8. Multi-Output Networks & 7-Segment Displays

8.1 Multi-output Minimisation

- Until now, we have considered minimisation on an individual level. In other words, we have minimised logic for a single output for a given set of inputs.

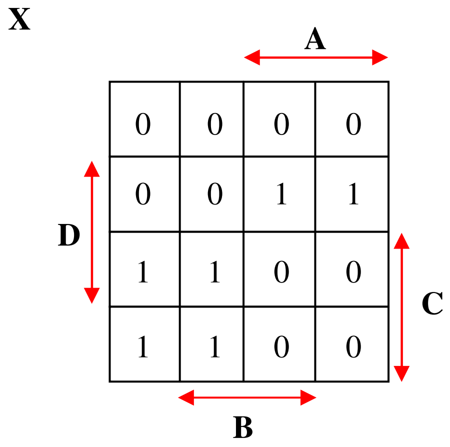
- However, many digital circuits have multiple outputs associated with the same set of inputs.

- For example, consider the JK counter in the previous set of notes. This had 4 flipflops, with two inputs each. Thus, we had 8 outputs in total (and 8 Karnaugh Maps) associated with the same set of inputs.

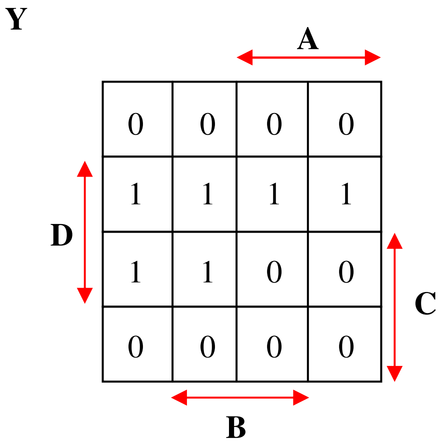
- Such circuits **require the implementation of several functions for the same set of input variables**.

- As such, it is useful to solve these functions from a collective viewpoint, i.e. we look to minimise the logic across all outputs simultaneously (and not treat each output individually).

- By way of a simple illustration, consider the following two Karnaugh Maps for arbitrary functions X and Y:



0	0	0	0
0	0	1	1
1	1	0	0
1	1	0	0



0	0	0	0
1	1	1	1
1	1	0	0
0	0	0	0

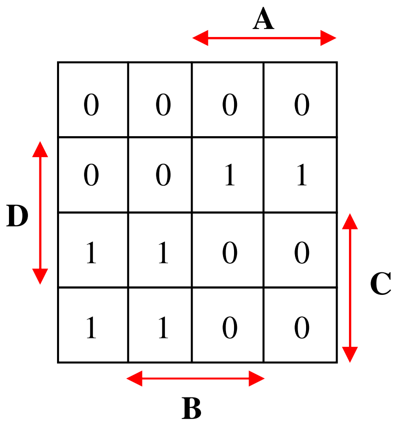
- Minimising each function individually gives:

$$X = \bar{A}C + A\bar{C}D \qquad \text{and} \qquad Y = \bar{C}D + \bar{A}D$$

- This requires a total of 6 gates (4 x AND and 2 x OR).

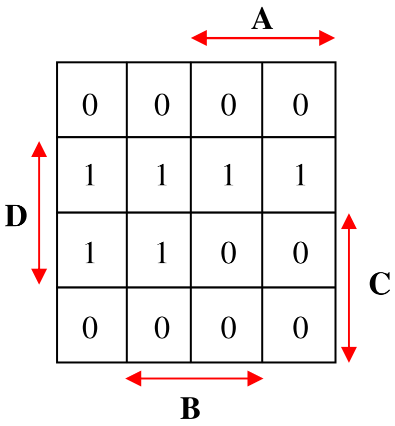
- However, if we consider both maps collectively, we obtain the following minimal solution:

X



0	0	0	0
0	0	1	1
1	1	0	0
1	1	0	0

Y



0	0	0	0
1	1	1	1
1	1	0	0
0	0	0	0

- Minimising collectively gives:

$$X = \overline{A}C + A\overline{C}D \qquad \text{and} \qquad Y = A\overline{C}D + \overline{A}D$$

- This requires a total of 5 gates (3 x AND and 2 x OR).

- Hence, although we do not have a minimal individual solution for function Y, we have nevertheless an overall multi-output solution that is minimal in terms of gates required.

- This is because ***we are reusing available gates from other functions***. For example, we use the AND gate $A\overline{C}D$ for function Y as this is already required by function X.

- This technique is referred to as **multi-output (or multiple-output) minimisation.**

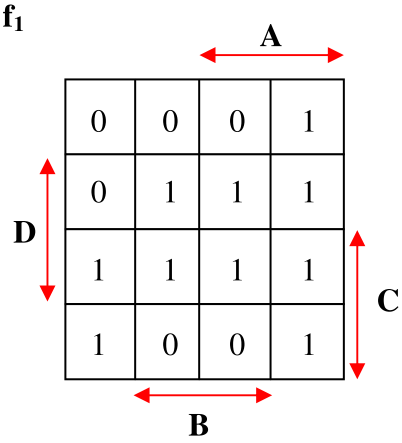
- ***Ex. 8.1 Using multi-output minimisation, design a circuit with 4 inputs and 3 outputs which implements the functions:***

$$f_{1(A,B,C,D)} = \Sigma(2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$$

$$f_{2(A,B,C,D)} = \Sigma(2, 3, 5, 6, 7, 10, 11, 14, 15)$$

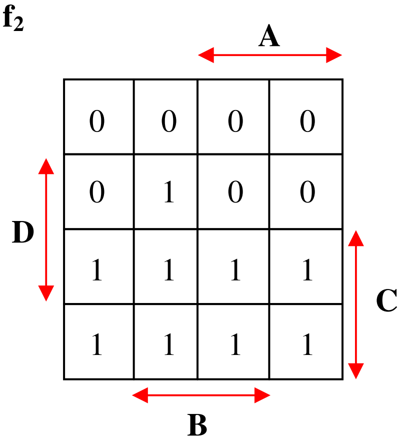
$$f_{3(A,B,C,D)} = \Sigma(6, 7, 8, 9, 13, 14, 15)$$

f_1



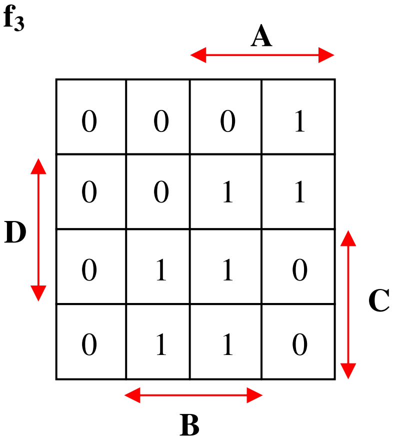
0	0	0	1
0	1	1	1
1	1	1	1
1	0	0	1

f_2



0	0	0	0
0	1	0	0
1	1	1	1
1	1	1	1

f_3



0	0	0	1
0	0	1	1
0	1	1	0
0	1	1	0

- Assuming that complemented inputs are available, this **gives an 8 gate solution** (3 x OR and 5 x AND) as follows:

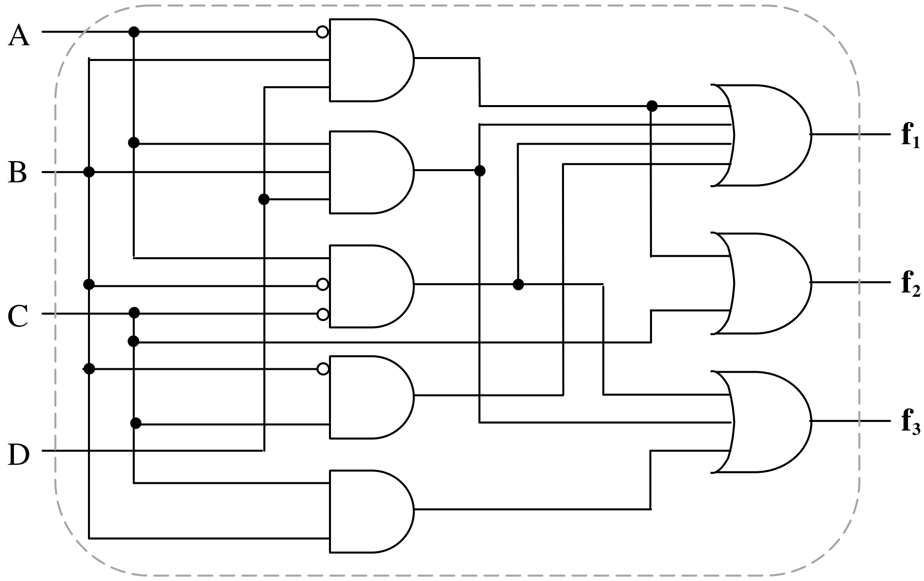
$$f_1 = \overline{A}BD + ABD + A\overline{B}\,\overline{C} + \overline{B}C$$

$$f_2 = C + \overline{A}\underline{BD}$$

$$f_3 = BC + A\overline{B}\,\overline{C} + ABD$$

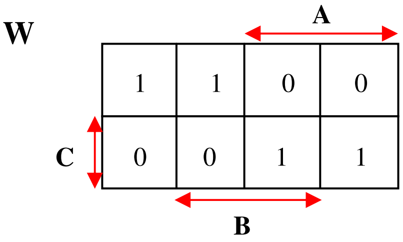
- Note – individually, these are not minimal solutions. However, **had we minimised each function individually**, we would require an overall **total of 10 gates**. *Verify this yourself.*

- Finally, the overall circuit implementation is as follows:

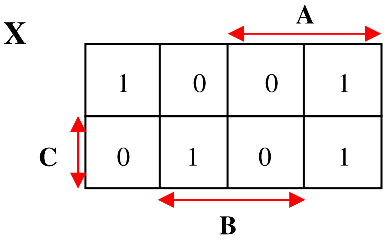


- ***Ex. 8.2 Using multi-output minimisation, obtain a minimal implementation of the following four functions:***

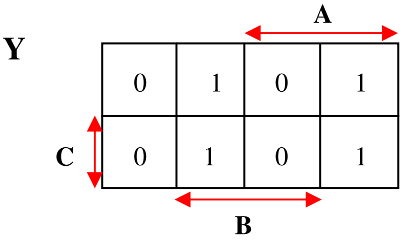
$$W_{(A,B,C)} = \Sigma(0, 2, 5, 7);\ \ X_{(A,B,C)} = \Sigma(0, 3, 4, 5);\ \ Y_{(A,B,C)} = \Sigma(2, 3, 4, 5);\ \ Z_{(A,B,C)} = \Sigma(2, 5, 7)$$



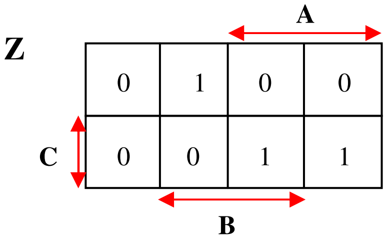
1	1	0	0
0	0	1	1



1	0	0	1
0	1	0	1



0	1	0	1
0	1	0	1



0	1	0	0
0	0	1	1

- Assuming that complemented inputs are available, this gives a 9 gate solution (4 x OR and 5 x AND) as follows:

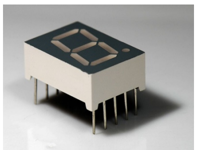
$$W = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + AC$$

$$X = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}BC + A\overline{B}$$

$$Y = \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}$$

$$Z = \overline{A}B\overline{C} + AC$$

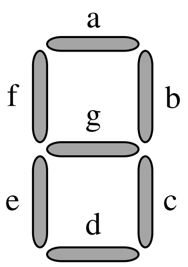
8.2 The 7-segment display



- A widely used and commonly seen multi-output unit is the 7-segment display.

- These displays are used in conjunction with logic circuits that can decode a BCD (binary coded decimal) number to activate the appropriate digits on the display.

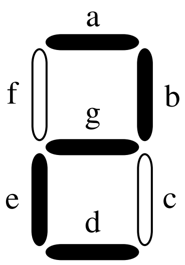
- The 7-segment displays can be found in a wide range of digital devices including watches, alarm clocks, microwave ovens, petrol pumps, etc.



- In this section of the notes, we will look at designing such a decoder circuit, but first we will examine the actual 7-segment display in a little more detail.

- The diagram to the right shows the seven segments (hence the name!) that make up the display.

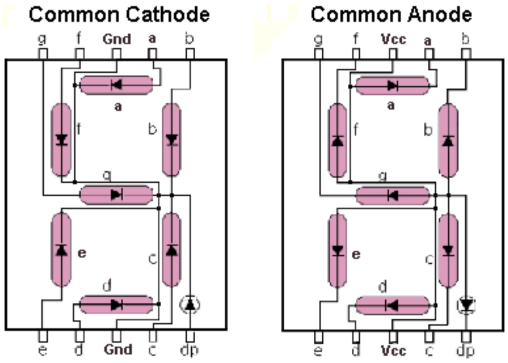
- The segments are labelled a, b, c, d, e, f and g as shown.



- Each of these segments constitutes a single LED which can be turned on or left off as required.

- Thus, for example, if we wish to display the digit ‘2’ on the display, we would need to turn ON segments a, b, d, e and g, i.e.:

- There are two different ways in which the 7-segment display can be configured and hence, there are two different ways in which the segments can be turned on.



Common Cathode Mode ...

- In common **cathode (negative)** mode, each segment is **connected to ground** (i.e. all segments have a common ground connection), as shown below.

- Thus, in order to turn the segment on, we need to **add a logic '1' (i.e. 5V) to its input**.

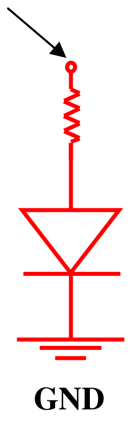
Common Anode Mode ...

- In common **anode (positive)** mode, each segment is **connected to power** (i.e. all segments have a common 5V connection), as shown below.

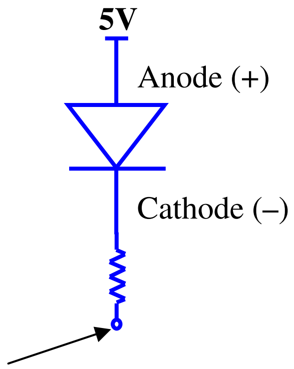
- Thus, in order to turn the segment on, we need to **add a logic '0' (i.e. ground) to its input**.

Common Cathode Mode
- logic '1' to turn on

5V placed here will cause LED to emit light



Common Anode Mode
- logic '0' to turn on



GND (Ground) placed here will cause LED to emit

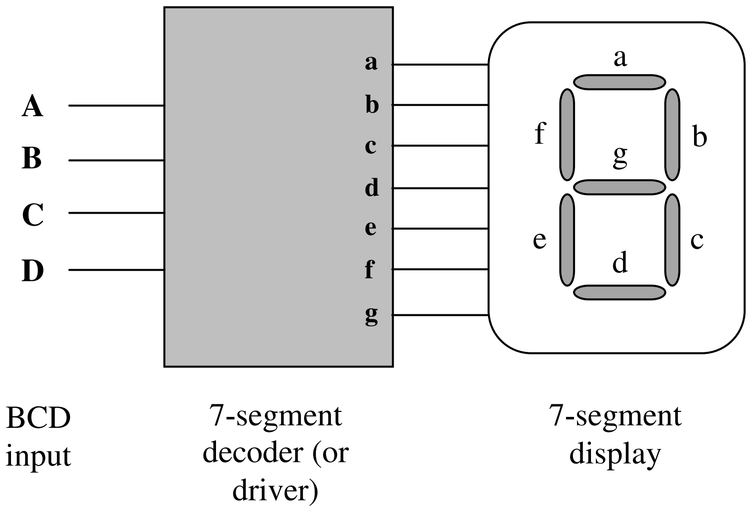
BCD to 7-segment decoder

- A BCD to 7-segment decoder is a combinational circuit that accepts a decimal digit in BCD and generates the appropriate outputs for selection of segments in the 7-segment display indicator.

- Recall that the 8421 BCD represents each decimal digit from 0 to 9 using a 4-bit binary code.

- The seven outputs of the decoder (a, b, c, d, e, f, g) select the corresponding segments in the 7-segment display, as shown previously.

- The overall setup for the circuit is as follows:

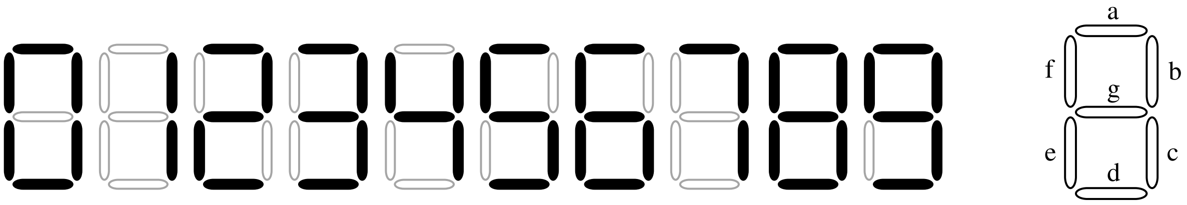


- ***Ex. 8.3 Design a BCD to 7-segment decoder for use with a 8421 BCD counter. Invalid states of the counter are to result in a blank display. Implement using NAND gates. Note that the display is connected in common cathode mode.***

- A single 7-segment display can only display digits 0 to 9. An 8421 BCD counter can count from 0 to 15. Hence, the counter outputs 10 to 15 are invalid states.

- The 7-segment display is connected in cathode mode and therefore we need to apply a logic '1' in order to switch a segment ON.

- The 10 digits to display are as follows:



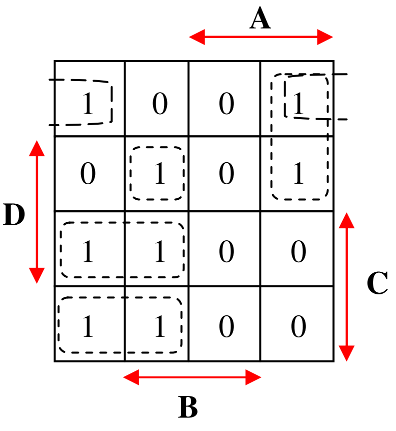
- From this, we can determine a table that shows which of the seven segments need to be switched on and when they need to be switched on, as follows:

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

- The next step is to derive a Karnaugh Map for each segment and to carry out multi-output minimisation in order to achieve an overall minimal solution.

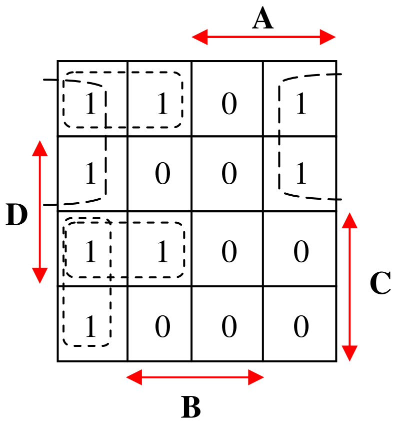
- Unused or invalid states are required to be blank in this case and so need to be inserted as a logic ‘0’ in the Karnaugh Maps.

a



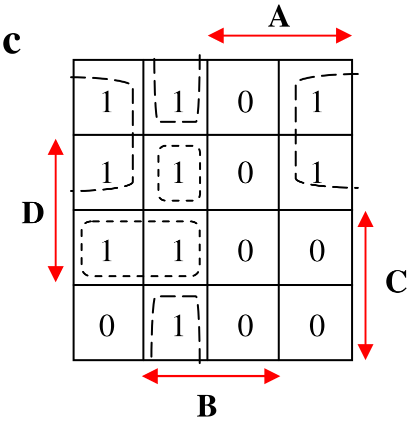
1	0	0	1
0	1	0	1
1	1	0	0
1	1	0	0

b



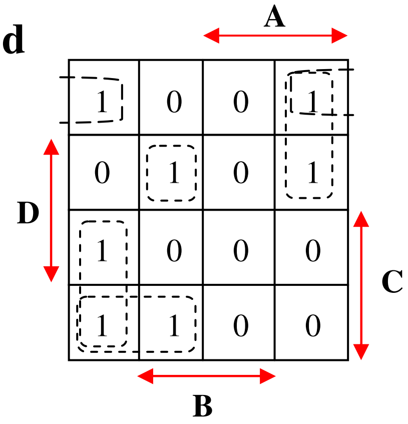
1	1	0	1
1	0	0	1
1	1	0	0
1	0	0	0

c



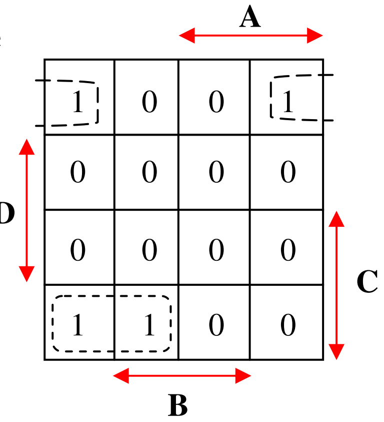
1	1	0	1
1	1	0	1
1	1	0	0
0	1	0	0

d



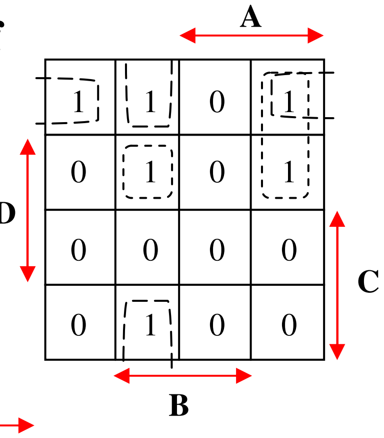
1	0	0	1
0	1	0	1
1	0	0	0
1	1	0	0

e



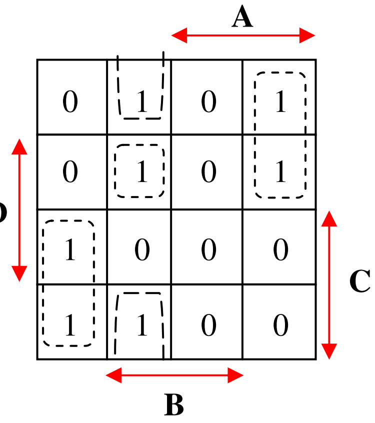
1	0	0	1
0	0	0	0
0	0	0	0
1	1	0	0

f



1	1	0	1
0	1	0	1
0	0	0	0
0	1	0	0

g



0	1	0	1
0	1	0	1
1	0	0	0
1	1	0	0

- This gives, in NAND-only implementation:

a $= A\bar{B}\bar{C} + \bar{A}B\bar{C}D + \bar{A}CD + \bar{A}C\bar{D} + \bar{B}\bar{C}\bar{D} = \overline{\overline{(A\bar{B}\bar{C})}.\overline{(\bar{A}B\bar{C}D)}.\overline{(\bar{A}CD)}.\overline{(\bar{A}C\bar{D})}.\overline{(\bar{B}\bar{C}\bar{D})}}$

b $= \bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}\bar{C}\bar{D} + \bar{A}CD = \overline{\overline{(\bar{B}\bar{C})}.\overline{(\bar{A}\bar{B}C)}.\overline{(\bar{A}\bar{C}\bar{D})}.\overline{(\bar{A}CD)}}$

c $= \bar{B}\bar{C} + \bar{A}B\bar{D} + \bar{A}B\bar{C}D + \bar{A}CD = \overline{\overline{(\bar{B}\bar{C})}.\overline{(\bar{A}B\bar{D})}.\overline{(\bar{A}B\bar{C}D)}.\overline{(\bar{A}CD)}}$

d $= A\bar{B}\bar{C} + \bar{A}C\bar{D} + \bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C + \bar{A}B\bar{C}D = \overline{\overline{(A\bar{B}\bar{C})}.\overline{(\bar{A}C\bar{D})}.\overline{(\bar{B}\bar{C}\bar{D})}.\overline{(\bar{A}\bar{B}C)}.\overline{(\bar{A}B\bar{C}D)}}$

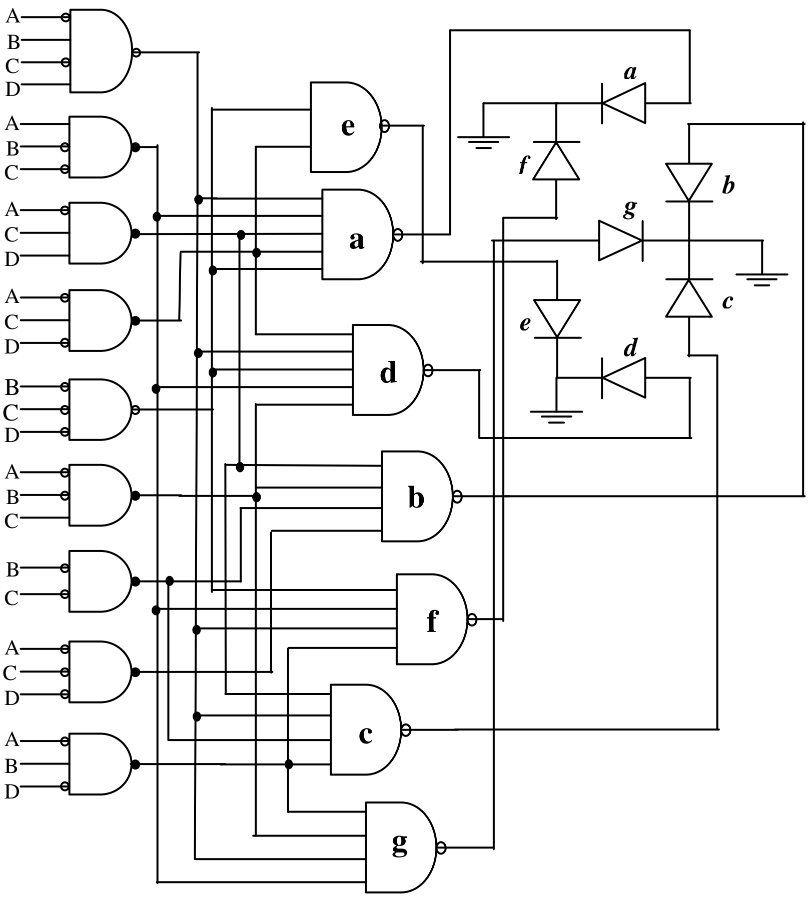
e $= \bar{A}C\bar{D} + \bar{B}\bar{C}\bar{D} = \overline{\overline{(\bar{A}C\bar{D})}.\overline{(\bar{B}\bar{C}\bar{D})}}$

f $= A\bar{B}\bar{C} + \bar{A}B\bar{D} + \bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D = \overline{\overline{(A\bar{B}\bar{C})}.\overline{(\bar{A}B\bar{D})}.\overline{(\bar{B}\bar{C}\bar{D})}.\overline{(\bar{A}B\bar{C}D)}}$

g $= A\bar{B}\bar{C} + \bar{A}B\bar{D} + \bar{A}\bar{B}C + \bar{A}B\bar{C}D = \overline{\overline{(A\bar{B}\bar{C})}.\overline{(\bar{A}B\bar{D})}.\overline{(\bar{A}\bar{B}C)}.\overline{(\bar{A}B\bar{C}D)}}$

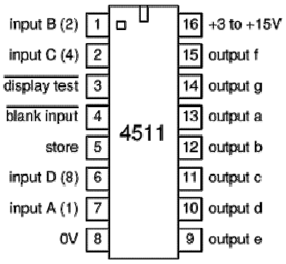
- This yields a 16 NAND gate solution. *Had we minimised individually, we would have obtained a 22 NAND gate solution which would require an additional two NAND ICs!*

- Thus, the final decoder circuit looks like:



- Note – we could have made the minimisation a little easier, if given the choice, by using don’t care terms for the unused states instead.

- Finally, the standard 8421 BCD to 7-segment decoder (or driver) comes in chip format – namely the 5V CMOS 4511 IC.



input B (2) 1
input C (4) 2
display test 3
blank input 4
store 5
input D (8) 6
input A (1) 7
0V 8
4511
16 +3 to +15V
15 output f
14 output g
13 output a
12 output b
11 output c
10 output d
9 output e