip-Flop** Also single-bit, but edge-triggered

**SR (Set Reset) Latch**




| S | R | Q | Q′ |
|---|---|---|---|
| 0 | 0 | Qp | Qp′ remember |
| 0 | 1 | 0 | reset (clear) |
| 1 | 0 | 1 | set |
| 1 | 1 | unpredictable | |

What does **unpredictable** mean? Notice in a NOR gate, if either input = 1 to a gate, its output = 0 (**1** is a deterministic input)**:**

| A | B | (A+B)′ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

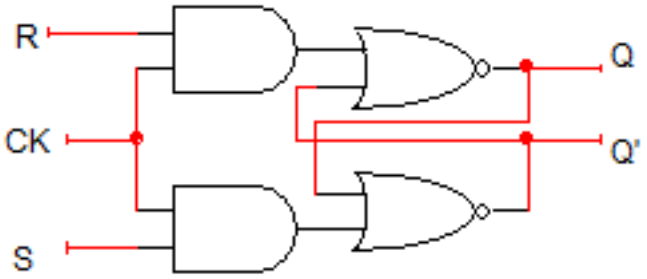
So, although you wouldn't usually try to *set* and *reset* at the same time (it doesn't make sense), if you did, Q and Q′ will both be 0 (which is not unpredictable).

However, when you go back to the *remember* state (S=R=0), Q and Q′ will not stay at 0 0. The circuit passes through one of either the *set* or *reset* state on its way back to the *remember* state, and Q and Q' change to the complement of one another.

Since the final state depends on which transitional state was sensed on the way back to *remember*, you cannot predict whether the final state of Q will be 1 or 0.

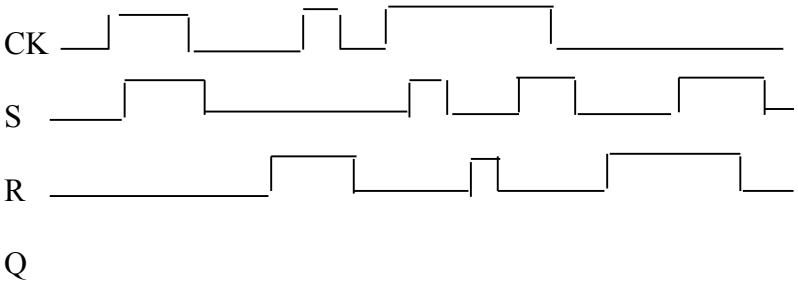
**Clocked SR Latch**

Incorporates a clock input/level-sensitive



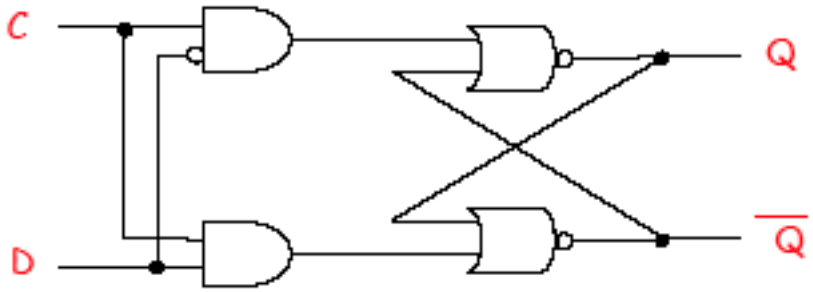

Output Q can change in response to S and R whenever the CK input is asserted.

How does Q respond to the following inputs?



**D Latch**

Avoids unpredictable state, because a single input D determines the next state of the circuit.

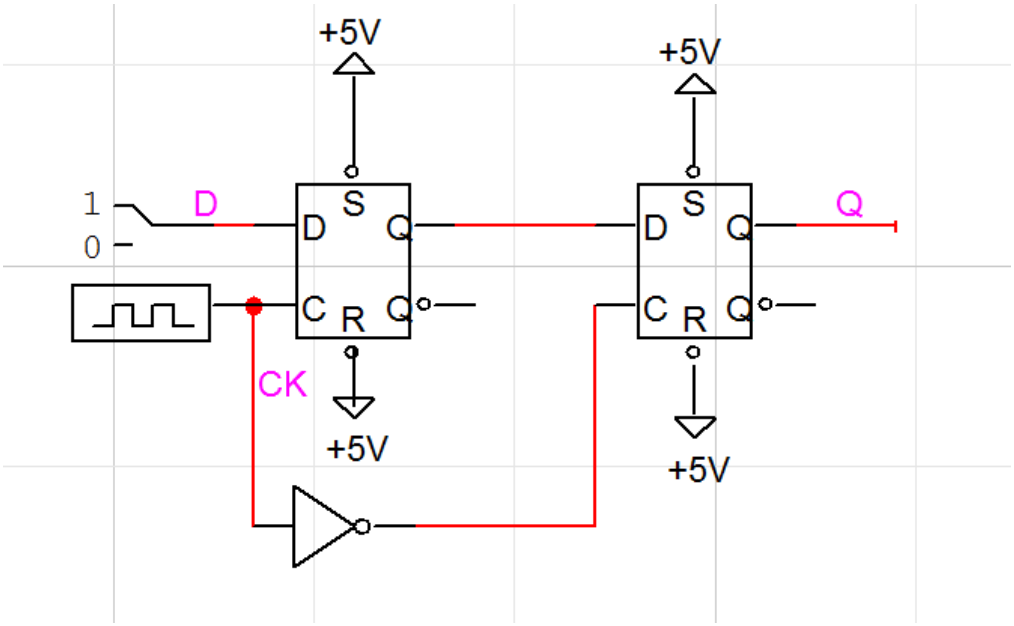



| D | Qnext |
|---|---|
| 0 | 0 |
| 1 | 1 |

**D Flip-Flop**

Changes state on a clock transition (edge), rather than whenever the clock is asserted.

Internally, a flip-flop is made from 2 latches. The first latch is controlled by the clock, but the second latch is controlled by the *inverse* of the clock:

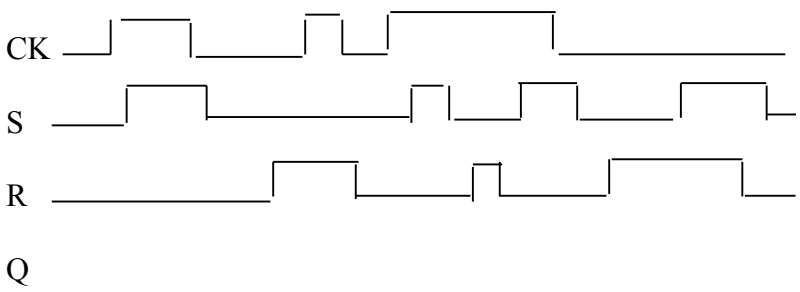



So, the input D will not be passed from the first latch to the second latch until the clock goes low.

Once the clock is low, a new value on D will not store into the first latch. Overall, the flip-flop can change value only *exactly* at the transition of the clock from high to low.

Output Q can change in response to S and R only on the positive edge of the clock.

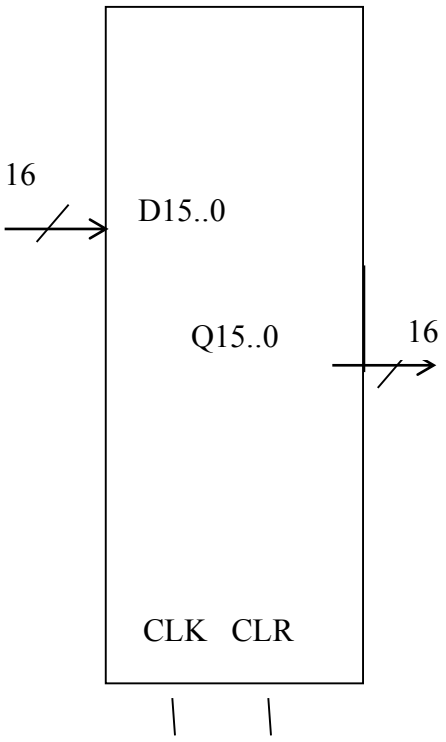
How does Q respond to the following inputs?



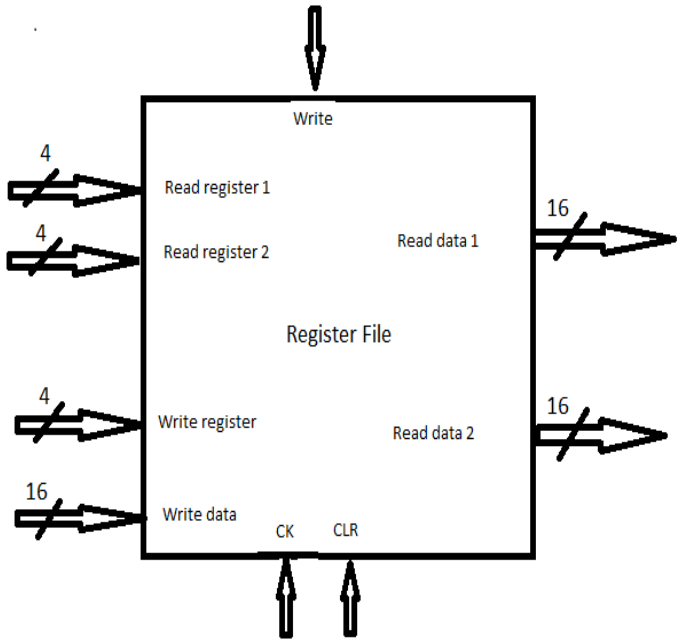

Notice the difference between Q and the output for the earlier clocked latch example.

## Circuits using Flip-flops

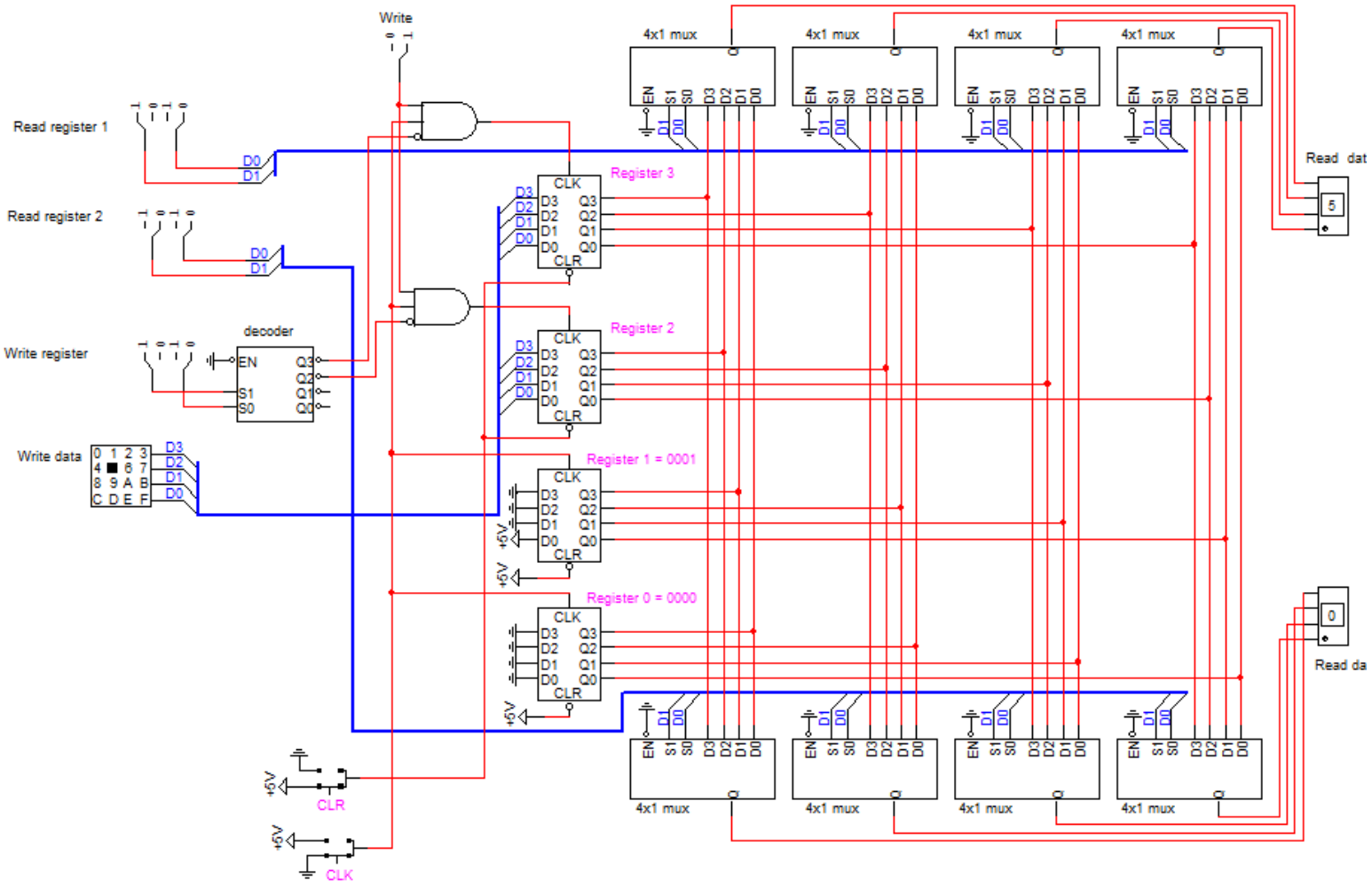
**Register -** n-bit memory, uses *n* flip-flops, and shared *clock* and *clear* inputs



**Register File** set of registers



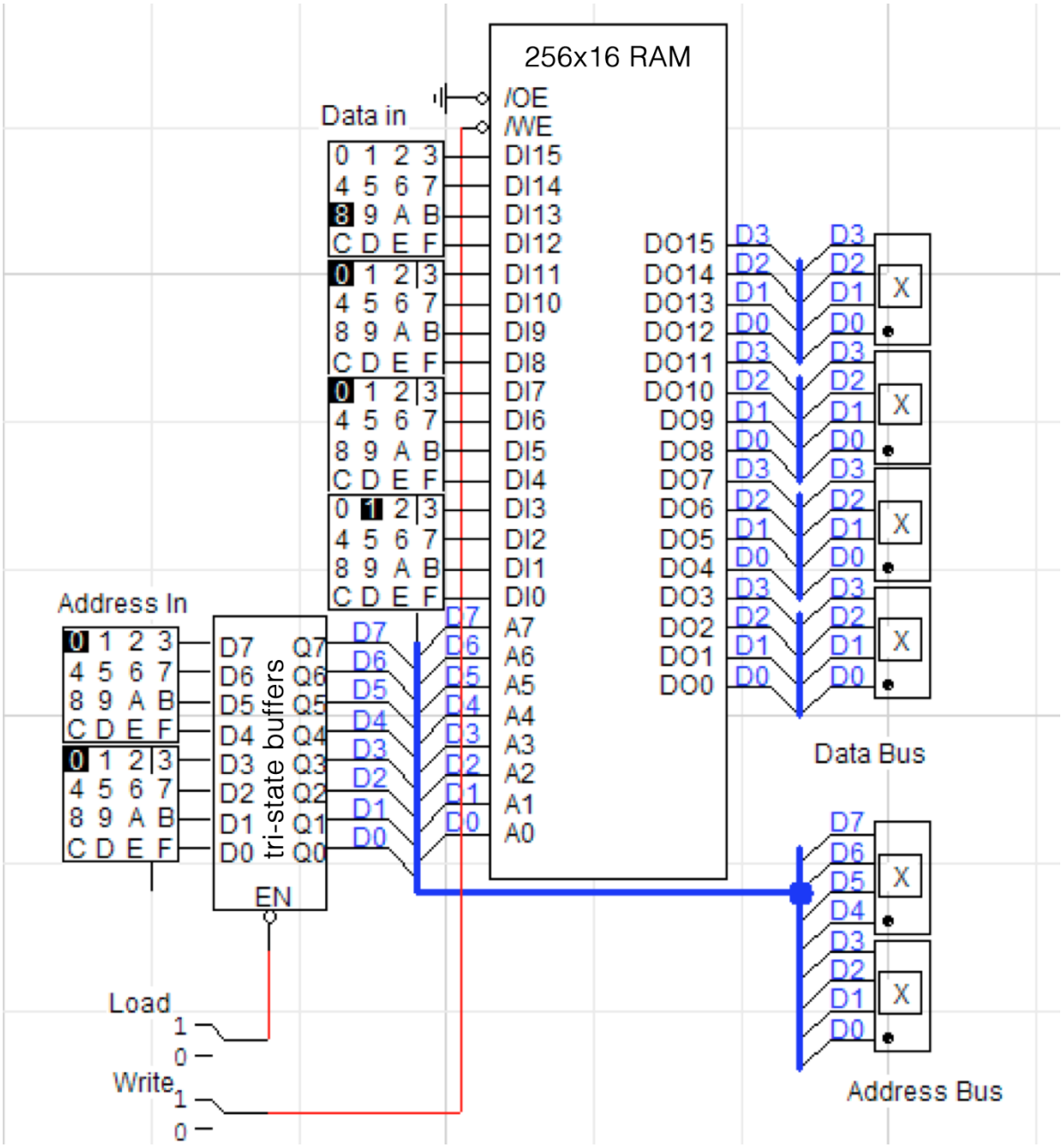

- **Write** is the write control signal.
- **Write register** is the number of a register to be written with a new value
- **Read register 1** and **2** indicate which 2 registers can be read at data ports **Read data 1** and **Read data 2** at any given time
- clear and clock (**CLR** and **CLK**) are shared by all the 16 registers.




- 2 sets of 4 x 1 multiplexers select which 2 registers are currently being output at the two read ports.

- A decoder uses the write register number to select which of the 4 registers will receive a new value on a write.

**RAM memory** contains multiple flip-flops, organized into n-bit words, where each word can be accessed through use of an address:

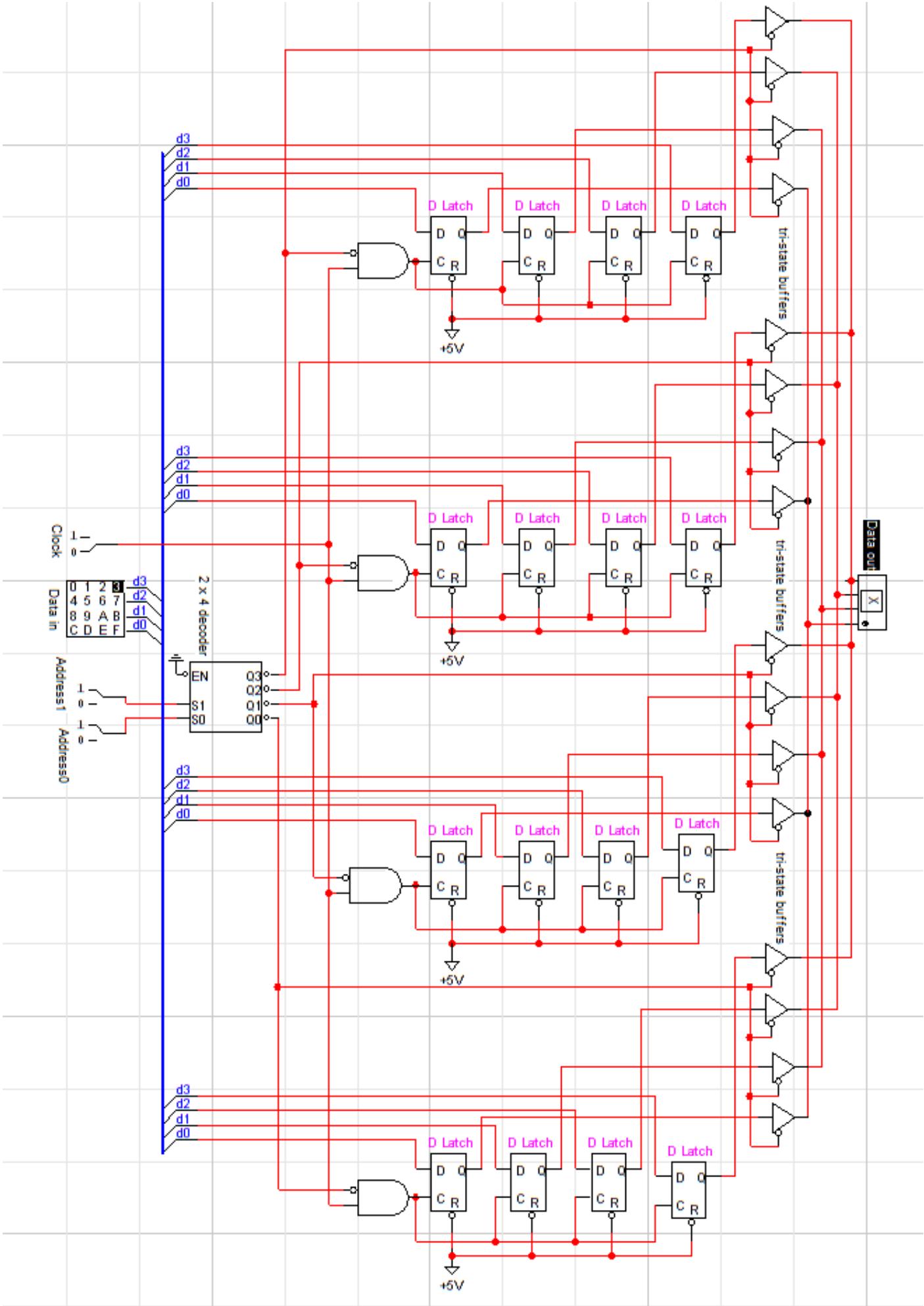



Set Load = 0 (enables tri-state buffers so that address lines are connected to memory)

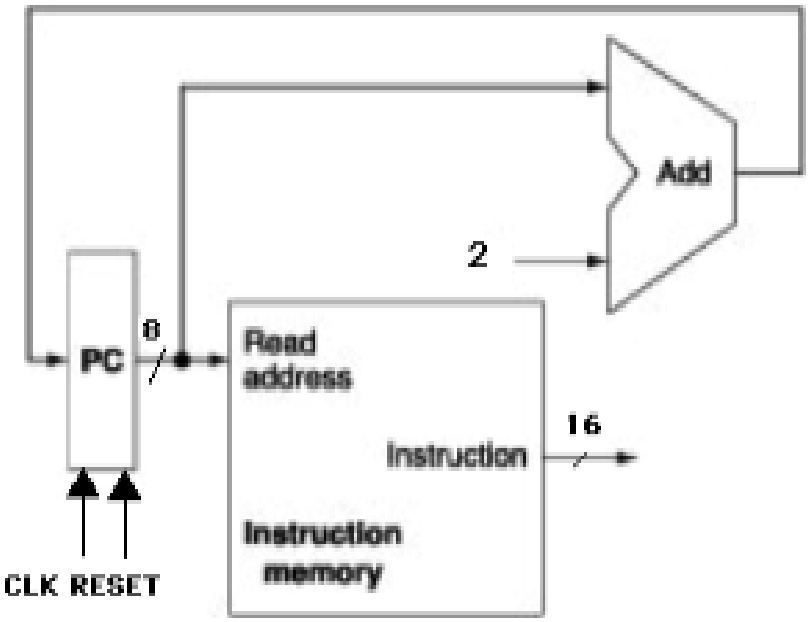
Set Addressin to location in memory

Set Datain to value to be stored at specified address

Set Write = 0 (data is stored in memory), then back to 1


d3
d2
d1
d0
D Latch
D Q
C R
+5V
tri-state buffers
Data out
X
Clock
Data in
0 1 2 3
4 5 6 7
8 9 A B
C D E F
2 x 4 decoder
EN
S1
S0
Q3
Q2
Q1
Q0
Address1
Address0

## Instruction Fetch




1. The **Instruction memory** contains 256 8-bit values (bytes). Each byte has a unique 8-bit location, or **address**.

2. The Program Counter register (**PC**) is initialized to address 0 by a **reset**.

3. Address 0 is always the address of the first **instruction** in the program, which is stored in the Instruction memory. Each instruction is 16 bits in length, so it takes 2 bytes to store an instruction.

4. Address 0 is input to Instruction Memory, and the instruction stored at address 0 is produced at data outputs

5. The value of the PC is also an input to the adder, which adds 2 to the address to calculate the address of the instruction in memory (since the instruction is 2 bytes long, you need to move to an address 2 bytes away to get to the address of the next instruction in memory).

6. The output of the adder is applied to the inputs of the PC, so that when the PC is **clock**ed, it becomes the next address applied to the memory (and the next instruction stored in memory is then produced at the data outputs).

7. So, every time the PC is clocked, the next sequential instruction in memory is produced at the data outputs.