5. TOY Simulator
**Goal.** Write a program to "simulate" the behavior of the TOY machine.

- TOY simulator in Java.
- TOY simulator in TOY!

```java
public class TOY {
    public static void main(String[] args) {
        int pc = 0x10;  // program counter
        int[] R = new int[16];  // registers
        int[] mem = new int[256];  // main memory

        // READ IN .toy FILE
        while (true) {
            // FETCH INSTRUCTION and DECODE
            ...
            // EXECUTE
            ...
        }
    }
}
```

% java TOY add-stdin.toy
A012  standard input
002B  standard output
A03D  standard output
**TOY Simulator: Fetch**

**Fetch.** Extract destination register of \texttt{1cab} by shifting and masking.

```
int inst = mem[pc++];  // fetch and increment
int op  = (inst >> 12) & 15;  // opcode (bits 12-15)
int d   = (inst >> 8)  & 15;  // dest d   (bits 08-11)
int s   = (inst >> 4)  & 15;  // source s  (bits 04-07)
int t   = (inst >> 0)  & 15;  // source t  (bits 00-03)
int addr = (inst >> 0) & 255;  // addr     (bits 00-07)
```
```c
if (op == 0) break; // halt
else if (op == 0x01) R[d] = R[s] + R[t];
else if (op == 0x02) R[d] = R[s] - R[t];
else if (op == 0x03) R[d] = R[s] & R[t];
else if (op == 0x04) R[d] = R[s] ^ R[t];
else if (op == 0x05) R[d] = R[s] << R[t];
else if (op == 0x06) R[d] = R[s] >> R[t];
else if (op == 0x07) R[d] = addr;
else if (op == 0x08) R[d] = mem[addr];
else if (op == 0x09) mem[addr] = R[d];
else if (op == 0x10) R[d] = mem[R[t]]; 
else if (op == 0x11) mem[R[t]] = R[d];
else if (op == 0x12) if (R[d] == 0) pc = addr;
else if (op == 0x13) if (R[d] > 0) pc = addr;
else if (op == 0x14) pc = R[d];
else if (op == 0x15) R[d] = pc; pc = addr;
```
TOY Simulator: Omitted Details

Omitted details.

- Register 0 is always 0.
  - reset $R[0] = 0$ after each fetch-execute step

- Standard input and output.
  - if $\texttt{addr}$ is FF and opcode is load (indirect) then read in data
  - if $\texttt{addr}$ is FF and opcode is store (indirect) then write out data

- TOY registers are 16-bit integers; program counter is 8-bit.
  - Java \texttt{int} is 32-bit; Java \texttt{short} is 16-bit
  - use casts and bit-whacking

Complete implementation. See \texttt{TOY.java} on booksite.
Simulation

Consequences of simulation.

- Test out new machine or microprocessor using simulator.  
  (cheaper and faster than building actual machine)
- Easy to add new functionality to simulator.  
  (trace, single-step, breakpoint debugging)
- Reuse software from old machines.

Ancient programs still running on modern computers.

- Ticketron.
- Lode Runner on Apple IIe.
Q. Why is standard US rail gauge 4 feet, 8.5 inches?
Q. Why is standard US rail gauge 4 feet, 8.5 inches?
A. Same spacing as wheel ruts on old English roads.

Mail wagon, circa 1890
Backwards Compatibility

Q. Why is standard US rail gauge 4 feet, 8.5 inches?
A. Wheel rut spacing same as old Roman war chariots.
Backwards Compatibility

Q. Why is standard US rail gauge 4 feet, 8.5 inches?
A. Roman war chariot wide enough to accommodate "back ends" of two war horses!
Q. Why is Space Shuttle SRB long and narrow?
Simulation and Backwards Compatibility

Napoleon's march on Russia.
- Progress slower than expected.
- Eastern European ruts didn't match Roman gauge.
- Stuck in the field during Russian winter instead of Moscow.
- Lost war.

Simulation tradeoff:
- Simulation essential to reuse old software.
- Maintaining backward compatibility can lead to inelegance and inefficiency.
- Simulation needed to conquer world.