Graded out of 43 points.

General observations...

A lot of people were confused about how hexadecimal representations of colors work. Please read section 1b below carefully for more details.

Notice that the number of bytes need to be rounded up. For example, 26/8 is 3.25, so 26 bits requires 4 bytes of storage, not 3.25 bytes, and not 3 bytes.

Make sure that you read the problems carefully. For example, problem 4a asks for one program that prints the square and cube of the input. Some people wrote two separate programs. In addition, 4a requires the program to not print the input, but 4b requires it to print the input along with everything else.

Although you can find brackets and if/else statements in many higher level languages, the toy simulator does not recognize them, so please use the relevant syntax.

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.

If this—7F00000, 003FF00, 000007F, 7F3FF00, 003FF7F, 7F0007F—is your answer for 1b, I did not deduct any points, but this does not mean your answer is correct. Please read the explanation above.

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

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

330 million is 330 * 10^6 so 2^9 * 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 $31 trillion?

31 * 10^12 is 2^5 * 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 * 10^9 is 2^2 * 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):

4 points:

```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 (2 points each):

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

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

Problem 4: [10 pts: 5 + 5]

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

```
get
store n
mul n
print
mul n
print
stop
n
```
Many of you did not set aside a memory location to store the value, and the toy simulator thinks it's a non-existent instruction. Make sure your program squares and then cubes the input. Do not read the input multiple times.

(b) Paste your program for counting down here:

```plaintext
get
pr print
sub 1
ifpos pr
stop
```

Common errors include not printing the input, infinite loops, improper use of labels (label not found/names not matching), and not setting aside a memory location if you used one.