Below is a **step-by-step** guide for designing and filling a Karnaugh map (K-map) when the required output is "input minus 2" for a 3-bit input in two's-complement representation. We will call the inputs X_2, X_1, X_0 (3 bits) and the outputs Y_2, Y_1, Y_0 (the 3-bit result of Input $-$ 2).

---

1. Determine the Arithmetic Interpretation

Since the problem states "3-bit input, output is input minus 2," we interpret:

- The inputs ($X_2X_1X_0$) as a **3-bit two's-complement** number.
- The output ($Y_2Y_1Y_0$) must also be a **3-bit two's-complement** number that equals the (signed) input minus 2.

  Note: If you are doing an unsigned interpretation instead, you must clarify how negative results are handled. But in most "minus 2" examples using 3 bits, two's complement is the usual approach.

---

2. Write Out the Truth Table

List **all 8 possible input combinations** of (X_2, X_1, X_0) from `000` to `111`, interpret each as a signed 3-bit integer, **subtract 2**, then represent the result again as a 3-bit two's-complement number (Y_2, Y_1, Y_0).

Below is the complete table:

Inputs (dec)	$X_2X_1X_0$	Input Value (2's comp)	Input - 2	Output Value (2's comp)	$Y_2Y_1Y_0$
0	0 0 0	0	-2	110	1 1 0
1	0 0 1	1	-1	111	1 1 1
2	0 1 0	2	0	000	0 0 0
3	0 1 1	3	1	001	0 0 1
4	1 0 0	-4 (if 2's comp)	-6	010? (Check arithmetic)	see note ↓
5	1 0 1	-3	-5	???	see note ↓
6	1 1 0	-2	-4	100	1 0 0
7	1 1 1	-1	-3	101	1 0 1

Important: If your inputs above 3 (binary 100 to 111) are intended as $-4, -3, -2, -1$ in 2's complement, then "input minus 2" means:

$-4 - 2 = -6$ (which, in 3-bit 2's-complement, wraps around to 010? That's actually $+2$ in strict

Very often, in these exercises, one interprets only `000` through `011` as 0 to 3 and maybe sets "don't care" for higher inputs, or uses extended arithmetic. Make sure you clarify the interpretation for inputs like `100` through `111`.

If you actually want "unsigned 4 minus 2 = 2," then that's `010` and so on.

Either way, fill out each row to get the correct $(Y_2Y_1Y_0)$.

(Below is a consistent table if you interpret the inputs 4..7 (100..111) as unsigned 4..7, then subtract 2, staying in plain binary—common for simpler digital design labs.)

$X_2X_1X_0$	In (unsigned)	In - 2	Out (3-bit binary)	$Y_2Y_1Y_0$
0 0 0	0	-2 (*)	110 (if forced)	1 1 0
0 0 1	1	-1 (*)	111 (if forced)	1 1 1
0 1 0	2	0	000	0 0 0
0 1 1	3	1	001	0 0 1
1 0 0	4	2	010	0 1 0
1 0 1	5	3	011	0 1 1
1 1 0	6	4	100	1 0 0
1 1 1	7	5	101	1 0 1

---

3. Build One K-Map Per Output Bit

You will have **three separate K-maps**—one for Y_2, one for Y_1, and one for Y_0. Each K-map has 3 inputs (X_2, X_1, X_0), so it's an 8-cell K-map:

1. **Label the Rows** with $X_2 = 0$ (top) and $X_2 = 1$ (bottom).
2. **Label the Columns** for (X_1, X_0) in Gray-code order: `00`, `01`, `11`, `10`.

	00	01	11	10
X2=0				
X2=1				

4. Fill in Each K-Map

4.1 K-map for Y_2

- For each row ($X_2X_1X_0$) in your truth table, if $Y_2 = 1$ then put a `1` in that cell. Otherwise put `0`.

4.2 K-map for Y_1

- Repeat, but now place `1` in the K-map whenever $Y_1 = 1$.

4.3 K-map for Y_0

- Same procedure for Y_0.

After this, you have three separate 3-variable K-maps, each with 1s and 0s placed accordingly.

---

5. Group the 1-Cells and Simplify (SOP)

For each K-map:

1. **Identify groups** of adjacent 1's in powers of 2 (1, 2, 4, or 8). Cells are adjacent if they differ by only one bit. Wrap around edges if your K-map supports it.
2. **Write each group** as a simplified product (AND) term, omitting the variable(s) that flip within that group.
3. **Sum (OR)** all group terms to get the final expression for that output bit.

Do this separately for Y_2, Y_1, and Y_0. The result is a set of **three SOP expressions**—one for each output bit.

---

6. Verify or Simulate

- **Check correctness** by comparing your simplified expressions to the original truth table rows.
- Optionally, **simulate** in a logic simulator or a tool like LTspice/Logisim to confirm that for each (X_2, X_1, X_0) input, your circuit outputs exactly "input minus 2."

Final Remarks

- The key steps are:
    1. **Form the correct truth table** (deciding on two's-complement or unsigned interpretation).
    2. **Build a K-map for each output bit**.
    3. **Fill and simplify** each K-map.

That completes the design problem for "3-bit input, output is input minus 2."