

Memory

Computer Science 240

Laboratory 7

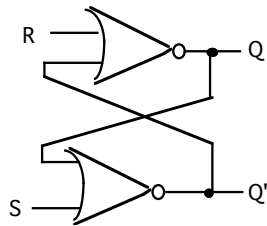
In lab today, you will design and/or investigate the behavior of memory circuits. Several circuits for use in this lab have been made available in the Lab Google group.

ALU

Exercise 0: Complete Exercise 6 from Lab 6, and verify that your circuit produces the results you predicted on your lab assignment for today.

SR Latch

Exercise 1: The SR latch is the basic unit of memory in most modern computers. In lecture, you were given an SR latch consisting of 2 cross-coupled NOR gates:



What is the truth table for this circuit?

S	R	Q
0	0	Q_0
0	1	0
1	0	1
1	1	Non-deterministic

Build the circuit on your protoboard. Use two DIP switches as inputs to S and R, and two indicator lamps as outputs Q and Q'. Verify the circuit by first setting S = 1 and R = 0, then setting S = 0 and R = 1.

Now set both S and R = 1. What happens to Q and Q'?

Q and Q' both = 0, because the NOR gates produce a 0 if either input is a 1.

Switch S to 0 and back to 1. Do the same for R. Explain what happens:

When one of the input goes to 0, the outputs become the complement of one another again (either a set or reset state). As soon as both inputs are 1 again, the outputs are again both 0.

Switch S to 0, then R to 0. What are the values for Q and Q'?

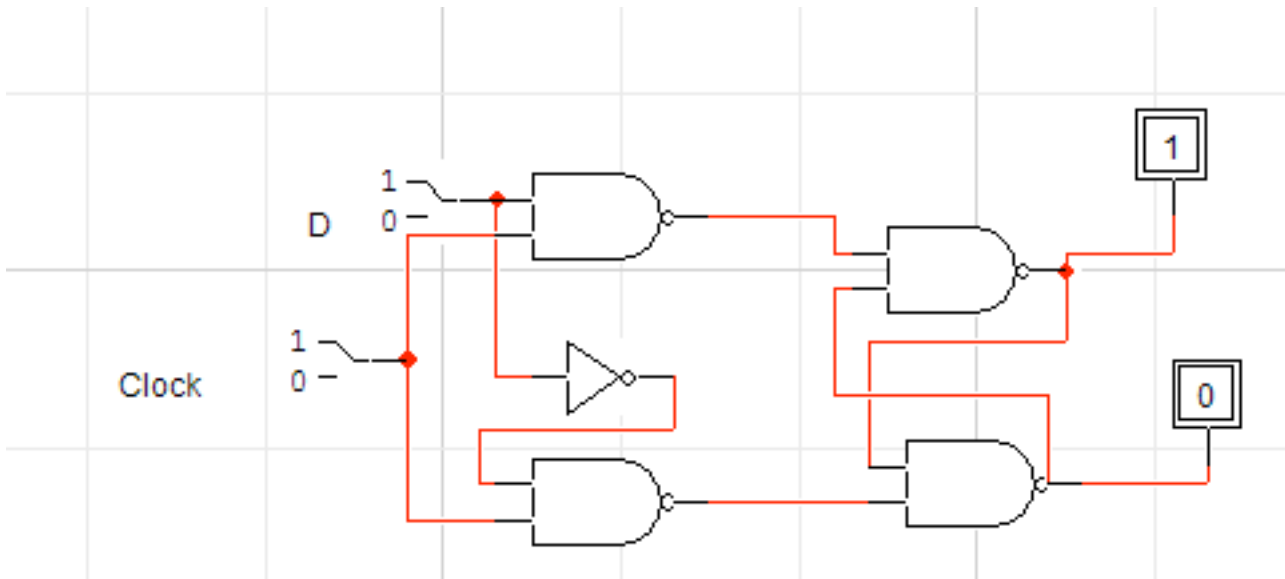
The circuit remembers the transitional state S = 0 R = 1, so the circuit ends up with Q = 0 Q' = 1

Go back to S = R = 1. Now, switch R to 0, then S to 0. What are the values for Q and Q'?

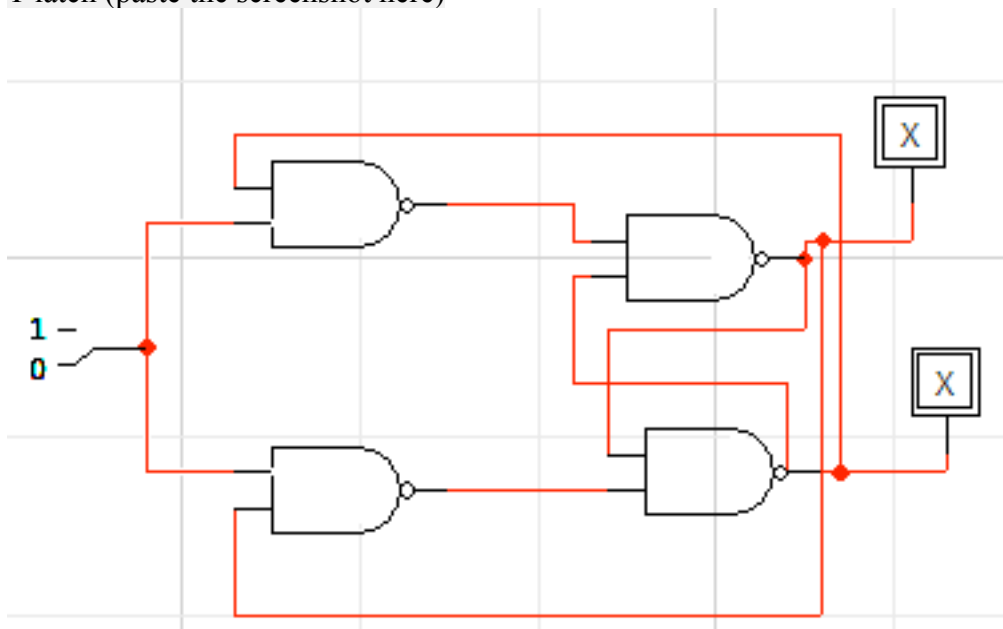
the same time!). One way to avoid this problem is to use a D latch (where only one input is used, to either set or reset the output), or a T latch (where the latch merely toggles to the opposite value of the previous output).

Use your LogicWorks clocked SR-latch from the last exercise . Show how to modify it to make it act as a:

D latch (paste the screenshot here)



T latch (paste the screenshot here)



Keep the circuit open for the next problem!

JK Latch

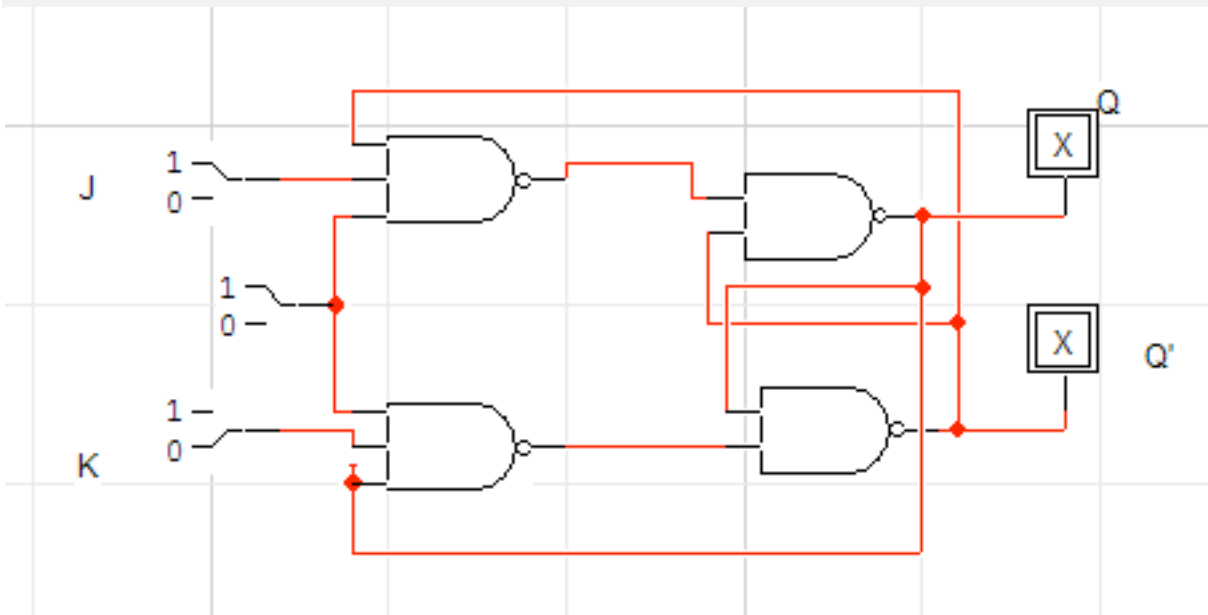
Another way to avoid the non-deterministic state of an SR is to use a JK latch, instead.

For a JK latch, the input combination of $J = K = 1$ causes the output of the latch to *toggle*, to the opposite state of the previous value, avoiding the troublesome unpredictable state of the SR latch.

Give the truth table for a JK latch:

J	K	Q
0	0	
0	1	
1	0	
1	1	

Modify the LogicWorks clocked SR-latch from the earlier exercise to make it act as a JK latch (paste the screenshot here):



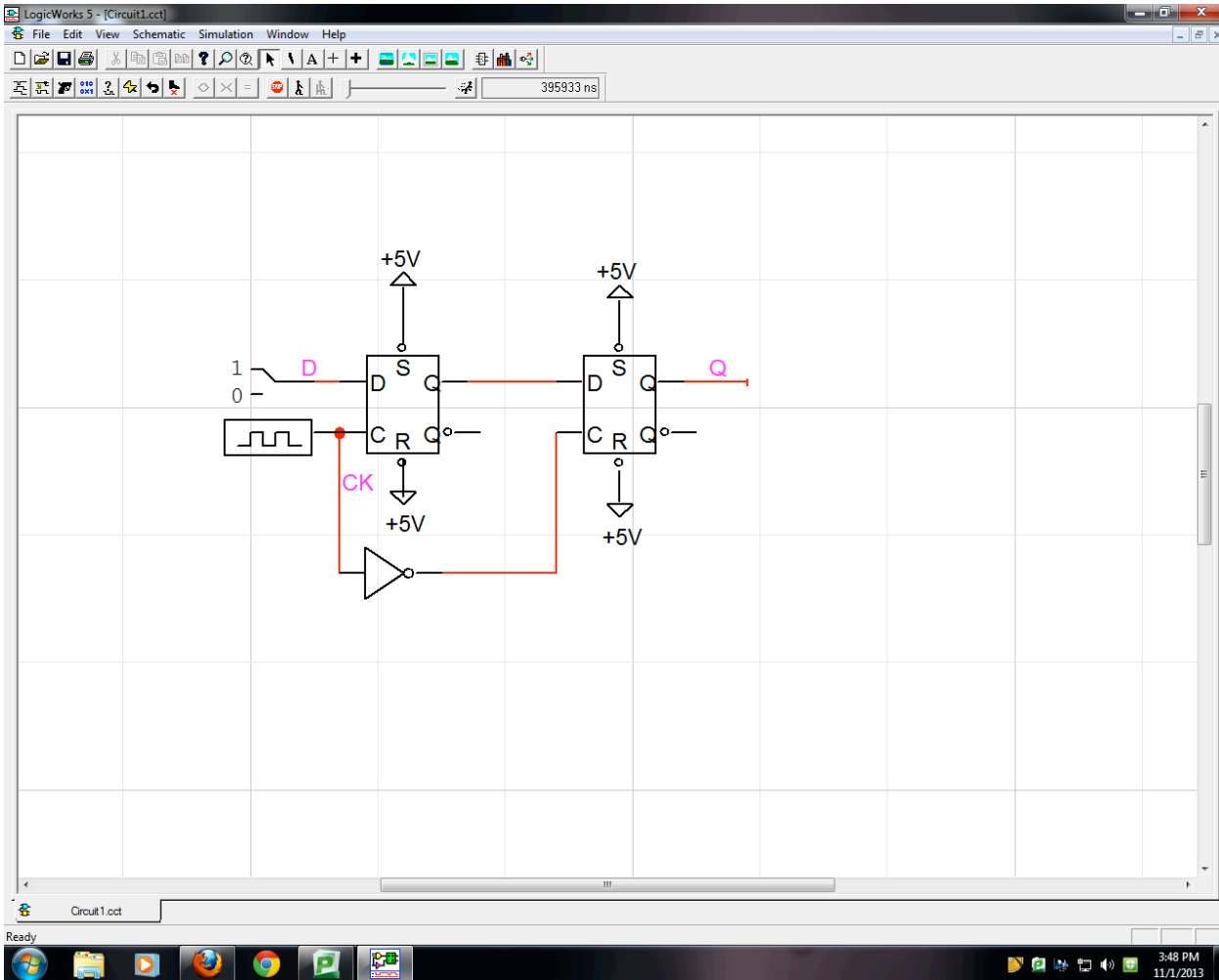
Explain what makes the outputs toggle when $J=K=1$:

The outputs toggle because, similar to the T latch, the outputs Q and Q' are opposite feedback inputs to the first set of and gates. When J and K are both 1, the opposite of the previous state acts as the controlling inputs to the first set of gates (and so toggles the outputs from the previous state).

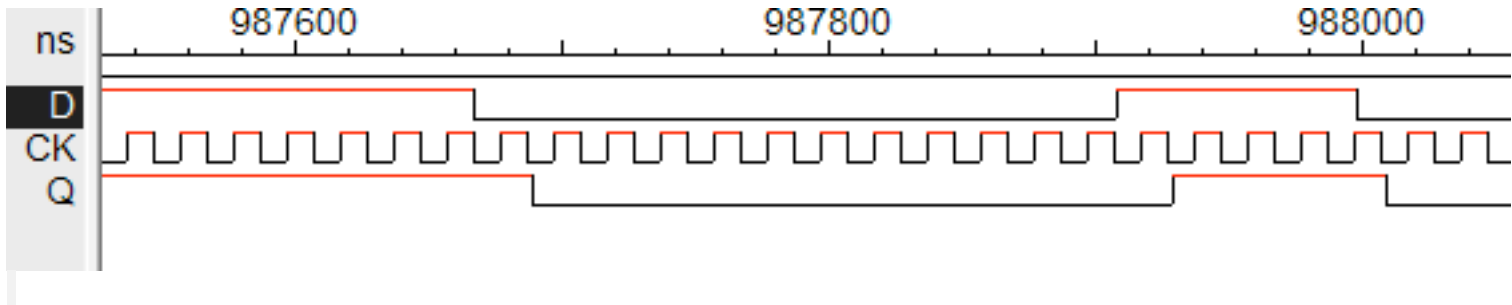
Flip-Flops

Exercise 4: As mentioned in lecture, the difference between a latch and a flip-flop is that a latch is *level-triggered*, while a flip-flop is *edge-triggered*. Flip-flops are used when it is necessary to sample output values at a particular instant of time (on the positive or negative edge, as the case may be).

Using LogicWorks, construct a D flip-flop with a falling-edge trigger (the device shown is a D Latch, which you can find in the Parts Palette).



Label CK, D, and Q as signals, so their waveforms show up in the timing window. Toggle D high and low several times, and do a screen shot which shows the timing window. Paste it here:

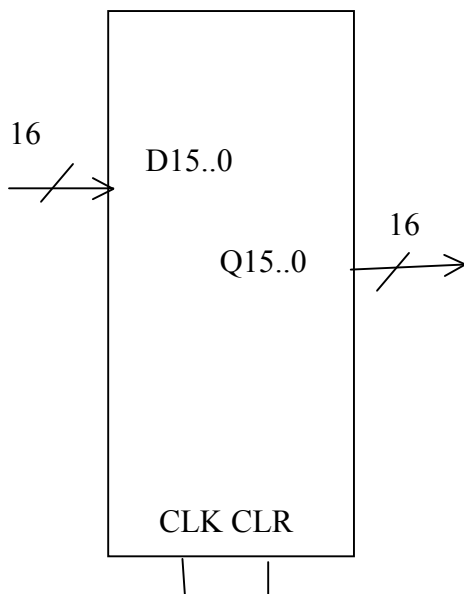


Explain the relationship between CK, D, and Q:

When D changes, the first latch in the flip-flop stores D when the clock goes high. However, the second latch, which produces Q, does not get the value of D until the clock goes low, which is shown in the above diagram (Q always changes value on the falling edge of the clock).

Register

Exercise 5: In lecture, you have learned that a **register** is an n-bit memory (contains n flip-flops, which share a common clock, and are therefore all written to simultaneously). For example, the following device can store 16 bits of information:



You can do the following with this device:

- You can read the current contents of the register at the outputs **Q15..0**
- You can write a new value to all 16 bits at once by setting the value at the inputs **D15..0**, and then activating (setting to high, then back to low) the **CLK** input.

- You can also clear all 16 bits to 0 at any time by activating the **CLR** input (when not using the **CLR** input, keep it set to 0).

Open *the register.cct* file. Use 4 hex keyboards for the 16 inputs. Use 4 hex displays for the outputs. Connect a binary switch to each of the **CLK** and **CLR** inputs.

Write several values to the register by doing the following for each:

- Set the value on the **D015..0** inputs.
- Activate the clock (set the **CLK** to 1 then back to 0)

D15..0

2323

A5A5

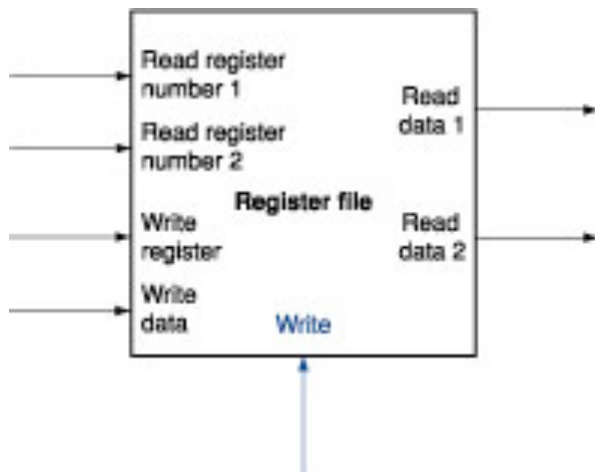
1234

Clear the outputs by setting the CLR line to high and back to low.

Register File

Exercise 6: In lecture, you have learned that a **register file** is a set of registers that can be read from and written to. The register number specifies which register from the set is accessed at any particular time.

Here is a black box diagram for a register file with 2 read ports and 1 write port. This means that you can read from 2 registers and write to a third register at the same time.



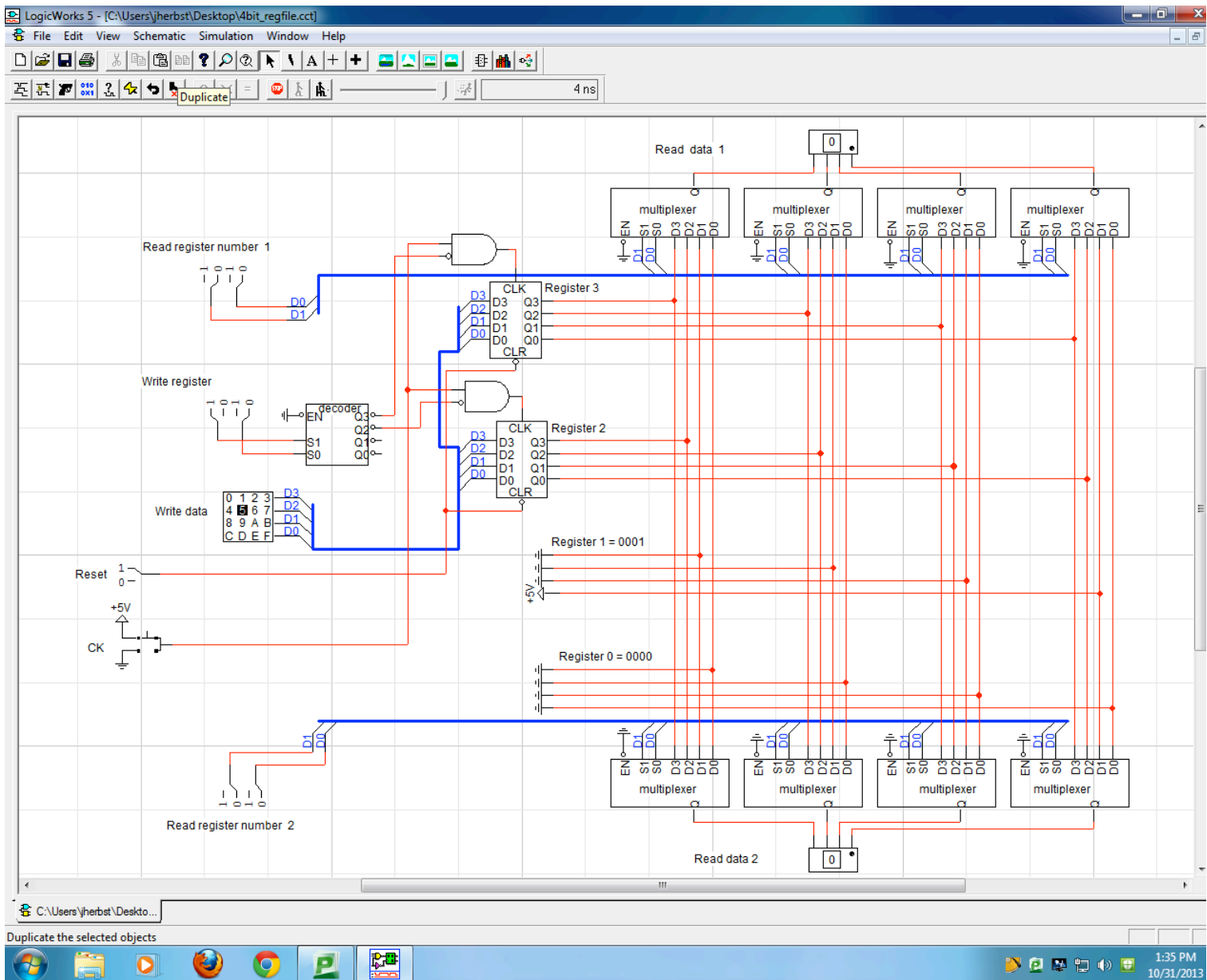
You can read the contents of 2 registers at any given time by specifying the number of the first register at the *Read register number 1* input, and the number for the second register on *Read register number 2*.

The contents of the two registers are then available at the *Read data 1/ Read data 2* outputs.

To write a new value to a specified register, set the *Write register* input to the number of the register, set the *Write data* inputs to the value to be written to the register, and then activate the *Write* line.

Below is LogicWorks circuit which implements a register file for a set of 4 4-bit registers, Register 0 always produces a value of 0, and register 1 always produces a value of 1 (those values are hardwired in this circuit).

This register file is small enough that you can examine and understand the underlying circuitry.



Open the *4bit_regfile.cct* file. To operate the circuit, follow these steps:

Clear all the registers to 0 (except register 1, which will always have a value of 1)

- Set **reset** to 0 and back to 1.

NOTE: It is important to keep the reset = 1 when you are done, since it is an active low signal. Leaving reset = 1 will force all the registers to stay 0.

Read the value of register 0 from the Read data 1 display

- Change **Read register number 1** to 0
(You should see **0** displayed on the **Read data 1** display)

Read the value of register 1 from the Read data 2 display

- Change **Read register number 2** to 1
(You should see **1** displayed on the **Read data 2** display)

Write a value of 9 to register 2

- Set **Write Register** to 2, and set **Write Data** to 9
- Click on **CK**

Write a value of B to register 3

- Set **Write Register** to 3, and set **Write Data** to B
- Click on **CK**

Read the value of register 2 from the Read data 1 display

- Set **Read register number 1** to 2
(You should see **9** displayed on the **Read Data 2** display)

Read the value of register 3 from the Read data 2 display

- Set **Read register number 2** to 3
(You should see **B** displayed on the **Read data 2** display)

Answer the following questions, now that you have operated the circuit:

1. What is the purpose of the decoder device in the circuit?

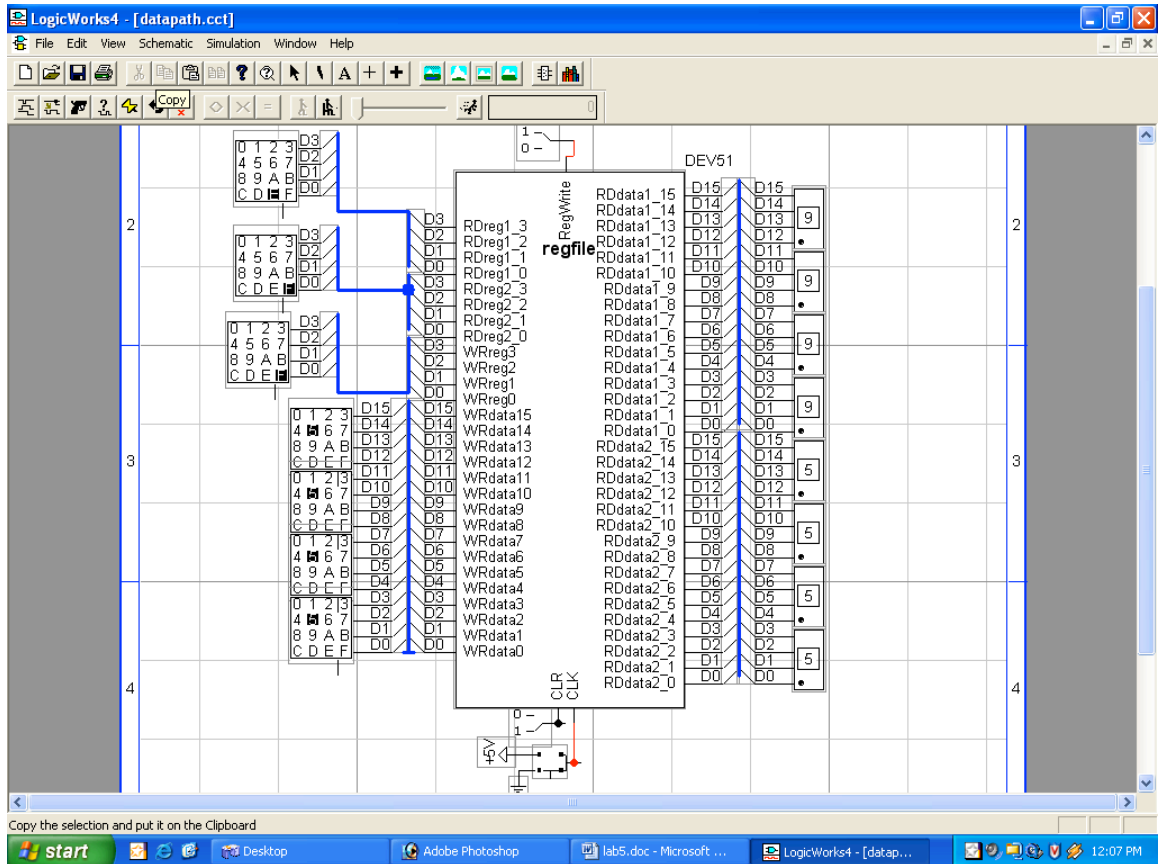
It selects which register gets a new value when a write occurs.

2. Explain how the multiplexers are used to produce the value of the specified registers and the Read data 1 and Read data 2 outputs:

Each multiplexer selects a digit of the result by choosing which register to use for that bit, using the register number as the select lines for all the multiplexers. Since all the multiplexers use the same select lines, the combined result from all the multiplexers is the value of the register specified by the register number.

Demonstrate to the instructor that you understand the operation of the circuit.

Open the *regfile.cct* file. It contains 16 16-bit registers, and will be used for the mini-MIPS datapath. The internal circuits have been abstracted to a black box.



Examine the LogicWorks version of the register file circuit. Notice that the write input is called **RegWrite**, and is located at the top of the device. Also notice that a **CLR** and **CLK** input are located at the bottom of the device, and have the same operation as a single register (**CLR** will clear all the registers. If **RegWrite** is 1, a new value will be written to the write register specified when **CLK** is activated).

The **CLK** switch is a pushbutton, so clicking once activates the input (sets to 1 and back to 0).

Internally, the register file contains 16 registers and two sets of 16 multiplexers, one for each of the read data ports.

There is also a decoder, which uses the write register number to select which of the 16 registers will receive a new value on a write.

Register 0 will always hold a 0, and register 1 will always hold a 1.

Perform the following steps to clear all the registers:

1. Make sure **CLR** = 1, **CLK** = 0, and **RegWrite** = 0. Whenever you are not actively writing a new value to a register, this is the default state of these inputs.
2. Activate the **CLR** (set to 0 and back to 1).
3. Read two registers at a time (start with **RDreg1** = 0, **RDreg2** = 1), and verify that all registers have been cleared. There will be one exception! What is it?

Write the following values to registers 2 – 15, using the following steps:

1. Set **CLR** = 1.
2. Keep **RDreg1** and **RDreg2** set to 0 and 1 (so you will see values of **R0** = 0000 and **R1** = 0001 at the **RDdata1** and **RDdata2** outputs).
3. To write the new values, repeat for each register:
 - o Set **WRreg** to the number of the register to write to.
 - o Set **WRdata** to the data specified.
 - o Set **RegWrite** = 1.
 - o Activate the **CLK**.
4. When done, set **RegWrite** = 0 again.

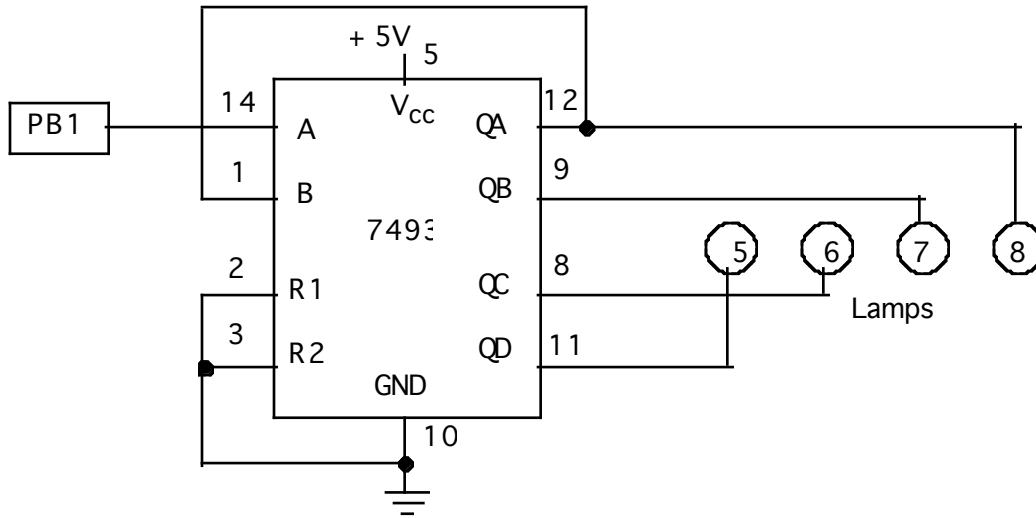
<u>WRreg</u>	<u>WRdata</u>
2	1E1E
3	2D2D
4	3C3C
5	4B4B
6	5A5A
7	6969
8	7878
9	0F0F
A	FFFF
B	0123
C	4567
D	89AB
E	CDEF
F	FFCC

After you have written all the values specified, check that the registers contain the expected values by reading two registers at a time (again, start with **RDreg1** = 0, **RDreg2** = 1).

Demonstrate to the instructor.

Binary Counter

Exercise 7. On your Protoboard, connect IC 7493 by wiring the circuit as shown below.

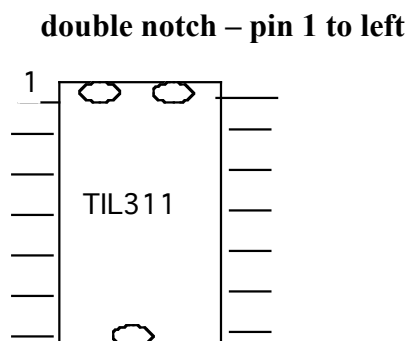
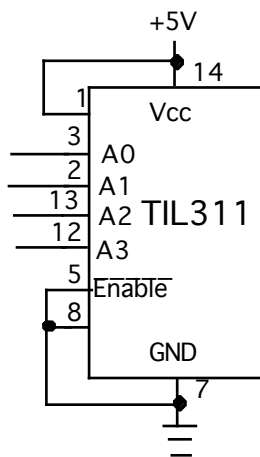


Turn the power on and observe the four indicator lamps/LEDs. Press the PB1 button repeatedly, until all the lamps are green.

- Record the lamp outputs in the truth table below (use a '1' to indicate lamp lit, and a '0' for not lit). Repeat each time you clock the circuit, until the outputs are all dim again.

QD	QC	QB	QA
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

3. Do you recognize the pattern of the lamp outputs? How would you characterize this pattern? Describe what you think this device does: **The binary numbers from 0 to 15. This device is a binary counter.**
4. Turn the power off, but **do not disconnect your circuit!** Insert the TIL311 (shown below) into your protoboard (**Note:** the actual device has a **double notch** on one end, and **pin 1 is to the left of the double notch**).

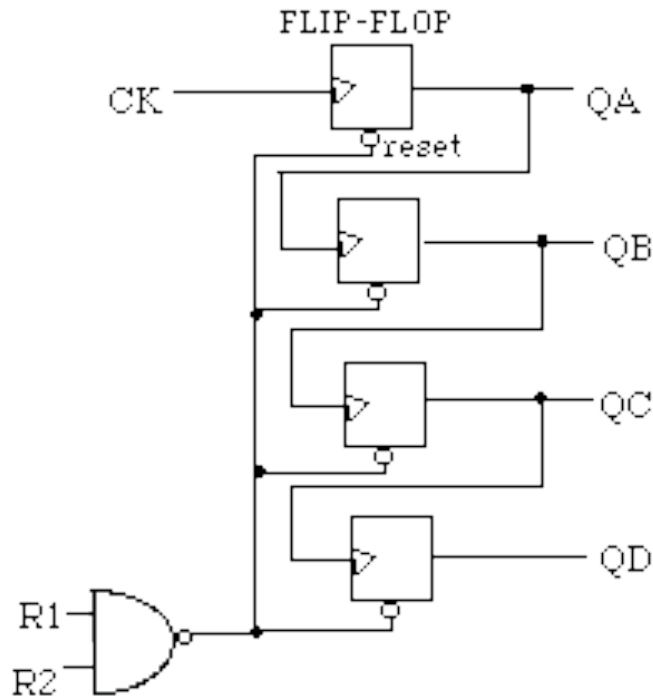


Connect the inputs A0 – A3 of the TIL311 to the outputs QA – QD of the 7493.

5. Power up your circuit. Observe the TIL311 as you repeatedly push the PB1 button. Record the symbols displayed for each of the 16 counts: **0 1 2 3 4 5 6 7 8 9 A B C D E F**
6. What do these symbols represent? What is the purpose of the TIL311, and how is that related to what the 7493 does? **Hexadecimal values 0 to 15. The TIL311 takes a 4-bit binary value as an input, and displays it on a 7-segment LED as its corresponding hexadecimal value.**

Timing Diagrams

Exercise 8: Internally, the binary counter consists of 4 flip-flops connected together in the following configuration:



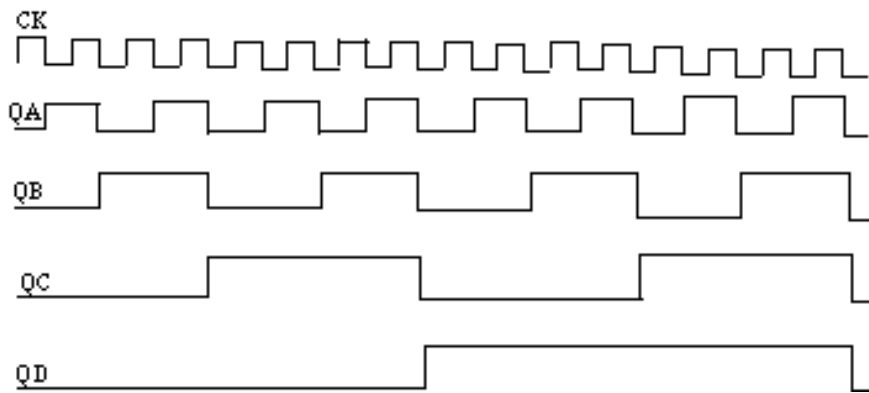
You can graphically observe the outputs of the binary counter in LogicWorks by using the Timing Window.

1. Using LogicWorks, implement the binary counter circuit (use a **clock** as an input to CLKA (you can find it in the **Simulation IO** library in the **Parts Palette**):
2. Use the **Text Tool** in the **Drawing Toolbar** to label the CLKA input as **CK**.

Note: to label a signal, you **must** position the Text Tool to touch the wire representing the signal (not the clock input device, itself). You will know you have done it correctly if the text labeling color is **pink**, and you see the signal listed in the **Timing Window** display at the bottom on the screen.

3. In the same way, label each of the outputs of the binary counter (in the following order) as **QD**, **QC**, **QB**, and **QA**. Once the circuit is complete, make sure to click the **Clear Unknowns** button. Slow down the clock rate with the **Set Simulator Speed** slider in the **Simulation Toolbar** (drag it towards the left).

4. In the **Timing Window**, you should observe 5 waveforms as shown below (if you can't see all five, make the **Timing Window** larger).



5. Explain how the signals are related to one another in terms of the flip-flops that are used to produce the signals. Each signal is a clock to the flip-flop whose output is the next most significant digit in the binary number. Since the flip-flops can only change state on the falling edge of their clock input, each output changes at half the frequency of its clock input.
6. Explain why the signals represent the binary numbers. In the decimal system, the least significant digit changes for each consecutive number. The next least significant digit (the 10s place) changes every 10 values. The hundreds place changes every 100 values, etc. With binary number, it works exactly the same way, except the n th digit changes every power of 2^n . So, the frequency of a particular digit changing value is twice that of the next most significant digit..