General observations...

Some confusion about how bits represent information and about how hexadecimal works as a shorthand for 4-bit patterns. Judging by questions at office hours, people are not yet fully familiar with basic patterns of bits, like 1 followed by n zeros (which is $2^n$) or n 1's, which is $2^n - 1$. The corresponding decimal values (1, 2, 4, 8, 16, …; 1, 3, 7, 15, …) ought to be familiar up to at least 1024 as well. These patterns and numbers show up all the time, including settings like problem sets and exams, so it’s a good idea to learn them once and for all.

There was considerable confusion as well about how the Toy machine works. Please note: spaces are significant in Toy programs in my hacky implementation: labels must be in the first column (no preceding spaces), and instructions like add must not be in the first column. Also, appealing as they might appear, you can’t write things like “ifzero print” or “ifpos goto stop”. (Higher-level languages allow such statements, which is one reason why they are much easier to use.)

A number of people created special-purpose instructions for tasks like squaring or cubing. That wasn’t what was asked for, but I gave limited partial credit if the code appeared to work.

Problem 1: [15 pts: 3 + 12]
(a) Your three colors, with names and hex values:

Some very creatively-named examples were submitted.

(b) Write down the hex representations with 7-10-7 allocation of 24 bits.

<table>
<thead>
<tr>
<th>Color</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>FE0000</td>
</tr>
<tr>
<td>green</td>
<td>01FF80</td>
</tr>
<tr>
<td>blue</td>
<td>00007F</td>
</tr>
<tr>
<td>yellow</td>
<td>FFFF80</td>
</tr>
<tr>
<td>cyan</td>
<td>01FFFF</td>
</tr>
<tr>
<td>magenta</td>
<td>FE007F</td>
</tr>
</tbody>
</table>

This seems easiest if you lay out 24 bits in a row. For red, the first 7 are 1’s, the remaining 17 are 0’s. So the pattern is 11111110 00000000 00000000. Notice where I put spaces. Those mark the byte boundaries. Now write each 8 bits as two hex digits: 1111 1110 is FE. The other colors are exactly the same idea. Here’s green: 00000001 11111111 10000000, which is 01FF80.

**Problem 2:** [8 pts: 2 each]

(a) How many bits for the population of the USA? How many bytes?

330 million is $330 \times 10^6$ so $2^9 \times 2^{20}$ so 29 bits, 4 bytes

(b) How many bits for the population of California? How many bytes?

40 million: 26 bits, 4 bytes

(c) How many bytes for the US national debt of $29$ trillion?

$29 \times 10^{12}$ is $2^5 \times 2^{40}$ so 45 bits so 6 bytes should do it.

(d) How many bytes for the number of monthly FB users, 2.9 billion?

$3 \times 10^9$ is $2^2 \times 2^{30}$ so 32 bits or 4 bytes.

**Problem 3:** [10 pts: 2 + 8]
(a) What section of the von Neumann paper discusses multiplication hardware tradeoffs?

First appears in section 1.5: “the operation of multiplication could be eliminated from the device as an elementary process if one were willing to view it as a properly ordered series of additions.” I accepted section 5, which has much more detail.

(b) What extra code did you add or modify in the simulator? Paste it here, preferably in a monospace font like Courier (so things line up cleanly):

```java
} else if (opcode[pc] == "mul") {
    accumulator = parseFloat(accumulator) *
    parseFloat(litoradr(adr[pc]));
```

This would most naturally be right after the code for “sub”

There are also two places where you have to add “mul” to the list of opcodes, like this error test:

```java
    if (adr[i] == "" &&
        /store|load|add|sub|mul|goto|ifpos|ifzero/.test(opcode[i]))
```

and

```java
    return
    /get|print|store|load|add|sub|goto|ifpos|ifzero|stop/.test(op);
```

**Problem 4: [10 pts: 4 + 6]**

(a) Paste your program for printing the square and cube here:

```java
get
store n
mul n
print
mul n
print
stop
```

(b) Paste your program for counting down here:

```java
get
pr
print
sub 2
ifpos pr
stop
```
This part proved harder than it was meant to be. But it should have been easy to get started: you have to read a number and print it. Then there are only two choices: subtract 2 and test, or test and subtract 2. Either way, if you're not done yet, you have to print and subtract again.

It's an easy program to test. In spite of that, a fair number of answers failed to print the input number, or printed -1 for odd inputs, or went into infinite loops.