1 Why Binary? A better idea? (3 points)

The binary logic widely used nowadays permits each wire or line or storage location in memory to represent only one of two possibilities, as you know. So, two lines permit 4 possible combinations (the two “binary digits” or “bits” can encode 4 distinct combinations). 3 bits can encode 8 combinations; and so on.

1.1 How many combinations, for a given number of binary lines? (1 point)

A “binary digit” or “bit” is provided by each wire or line carrying a conventional digital signal. Early microprocessors used 16 lines to address memory, providing $2^{16}$ combinations and thus $2^{16}$ distinct addresses or locations in memory space. The processors on early IBM PC’s worked this way—limiting address space, directly addressable, to $2^{10} \times 2^{10}$; and so on. (IBM worked around this by letting the processor switch among such small “segments” of memory. The version of the 8051 processor that you’ll soon be using also uses just 16 lines to address memory, and also offers this somewhat-clumsy ‘paging’ option.)

Later processors, including the 68000 family, used more lines to address memory. The 68000 addresses were defined using 32 lines, allowing it to address $2^{32}$ locations. The old IBM PC bus (EISA) used 20 lines, permitting it to address $2^{20}$ locations.
1.2 How much improvement in density of info, with more levels? (1 point)
If one were to adopt a 3-level scheme (“ternary” rather than binary), then each line could represent one of 3 values. How many combinations could be encoded by three ternary lines or digits?

How many could be encoded by three quaternary lines or digits (a 4-level scheme)?

1.3 ...then why is anything other than binary rare? (1 point)
Then why not use ternary, quaternary or more?

2 Numbers (2 points)
Write the decimal equivalents of the following binary numbers (even if you’ve learned another algorithm for evaluating 2’s comp, try using the fact that the MSB carries its usual weight—but is negative). (See Class Notes D1/13.)

<table>
<thead>
<tr>
<th>Number</th>
<th>Decimal</th>
<th>--if Unsigned binary</th>
<th>--if signed (2’s comp) binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10111</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3 Buttons and Switches (3 points)

Hook up a button so that when pressed it outputs a logical 1 (5V) and a logical 0 (0V) otherwise. Never let infinite current flow.

Hook up a button so that when pressed it outputs a logical 0 (0V) and a logical 1 (5V) otherwise.

Hook up a SPDT switch (single pole, double throw – meaning three terminal) to output either a logical 1 or 0, depending on the position of the switch.

4 Implement this and that function (2 points)

Use AND, OR and NOT gates to implement the following functions (F, G, H, and I) based on the state of inputs A and B.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
5 H-Bridge (8 points)

This problem is to give a deeper understanding of how CMOS gates work, but in a different context.

You want to control a motor, making it spin with a logic signal called P (for Power) and choosing the direction with another signal called W (for clockWise). The motor needs more current than any of our logic chips can provide, so you will use two NMOS and two PMOS power transistors. (The NMOS part IRL510 in your drawer can switch up to 100V and pass up to 5A.) You will control gates of the transistors with the outputs of logic gates. (Note ‘x’ here means “don’t care”: either state produces the same output.)

Part 1 (3 points): Here’s a CMOS inverter to remind you how NMOS and PMOS transistors work and how to draw them. PMOS (top) conducts when its gate is ___________. NMOS (bottom) conducts when its gate is ____________.

As drawn, how much current flows in each state?

- When A is HIGH, __________ flows.
- When A is LOW, __________ flows.

If you tied the output through a 10 kΩ resistor to ground, how much current would flow for each case? (Draw a picture.)

- When A is HIGH, __________ flows toward ____________.
- When A is LOW, __________ flows toward ____________.

If the 10 kΩ resistor at the output was tied to +5 V, how much current would flow?

- When A is HIGH, __________ flows toward ____________.
- When A is LOW, __________ flows toward ____________.
Part 2 (2 points): Hook up the motor to the two NMOS and two PMOS transistors so you can control the direction of current through the motor (and whether the current flows at all). In CMOS logic gates, the goal is to never have current flow straight from \(+5 \text{ V}\) to \(0 \text{ V}\), and the same is true here, it should be able to flow through the motor. Hint: There’s a reason the bridge in the question’s title is named after the letter H.

Part 3 (3 points): Design the logic that drives the gates of the NMOS and PMOS transistors. Hint: As is often the case, a truth table helps. Give the transistor gates names (A, B, C, D) and label them on your diagram above. Write out a truth table for each of the gates, for every possible combination of the control signals \(P\) and \(W\). Implement your truth table with gates.
6 Two Forms of Digital Comparator (5 points)

Both of these exercises are taken from Lab D1 (the first question is expanded slightly—from 2-bits to four):

6.1 4-bit Equality Detector (2 points)

Use any gates to make a comparator that detects equality between two 4-bit numbers. (This circuit, widened, is used a lot in computers, where a device often needs to watch the public “address bus,” so as to respond upon seeing its own distinctive “address.”)

Let’s call the four bits of the number A “A3 A2 A1 A0” and the four bits of the number B “B3 B2 B1 B0”.

Hint: the XOR function is a big help.... You can think of XOR as a 1-bit inequality detector.

6.2 2-bit A > B Detector (3 points)

Use any gates to make a comparator that detects when one of a pair of two bit number (call it “A”) is larger than the other (call it “B”).

You need not make the circuit symmetrical: it need not detect B > A, only A > B, versus the contrary case “A not-greater-than B”—formally, A ≤ B). Let’s call the two bits of the numbers “A1, A0, B1, B0”. A1, B1 is the more significant (left) bit.