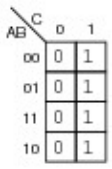
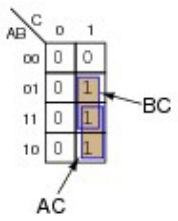



Somewhere, there must have been a mistake made in the first student's grouping of 1's in the Karnaugh map, because the map shown above is the only one proper for an answer of C, and it is not the same as the real map for the given truth table. Explain where the mistake was made, and what the proper grouping of 1's should be.

Reveal answer

Proper grouping of 1's in the Karnaugh map:




**Notes:**

The purpose of this question is to illustrate how it is incorrect to identify clusters of arbitrary size in a Karnaugh map. A cluster of three, as seen in this scenario, leads to an incorrect conclusion. Of course, one could easily quote a textbook as to the proper numbers and patterns of 1's to identify in a Karnaugh map, but it is so much more informative (in my opinion) to illustrate by example. Posing a dilemma such as this makes students *think* about why the answer is wrong, rather than asking them to remember seemingly arbitrary rules.

- **Question 10**

State the rules for properly identifying common groups in a Karnaugh map.

Reveal answer

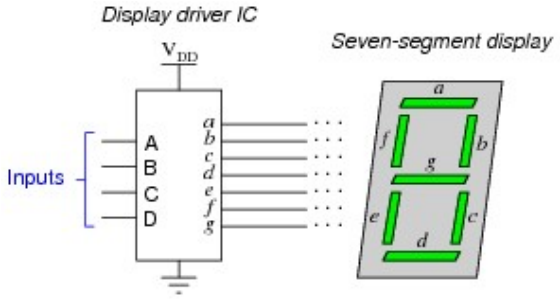
Any good introductory digital textbook will give the rules you need to do Karnaugh mapping. I leave you to research these rules for yourself!

**Notes:**

The answer speaks for itself here - let your students research these rules, and ask them exactly where they found them (including the page numbers in their textbook(s)!).

- **Question 11**

A *seven segment* *decoder* is a digital circuit designed to drive a very common type of digital display device: a set of LED (or LCD) segments that render numerals 0 through 9 at the command of a four-bit code:



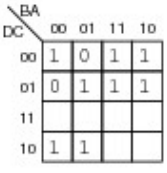

The behavior of the display driver IC may be represented by a truth table with seven outputs: one for each segment of the seven-segment display (a through g). In the following table, a "1" output represents an active display segment, while a "0" output represents an inactive segment:

| D | C | B | A | a | b | c | d | e | f | g | Display |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | "0" |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | "1" |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | "2" |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | "3" |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | "4" |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | "5" |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | "6" |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | "7" |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | "8" |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | "9" |

A real-life example such as this provides an excellent showcase for techniques such as Karnaugh mapping. Let's take output a for example, showing it without all the other outputs included in the truth table:

| D | C | B | A | a |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |

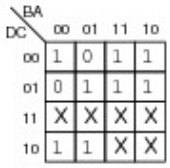
Plotting a Karnaugh map for output a, we get this result:




Identify adjacent groups of 1's in this Karnaugh map, and generate a minimal SOP expression from those groupings.

Note that six of the cells are blank because the truth table does not list all the possible input combinations with four variables (A, B, C, and D). With these large gaps in the Karnaugh map, it is difficult to form large groupings of 1's, and thus the resulting "minimal" SOP expression has several terms.

However, if we do not care about output a's state in the six non-specified truth table rows, we can fill in the remaining cells of the Karnaugh map with "don't care" symbols (usually the letter X) and use those cells as "wildcards" in determining groupings:

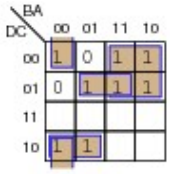



With this new Karnaugh map, identify adjacent groups of 1's, and generate a minimal SOP expression from those groupings.

Reveal answer

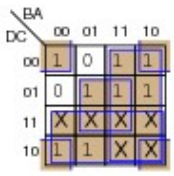
Karnaugh map groupings with strict "1" groups:

$$\overline{D}B + \overline{D}CA + D\,\overline{C}\,\overline{B} + \overline{C}\,\overline{B}\,\overline{A}$$




Karnaugh map groupings with "don't care" wildcards:

$$D + B + CA + \overline{C}\,\overline{A}$$



Follow-up question: this question and answer merely focused on the a output for the BCD-to-7-segment decoder circuit. Imagine if we were to approach all seven outputs of the decoder circuit in these two fashions, first developing SOP expressions using strict groupings of "1" outputs, and then using "don't care" wildcards. Which of these two approaches do you suppose would yield the simplest gate circuitry overall? What impact would the two different solutions have on the decoder circuit's behavior for the six unspecified input combinations 1010, 1011, 1100, 1101, 1110, and 1111?

**Notes:**

One of the points of this question is for students to realize that bigger groups are better, in that they yield simpler SOP terms. Also, students should realize that the ability to use "don't care" states as "wildcard" placeholders in the Karnaugh map cells increases the chances of creating bigger groups.

Truth be known, I chose a pretty bad example to try to make an SOP expression from, since there are only two non-zero output conditions out of ten! Formulating a POS expression would have been easier, but that's a subject for another question!

- **Question 12**

When designing a circuit to emulate a truth table such as this where nearly all the input conditions result in "1" output states, it is easier to use Product-of-Sums (POS) expressions rather than Sum-of-Products (SOP) expressions:

| A | B | C | Output |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |