HMK # 5 Solutions

2) BINARY NUMBERS — 8-bit representation

(59)

$-59 \rightarrow$ Negative number $\rightarrow$ Signed Number Representation:

$(-59)_{10} = (\underbrace{1}_{\text{sign bit}}\ \underbrace{0111011}_{\text{Magnitude}})_2 = X$

$59 \rightarrow$ Positive number $\rightarrow$ unsigned Number Representation:

$(59)_{10} = (00111011)_2 = Y$

1's Complement Representation $= \bar{Y} = (11000100)_2$

2's Complement Representation $= \bar{Y}+1 = (11000101)_2$

$11000100 + 1 = 11000101$

(126)

$(126)_{10} = (01111110)_2$

$Y = 01111110$

$(-126)_{10} = (1\ 1111110)_2$ ⇨ It is not asked

sign bit — magnitude

Signed: 01111110

This is also not asked.

1's Complement $= \bar{Y} = (10000001)_2$

2's Complement $= \bar{Y}+1 = (10000010)_2$

1's complement: 01111110

2's compl: 01111110

NOTE: 1's and 2's complement representations derivations are needed only for negative (signed) numbers.

Obs - In exams, if you answer out of order and I can't find it (in this case, -59-500-126-500 was the order-

10-bit Representation

(500) $(500)_{10} = (0\,1111\,10100)_2$

$X = 0\,1111\,10100$

$(-500)_{10} =$ ~~$(1\,1111\,10100)_2$~~ there are the answers

sign bit ↑ Magnitude for -500

1's complement $\Rightarrow \overline{X} = (1000001011)_2$

2's complement $\Rightarrow \overline{X}+1 = (1000001100)_2$

$$\begin{array}{r} 1000001011 \\ + \qquad\quad 1 \\ \hline 1000001100 \end{array}$$

(3) Adding / Subtracting in 2's complement

$X = -200$

$Y = 100$

$(-200)_{10} = (11001000)_2$

$(100)_{10} = (001100100)_2$

$(200)_{10} = (011001000)_2$

1's complement

↳ 100110111

$$\begin{array}{r} 100110111 \\ + \qquad\quad 1 \\ \hline \end{array}$$

(-200) 2's comple. → 100111000

$\therefore$ $X+Y =$

$$\begin{array}{r} 100111000 \\ + \; 001100100 \\ \hline \end{array}$$

NO OVERFLOW! ← 011001110 0 ⇐ Result in 2's complement Notation

11001101 1 ⇐ Result in one's complement Notation

00110010 0 ⇐ Result in binary form

$-(001100100)_2 = -(100)_{10} = (-100)_{10}$ ✓

$X - Y = (-200) + (-100) = (-X) + (-Y)$

$(-200)_{10} = (100111000)_2$ ← two's complement

$(100)_{10} = 001100100$

$+ \quad 11001101\,1$ ← one's complement

$(-100)_{10} = (110011100)_2$ ← 2's complement

$(-X) + (-Y) =$ $\boxed{10}$ ← OVERFLOW

100111000

$+ \quad 110011100$

OVERFLOW!

CARRY (IGNORE FOR THE ANSWER) ← $(1)01101\,0100$

$01101\,00111$ ← one's complement

Result → $(100101100)_2$ ← binary number rep.

$= (300)_{10}$ ✓

Note: The answer should have been −300 but we would have needed to represent these numbers (operands) with 10-bits so we would be able to represent the sign bit for the result.

$X = 239 \quad Y = -56$

$X = (239)_{10}$

$-X = (-239)_{10}$

$Y = (56)_{10}$

$-Y = (-56)_{10}$

$(239)_{10} = 011101111$

$$\begin{array}{r} 100010000 \\ + \quad\quad 1 \\ \hline \end{array}$$

$(-239)_{10} = (100010001)_2 \Leftarrow$ 2's complement rep.

$(56)_{10} = 000111000$

$$\begin{array}{r} 111000111 \\ + \quad\quad 1 \\ \hline \end{array}$$

$(-56)_{10} = 111001000 \Leftarrow$ 2's complement rep.

$$X + Y = X + (-Y) = \begin{array}{r} \boxed{11} \\ 011101111 \\ + \; 111001000 \\ \hline (1)\,010110111 \end{array}$$

There is no OVERFLOW. The result is "right"

~~overflow~~ CARRY (IGNORE)

After truncating '1' from ~~overflow~~ CARRY we obtain: $(010110111)_2$

$= (183)_{10}$ ✓

$$X - Y = X - (-Y) = X + Y$$

$$= \begin{array}{r} 011101111 \\ + \; 000111000 \\ \hline \end{array}$$

NO OVERFLOW!

$0(1001001111)_2 = (295)_{10}$ ✓

$X = 64$

$Y = 243$

$(64)_{10} = 001000000$

11011111

$+ \quad 1$

$(-64)_{10} = (11000000)_2$ ⇐ 2's complement.

$(243)_{10} = 011110011$

$10000110 \quad 0$

$+ \quad 1$

$(-243)_{10} = (100001101)_2$ ⇐ 2's complement

$X + Y =$ 01 00 1000000 There is overflow.

$+ \quad 011110011$

(1001100011)$_2$ = $(307)_{10}$ ✓

~~NO~~ OVERFLOW!

Note: The 2's compl. representation of this number does not fit in 9 bits.

This is a negative number (not the answer we were expecting. That is corroborated by the detection of overflow.)

$X - Y = X + (-Y) =$ 001000000

$+ \quad 100001101$

~~NO~~ OVERFLOW! CARRY (1 0 1 00 1 1 0 1)$_2$

101001100 ← one's comp.

✓ $(179)_{10} = (10110011)_2$ ← binary

$X = -177$

$Y = -156$

$(177)_{10} = (010110001)_2$

$10100111\,0$
$+\qquad\quad 1$

$(-177)_{10} = (101001111)_2 \Leftarrow$ 2's compl. rep.

$(156)_{10} = (010011100)_2$

$10110001\,1$
$+\qquad\quad 1$

$(-156)_{10} = (101100100)_2 \Leftarrow$ 2's comp. rep.

$X + Y = (-X) + (-Y) =$

$\begin{array}{r} 10\,1\,0\,0\,1\,1\,1\,1 \\ +\ 1\,0\,1\,1\,0\,0\,1\,0\,0 \\ \hline (1)\,0\,1\,0\,1\,1\,0\,0\,1\,1 \end{array}$

OVERFLOW! ← CARRY (disregard)

$(010110011)_2 \Leftarrow$ 1's comp. result rep.

$(010110010)_2 \Leftarrow$ binary result representation.

$= (333)_{10}$ ✓

NOTICE THAT WE WOULD NEED 10-BITS TO REPRESENT NEGATIVE 300 AND 33 !!

$X - Y = (-X) - (-Y) = (-X) + Y$

NO OVERFLOW!

$\begin{array}{r} 10100\,1111\ (-177_{10}) \\ +\ 0100111\,00\ (156)_{10} \\ \hline (1)\,1111010\,11 \Leftarrow \text{2's complement} \end{array}$

$1111010\,10 \Leftarrow$ 1's complement

$0000101\,01 \Leftarrow$ binary represent.

$-(000010101)_2 = -(21)_{10}$ ✓

4) Digital Codes

NOTE: Adding '1' will make the numbers of "1's" odd.

8-bit ASCII code

a) MSB with odd parity ← # of 1's = 4, 4 is even so parity bit = 1

M = $(77)_{10}$ = $(1001101)_2$ Parity bit = 1

a = $(97)_{10}$ = ~~$(1000011)_2$~~ $(1100001)_2$ Parity bit = 0

r = $(114)_{10}$ = $(1110010)_2$ Parity bit = 1

c = $(99)_{10}$ = $(1100011)_2$ Parity bit = 1

e = $(101)_{10}$ = $(1100101)_2$ Parity bit = 1

l = $(108)_{10}$ = $(1101100)_2$ Parity bit = 1

l = $(108)_{10}$ = $(1101100)_2$ Parity bit = 1

o = $(111)_{10}$ = $(1101111)_2$ Parity bit = 1

M a r c

11001101 - 01100001 - 11110010 - 11100011

e l l o

11100101 - 11101100 - 11011100 - 11101111

b) BCD representations for unsigned decimal numbers have the range from 0 to 9 (0000 to 1001). 4-bit patterns not lying on that range should not occur on digital circuits and are considered as "don't cares".

- 0011 1100 0101 1010 . 0111 1110

- 3 X 5 X 7 X

c)

- 1 2 . 3 4

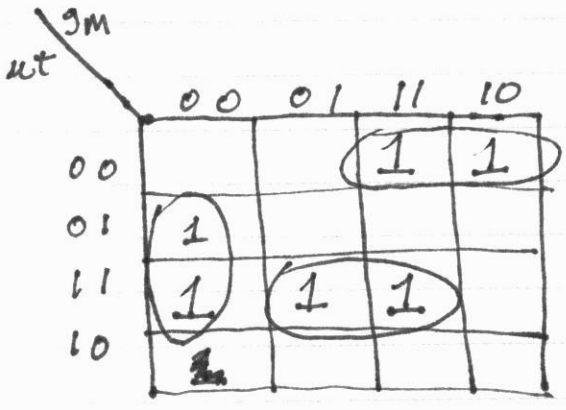
- 0001 0010 . 0011 0100

- 7 6 9 8 . 5 6 1

- 0111 0110 1001 1000 . 0101 0110 0001

(5) $F = \bar{g}\bar{m}\bar{u}t + gmut + \bar{g}mut + g\bar{m}\bar{u}\bar{t} + gm\bar{u}\bar{t} + \bar{g}\bar{m}ut$

g	m	u	t	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



Function to be implemented on VHDL:

$F = g\bar{u}\bar{t} + \bar{g}\bar{m}t + mut$

VHDL CODE:

```
Library ieee;
use ieee.std_logic_1164.all;

entity hwk5 is
 port ( g: in std_logic;
        m, u, t : in std_logic;
          F : out std_logic );
end hwk5;

architecture hwk5_arch of hwk5 is
begin
      F <= (g and (not u) and (not t))   OR
           ((not g) and (not m) and t)   OR
           ( m  and  u  and  t );

end hwk5_arch;
```

6) Tabular Minimization
Quine – McKluskey Method.

$$Q_{a,b,c,d} = \sum 0, 1, 3, 4, 7, 11, 13, 15 + d(9, 12, 14)$$

First make a list of minterms from the function and look for pairs that differ with a single bit.

Example: midterm 0 and midterm 1

0 0 0 0 — 0 0 0 1

Only differ with the last bit so they can be combined as 0, 1 – 0 0 0 X

List 1 ⟶ combining to make List 2

List 1		List 2		
0	0 0 0 0 ✓			
1	0 0 0 1 ✓	(1) 0, 1	0 0 0 X	
3	0 0 1 1 ✓	(2) 0, 4	0 X 0 0	
4	0 1 0 0 ✓	1, 3	0 0 X 1 ✓	
7	0 1 1 1 ✓	1, 9	X 0 0 1 ✓	
9	1 0 0 1 ✓	3, 7	0 X 1 1 ✓	
11	1 0 1 1 ✓	3, 11	X 0 1 1 ✓	
12	1 1 0 0 ✓	(3) 4, 12	X 1 0 0	
13	1 1 0 1 ✓	(4) 7, 15	X 1 1 1 ✓	This is included in (5)
14	1 1 1 0 ✓	9, 11	1 0 X 1 ✓	
15	1 1 1 1 ✓	9, 13	1 X 0 1 ✓	
		11, 15	1 X 1 1 ✓	
		12, 13	1 1 0 X ✓	
		12, 14	1 1 X 0 ✓	
		13, 15	1 1 X 1 ✓	
		14, 15	1 1 1 X ✓	

* checks are given to minterms that are included in the combinations

- Now combining minterms from List 2 we get:

List 3

1, 3, 9, 11	X0X1 ✓
1, 9, 3, 11	X0X1 ✓
3, 7, 11, 15	XX11 (5)
9, 11, 13, 15	1XX1 ✓
9, 13, 11, 15	1XX1 ✓
12, 13, 14, 15	11XX ✓
12, 14, 13, 15	11XX ✓

- Further combining List 4

1,3,9,11, 1,9,3,11	X0X1 (6)
9, 11, 13, 15, 9, 13, 11, 15	1XX1 (7)
12, 13, 14, 15, 12, 14, 13, 15	11XX (8)

The minterms that do not have a check mark and are numbered are our prime implicants. They could not have been combined any further.

- Now we make a table that shows what minterms our prime implicants cover.
Ex. p5 covers : 0011, 0111, 1011, 1111
3 7 11 15

"don't cares not need to be listed"

Prime Implicants	Minterms 0	1	3	4	7	11	13	15
p1 = 000 X	✓	✓						
p2 = 0X00	✓			✓				
p3 = X100				✓				
p4 = X111					✓			✓
p5 = XX11			✓		✓	✓		✓
p6 = X0X1		✓	✓					
p7 = 1XX1						✓	✓	✓
p8 = 11XX							✓	✓

- Row Domination - p2 dominates p3
p5 dominates p4
p7 dominates p8

So p3, p4, p8 can be discarded

Our New Table

Prime Implicants	Minterms 0	1	3	4	7	11	13	15
P1 = 000X	✓	✓						
P2 = 0X00	✓			✓				
P5 = XX11			✓		✓	✓		✓
P6 = X0X1		✓	✓					
P7 = 1XX1						✓	✓	✓

- Now we have P2 as the only essential prime implicant. i.e. P4 is the only one that covers minterm 4. Also P5 is the only one that covers minterm 7. And P7 is the only one that covers minterm 13. After removing the essential prime implicants row and the columns they cover we get:

Prime Implicants	Minterms 1
P1 = 000X	✓
P6 = X0X1	

Now we see that P6 does not cover any minterms that are left.

Complete cover can be $C = \{P1\}$

∴ The function can be realized as $Q = \bar{a}\,\bar{b}\,\bar{c}$

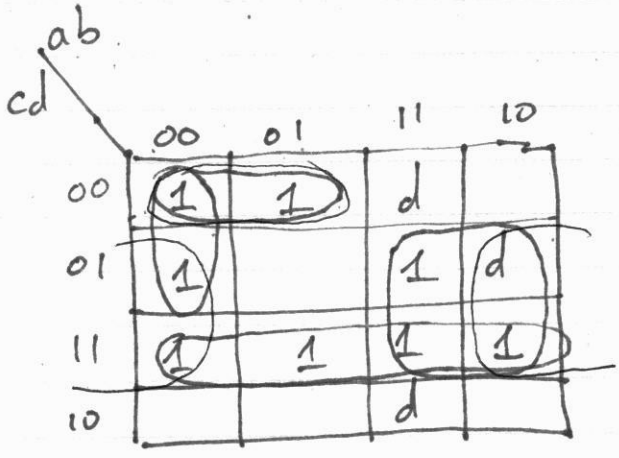
- From the previous table we can see that P6 can be discarded since it doesn't cover any minterms left.

So including our essential prime implicants and P1, our complete cover can be:

$$C = \{ P1, P2, P5, P7 \}$$

∴ The function can be realized as $Q = \bar{a}\bar{b}\bar{c} + \bar{a}\bar{d}\bar{c} + cd + ad$ ✓

We can even check this answer by minimizing it with the simpler K-map method.



$Q = \bar{a}\bar{c}\bar{d} + \bar{a}\bar{b}\bar{c} + cd + ad$ ✓

Less cost result:

$Q = \bar{a}\bar{c}\bar{d} + \underline{\bar{b}d} + ad + cd$

difference is that ~~is to~~ instead of P1 I'd have picked P6 to cover minterm 1. P6 is "cheaper" than P1.