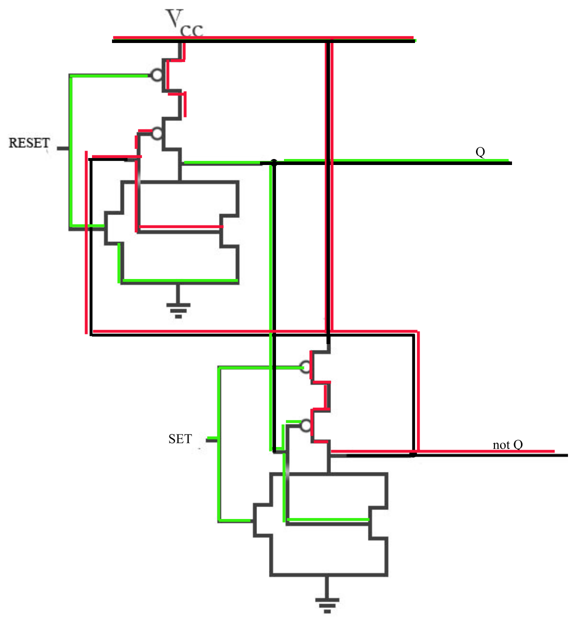
CS 237 Meeting 15 — 10/15/12

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
3. Lab 5 available online by tonight

Circular Circuits

1. Last time, I promised I would address one issue that crossed my mind when I was first introduced to the SR latch — the sneaking suspicion that it relied on a perpetual motion machine in which electrons chased themselves around the loop formed by the connected NOR gates.
2. If we examine the SR latch at the transistor level, we can see that no such loop exists:



3. In the diagram, I have highlighted wires at V_{CC} in red and wires at ground in green. It show an SR latch in one of its stable states. Both the S and R inputs are logical 0s and therefore at ground potential (green). The output (Q) is at ground representing 0. The $\bar{Q}$ output is a 1 (V_{CC}.
4. Recall that the transistors with the little circles (pMOS) are open switches when their gates are at V_{CC} and are closed when their gate inputs are at ground. The transistors without the little circles (nMOS) are open switches when their gate inputs are at ground and closed when the input is V_{CC}.
5. For electrons to be flowing at all in this circuit (let alone for them to be flowing in a circle), there would have to be a path of wires and closed switches from V_{CC} to ground. Instead, as the diagram shows, all such paths contain an open switch.

A Simple Sequential Circuit

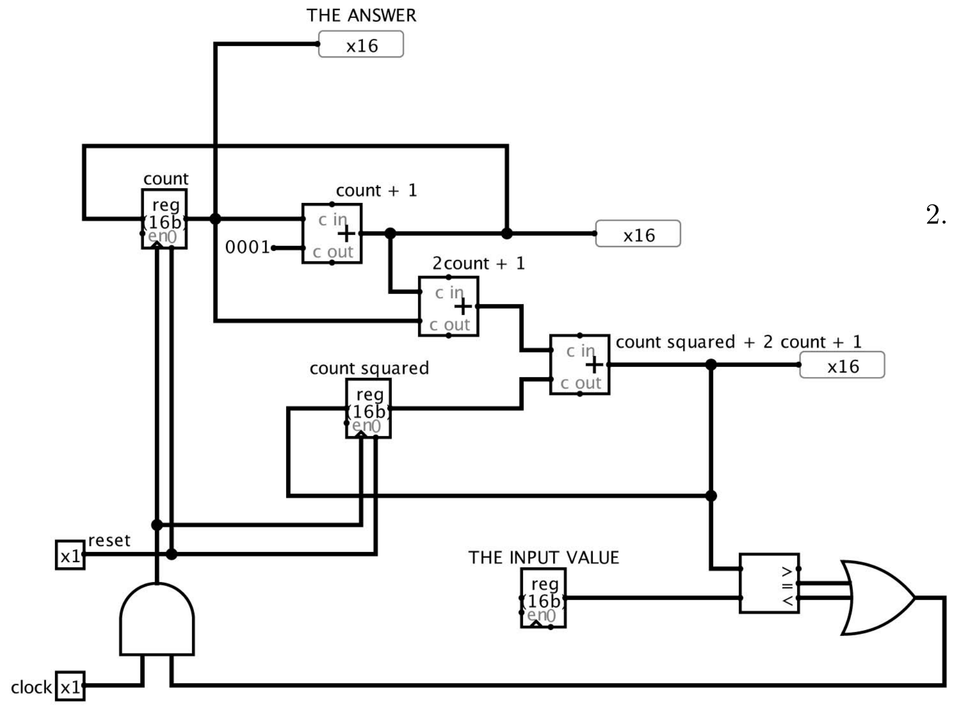
1. Last time, to illustrate this we sketched out a circuit that could implement a rather slow algorithm for computing integer approximations to square roots:

```
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;

}
```

I want to look at another example of such a circuit today, but first I want to give you a chance to ask questions about the square root example.

2. Our circuit consisted of a *data path'* that could repeatedly increment one register (think variable) and repeatedly increment the other in such a way that it remains equal to the square of the first.

3. We combined this data path with some *control circuitry* that enabled us to initialize the circuit with a new input value and ensured that the incrementing processes would stop once an answer was found.



Another Circuit Implementing an Algorithm

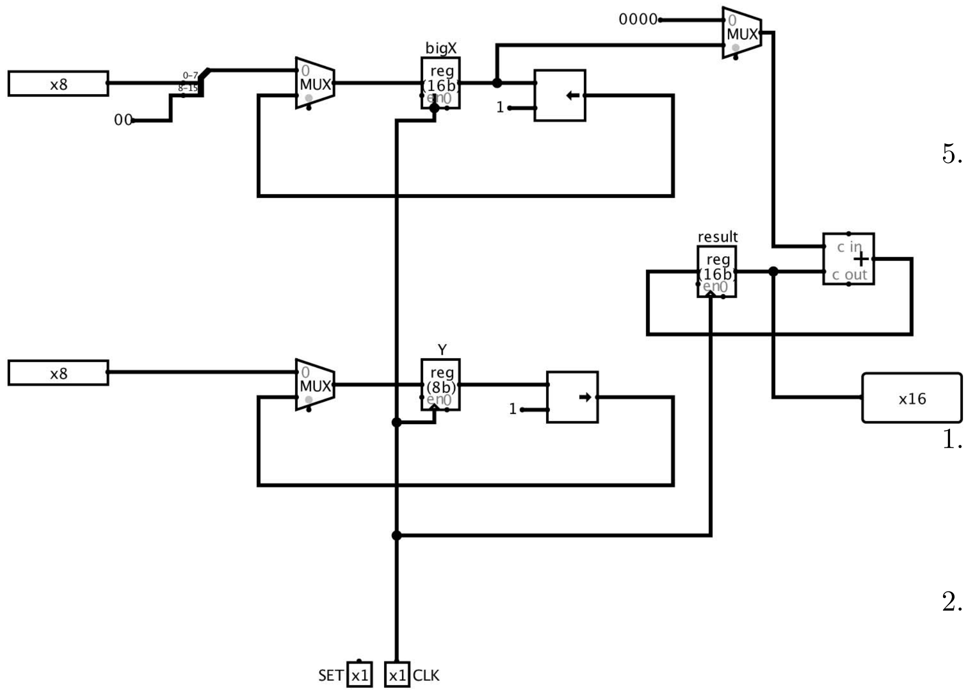
1. Consider the following C function:

```
unsigned short mult( unsigned char x, unsigned char y ) {
  unsigned short result = 0;
  unsigned short shiftedX = x;
  int p;

  for ( p = 0; p < 8; p++ ) {

    if ( y % 2 == 1 ) {
      result = result + shiftedX;
    }

    shiftedX = shiftedX << 1;
    y = y >> 1;
  }

  return result;
}
```

2. This code implements a simple binary multiplication algorithm

   - In the standard decimal multiplication algorithm, we sum together the results of multiplying the individual digits of one multiplicand and the power of 10 corresponding to its digit position by the other multiplicand.
   - Since in binary, the only digit values are 0 and 1, this is equivalent to either adding or not adding shifted copies of one multiplicand depending on whether specific bits in the other multiplicand are 0 or 1.
   - Each time around the loop, we shift one multiplicand left one bit and the other right one bit.
   - We add a copy of the left-shifted multiplicand to the accumulator, result, if the low order bit of the right-shifted multiplicand is 1.

3. The diagram below shows circuitry that might be used as the data path for a device intended to implement this multiplication algorithm.



- Two registers are provided to hold the multiplicands.
- Two shifters are used to shift the outputs of these registers in the appropriate directions.
- Each register's input comes through a multiplexer provided to made it possible to either load the register with a new input (When the reset input is asserted), or to load it with an appropriately shifted version of its preceding value.
- A third register is used to hold the accumulated sum. Its output is either added to 0 or the left-shifted multiplicand's current value and then fed back to itself as a possible new value.

4. Our goal is to add control circuitry to make the reset and clock inputs control the circuit appropriately.
   - When the reset signal is 1 and the clock cycles, new inputs should be loaded and the total should be set to 0.
   - When the clock cycles while reset is 0, one cycle of the multiplication loop should be executed.

5. Any suggestion?

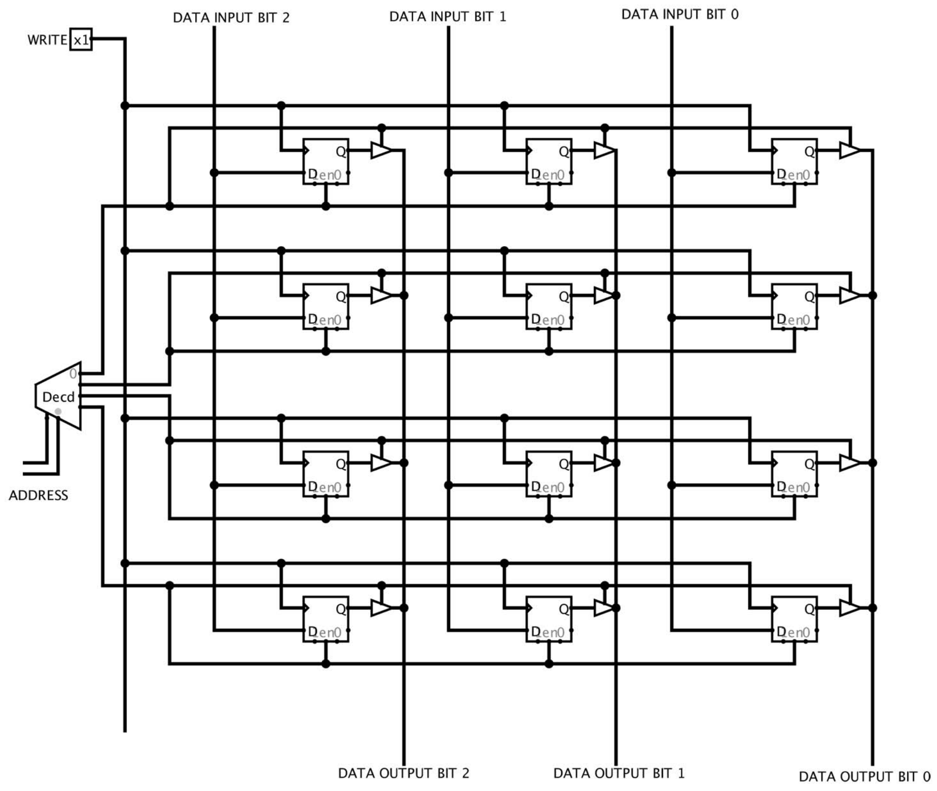
Building Multiword Memories

1. We have seen that several flip flops can be combined to form a register. Even more can be combined to form an addressable memory.

2. The address bits in a memory are processed by a decoder that sends a signal known as the *wordline* to each group of flip flops that together form an addressable unit (word) within the memory.

   The wordline tells the group of flip flops a) whether to update when the clock/write signal is asserted, and b) whether to output their current values onto outgoing *bitline*s.

3. The circuit shown below uses a component we have not talked about, a buffer, controlled by the word line to connect or disconnect each flip flops Q output to its bit line. This is logically equivalent to building a multiplexer to select which flip flops output to use as the output of the memory.



Memory with Fewer Transistors

1. The memory organization suggested above uses flip flops that each require a reasonably large number of transistors.

2. This provides a sufficient, abstract understanding of how memory might be built, but in practice there are alternatives to flip flops that can be used to store bits and that require fewer transistors.

3. The text describes such memories. For lack of time, we will not discuss them in class.