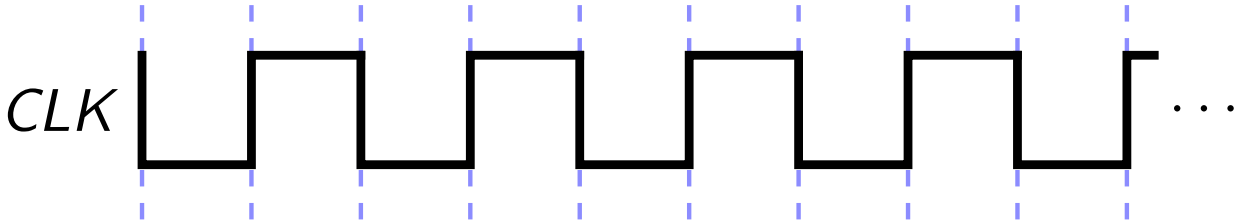
From flip-flops to registers
COMSM1302 Overview of Computer Architecture

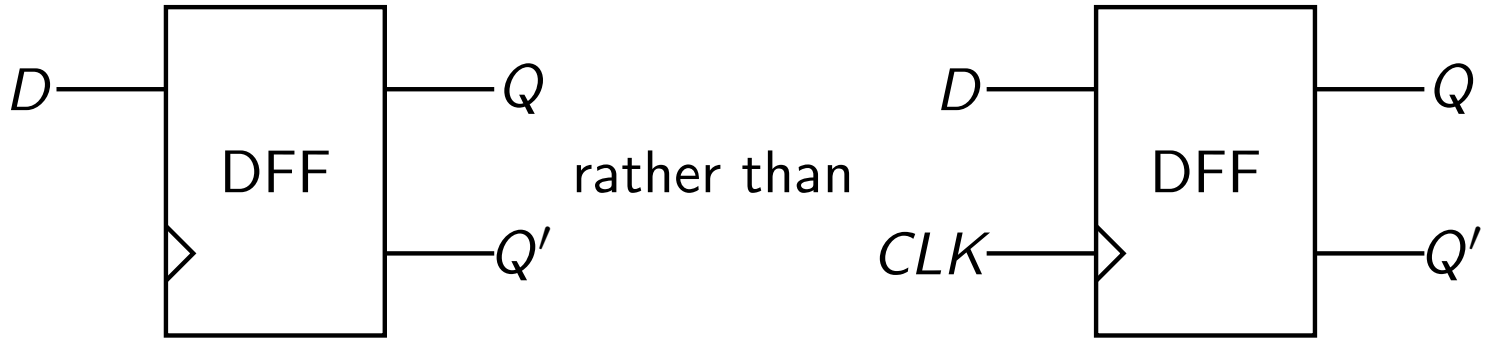
John Lapinskas, University of Bristol

Clocked circuits

Recall from last time our big idea: we drive all our timings with a *single* clock signal.

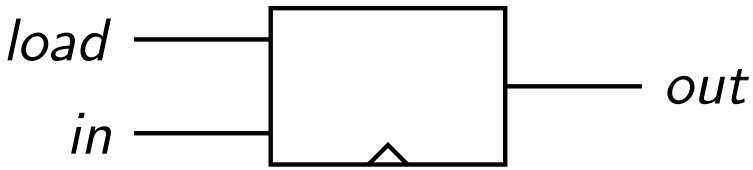


If an edge-triggered circuit is driven by this clock, we denote it by a triangle. To save space, since all circuits use the same clock, we often don't draw in the wire. So e.g. a D flip-flop is often drawn as:



1-bit registers

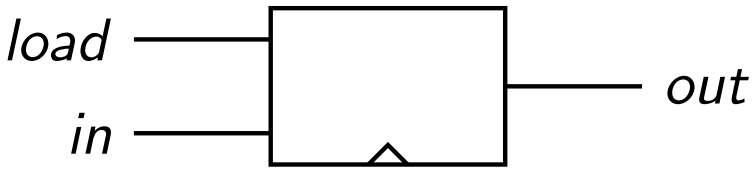
A (1-bit) **register** is a simple variant on a D flip-flop:



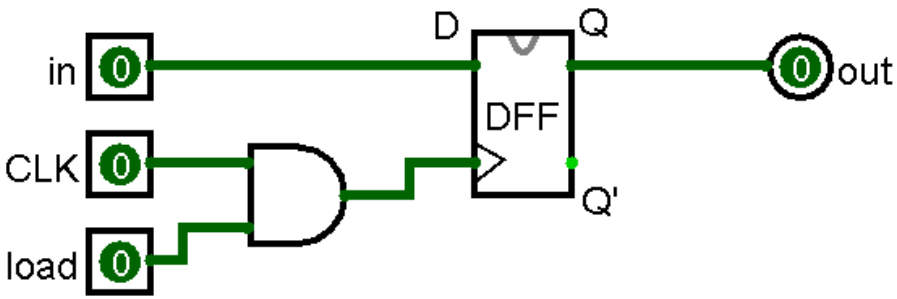
On a rising clock edge, if *load* is high, *out* takes on the value of *in*. If *load* is low, *out* remains unchanged. How should we design this?

1-bit registers

A (1-bit) **register** is a simple variant on a D flip-flop:

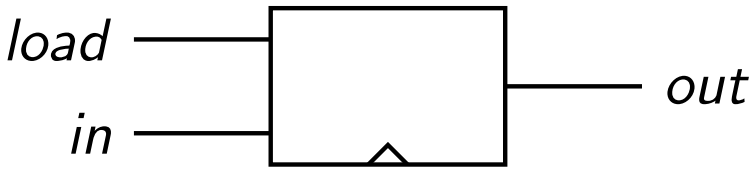


On a rising clock edge, if *load* is high, *out* takes on the value of *in*. If *load* is low, *out* remains unchanged. How should we design this?

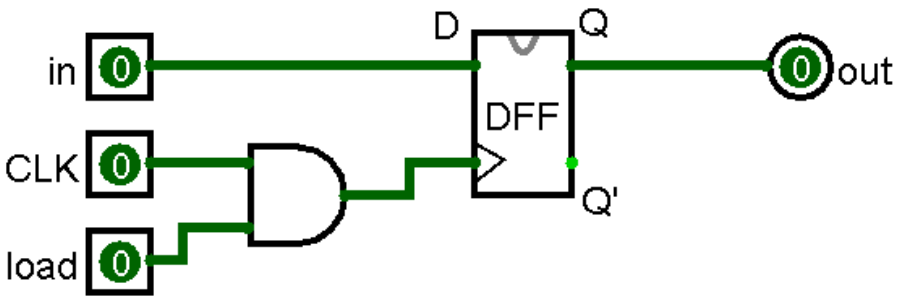


1-bit registers

A (1-bit) **register** is a simple variant on a D flip-flop:



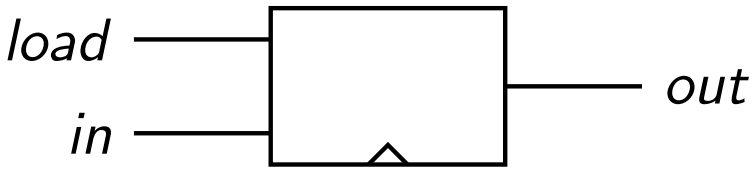
On a rising clock edge, if *load* is high, *out* takes on the value of *in*. If *load* is low, *out* remains unchanged. How should we design this?



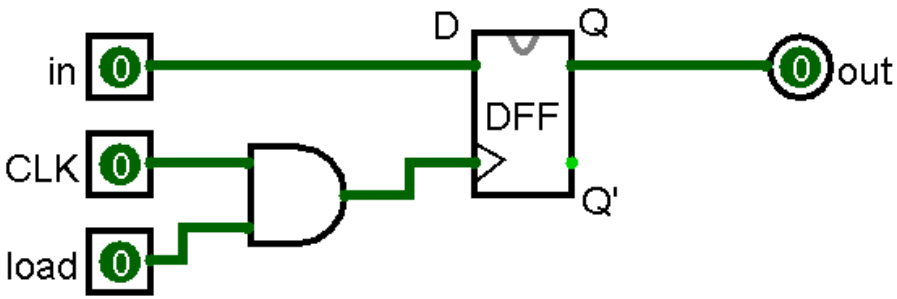
This design **fails**. Can you see why?

1-bit registers

A (1-bit) **register** is a simple variant on a D flip-flop:



On a rising clock edge, if *load* is high, *out* takes on the value of *in*. If *load* is low, *out* remains unchanged. How should we design this?



This design **fails**. Can you see why? Because here, *out* takes on the value of *in* on a rising edge of $load \wedge CLK$, not of CLK. So if e.g. *load* oscillates while CLK is high, *out* is unstable.

In general when working with flip-flops, it's a bad idea to mess with the clock signal — it's easy to introduce bugs like this. Instead...

1-bit registers: the correct way!

We use the outputs from one clock cycle to set the inputs for the next, using combinatorial logic. We want:

in	*load*	out_{old}	out_{new}
X	0	0	0
X	0	1	1
0	1	X	0
1	1	X	1

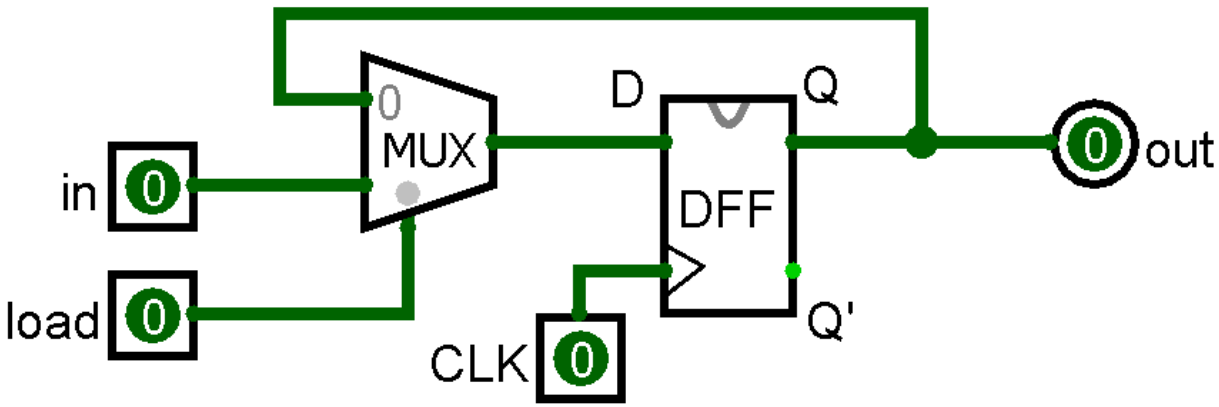
Here as last video, X means "don't care".

1-bit registers: the correct way!

We use the outputs from one clock cycle to set the inputs for the next, using combinatorial logic. We want:

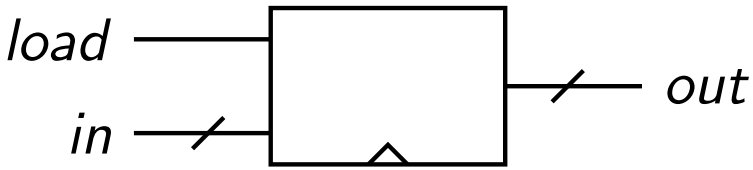
in	*load*	out_{old}	out_{new}
X	0	0	0
X	0	1	1
0	1	X	0
1	1	X	1

Here as last video, X means "don't care". We could expand this out into a full truth table and use a K-map... or just notice that this is a multiplexer!



Multi-bit registers

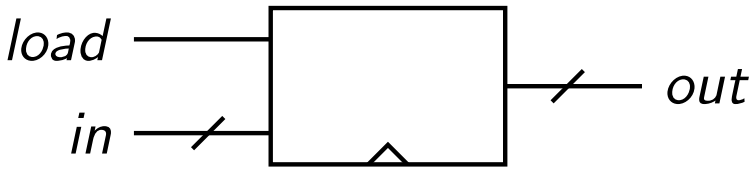
The Hack CPU uses not 1-bit registers but 16-bit registers, which we think of as holding a single 16-bit binary value each:



Rather than drawing 16 separate wires, we draw one wire with a slash through it, which we again think of as holding a binary value. We call this sort of "binary wire" a **bus**. (Logisim uses slightly different notation.)

Multi-bit registers

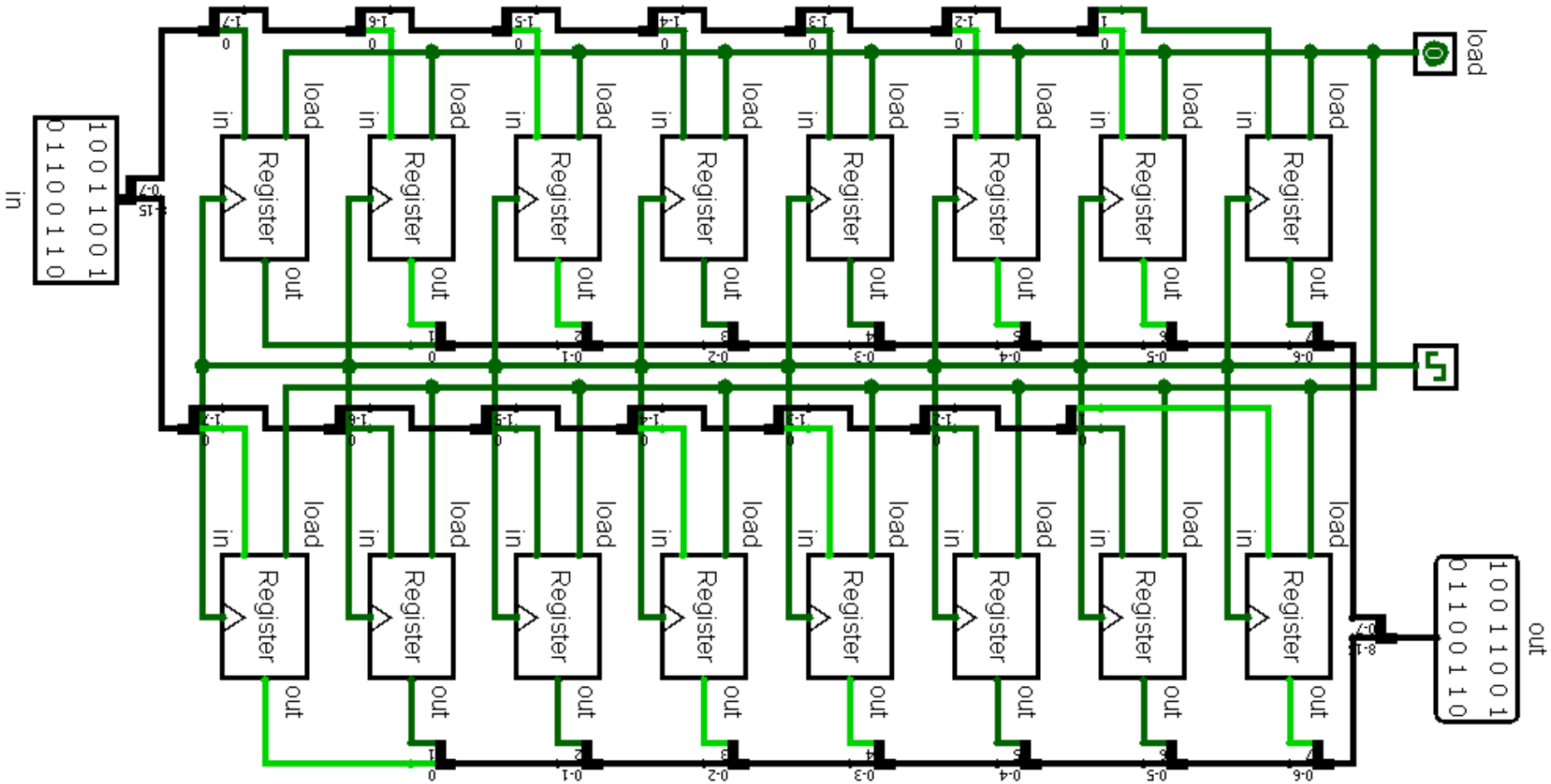
The Hack CPU uses not 1-bit registers but 16-bit registers, which we think of as holding a single 16-bit binary value each:



Rather than drawing 16 separate wires, we draw one wire with a slash through it, which we again think of as holding a binary value. We call this sort of "binary wire" a **bus**. (Logisim uses slightly different notation.)

It's simple to implement a 16-bit register, but it's a good excuse to show how buses and clocks work in Logisim.

Demonstration



[See video for a demonstration of assorted features in Logisim.
The circuit is available for download in readable form from the unit page.]