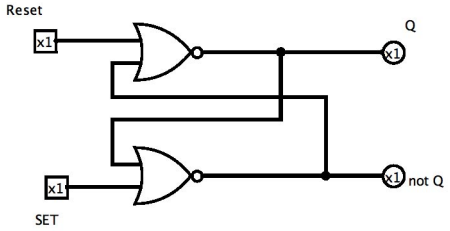
CS 237 Meeting 14 — 10/12/12

Announcements

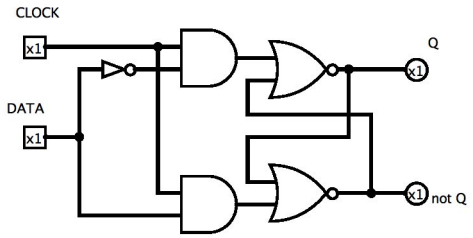
1. Midterm: Oct 26th. In class open book/notes.
2. When you implement heap sort, take advantage of the fact that the C code only invokes swap in one place in heap

What a Memory

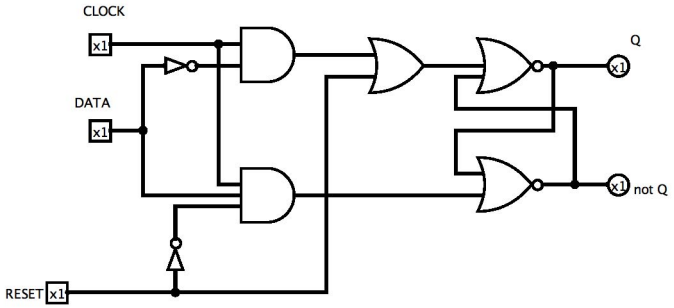
1. The basis for all memory circuits are devices that have more than one stable state and a means for an external device to cause a switch between these two stable states.
2. The canonical example of such a device is the SR latch, obtained by connecting two NOR gates in such a way that each of the gates takes the output from the other gate as an input.



3. The weakness of the SR latch is that it does not match our model of what memory should do. The SR latch remembers what input line was set to 1 most recently. When we think of memory, we think of a device that can remember a number (or at least a binary digit) that was available at some moment in the past.
4. The D latch accomplishes this by taking two inputs:
   - The D input, as described above is a bit to be remembered, and
   - The CLK input is set to 1 when we want to record each change in D and set to 0 when we want to remember a particular value of D and ignore future changes in D's value (until CLK becomes 1 again).



5. There are several minor features that can be (and are) incorporated into pre-packaged latches (and flip flops which we will discuss next) that are handy to have when working with them.
   - One feature is to include a "set" (or "reset") input that sets the value in the latch/flip-flop to 1 (or 0) independent of the values on the D and CLK inputs. This is easily implemented by adding a few gates to the circuit:



Non-political Flip Flops

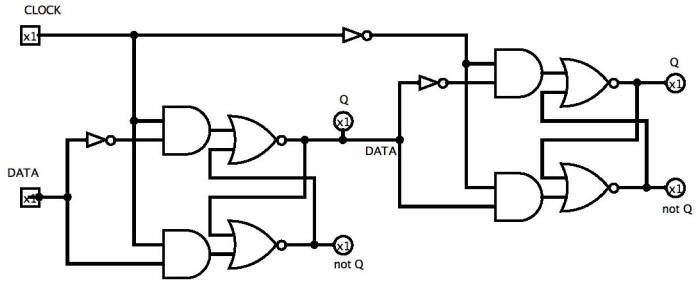
1. The easiest form of clock to include in a computer is one that turns on and off for equal amounts of time at regular intervals.
2. The "equal amounts of time" part is not optimal for latches.
   - Imagine how to build a very simple circuit that acts as a counter.
     - To hold the current value of the counter, we would use a big enough register to remember all the bits of the counter's binary representation.

- To be able to increment the counter, we could flip the low order bit on each clock cycle and include a half adder to determine the next value for each higher order bit.
- The result of this addition would then be connected to the input (D) bits of the register.
- All of the register's clock signals would be connected to a single clock.
- The intent would be that while the clock was set to 0, the adder would be computing the updated total. The, when the clock struck 1, the register would replace its former contents with the updated total.
- Note, this would be a great device to hook to the program counter register (possible adding 4 instead of 1).
- Unfortunately, if the clock was off for as long as it was on, the adder would update many of its output bits (possibly all) between the point where the clock goes to 1 and returns to 0. Since the clock would still be one, the register would update its contents.

3. The solution to this problem is the flip-flop, a device that is like a latch except it only updates its output to reflect its input at one of the edges of a clock pulse. Flip flops come in "leading edge" and "training edge" varieties.

4. One simple way to build a flip flop is to take two latches, make one's output be the next latches input, and hook one to the clock and the other to the clock's negation.

   - When the CLK signal is 0, the first latch remains unchanged.
   - When the CLK signal becomes 1, the first latch becomes transparent. Its output changes as soon as its input changes, but the second latch remains unchanged.
   - When the CLK signal falls from 1 to 0, the first latch stops changing and the second latch passes the final value of the first latch on as its new value.



This is called a Master Slave D Flip Flop.

5. By moving the not gate between the main clock signal and the signals to the D latches used as subcomponents, we can choose to either have the new value set at the point the clock rises or when it falls.

A Simple Sequential Circuit

1. To really appreciate how latches and flip flops can be used (and to appreciate why the CLK input is called CLK), we need to build a circuit that uses these components to implement a simple algorithm involving a loop.

2. Recall that way back when we were learning C and the basics of MIPS assembly, we considered several ways to implement algorithms to compute an integer approximation to the square root of an integer. The dumbest version looked like:

```
int dumbSqrt( int v ) {

  int ans=1;

  while ( ans*ans <= v ) {
    ans = ans + 1;
  }
  ans = ans - 1;
  return ans;

}
```

3. This algorithm is really dumb. Not only is it linear when it could be easily replaced with a log n algorithm based on a binary search. In addition, with each step it requires a multiplication which is a relatively time-consuming operation (compared to an addition).

4. There is a simple trick we can use to eliminate the need for multiplications in this algorithm. We know that for any x, $(x+1)^2 = x^2+2x+1$. Therefore, if we keep the values of both x and x^2 in variables we can write a loop that updates them together giving access to the value of x^2 without any "real" multiplications (i.e., the $2x$ can be done by shifting or with addition).

5. With this trick we can rewrite the C code as

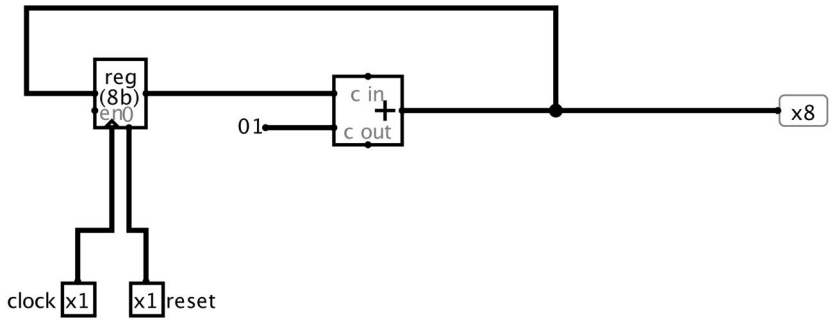
```
int lessDumbSqrt( int v ) {

  int ans=1;
  int ansSquared = 1;

  while ( ansSquared <= v ) {
    ansSquared = ansSquared + 2*ans + 1;
    ans = ans + 1;
  }
  ans = ans - 1;
  return ans;

}
```

6. We have seen that we can build a circuit to repeatedly increment the value in a register by adding one to the output of the register and feeding the result back in as the new input value for the register:



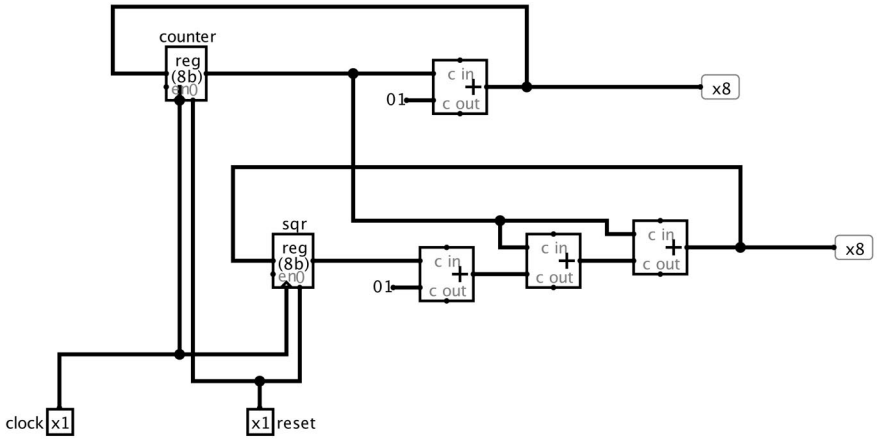
This circuit effectively implements the loop:

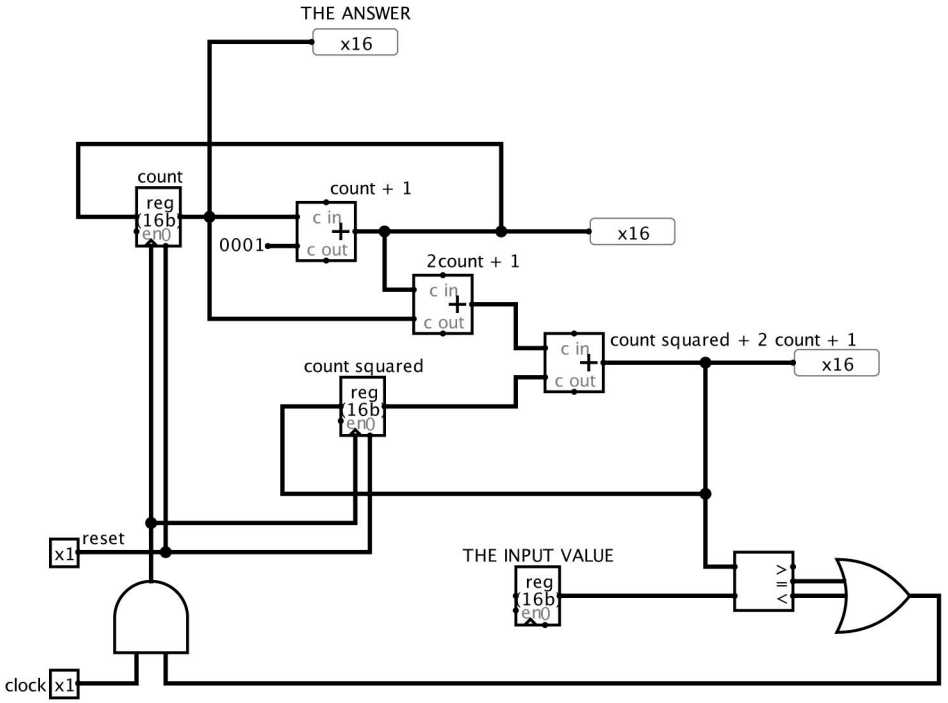
```
int ans=0;
while ( true ) {
  ans = ans + 1;
}
```

7. WIth a little work we can add components to this circuit so that it also computes the square of the simple counter, thereby executing the loop:

```
int ans=0;
int ansSq = 0;
while ( true ) {
  ansSq = ansSq + 2 * ans + 1;
  ans = ans + 1;
}
```

The circuit requires an additional register to hold the square. It then simply uses adders to combine the outputs of the registers in the appropriate way.





8. Finally, we can add logic to disable the clock input to the registers once the square exceeds a target value, thereby implementing the loop condition.