

*Instructions: 3 hours closed book, notes. Place answers in answer boxes at the bottom of each page. Show all work in blank space above the answer box. No electronic devices permitted.*

1. Let  $X=12_{(8)}$ ,  $Y=71.1_{(8)}$

(a) Compute  $Z_{(8)}=11_{(8)} * X + Y$

(b) Express  $Z$  as a binary number

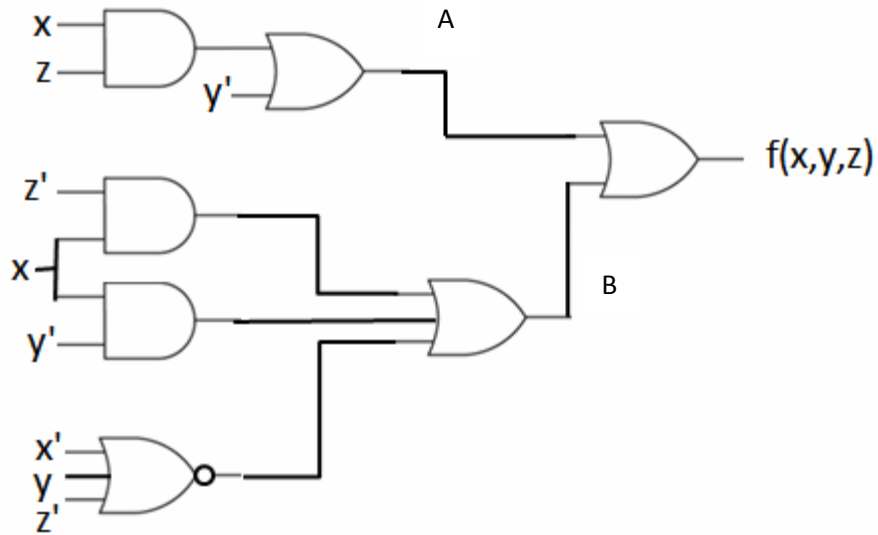
(c) What is the 8's complement of  $Y$ ?

$$\begin{array}{r}
 \text{(a) } 12 \quad 132.0 \\
 \times 11 \quad + 71.1 \\
 \hline
 12 \quad 223.1 \\
 \hline
 12 \\
 132
 \end{array}
 \quad
 \begin{array}{r}
 \text{(b) } 2 \quad 2 \quad 3 \quad . \quad 1 \\
 010 \quad 010 \quad 011 \quad . \quad 001
 \end{array}$$

(c)  $Y=71.1$ , 7's-complement of  $Y=06.6=6.6$ , 8's-complement  $6.7$

1 (a) 223.1	1 (b) 010010011.001	1 (c) 6.7
----------------	------------------------	--------------

2. The circuit diagram shown is a non-minimal realization of  $f(x,y,z)$ . Sketch the circuit diagram of a minimal realization of  $f$ . HINT: simplify  $f$ .



$$A = y' + xz, \quad B = xz' + xy' + (x' + y + z)'$$

$$= xz' + xy' + xy'z$$

$$= xz' + xy'$$

$$f = A + B = y' + xz + xz' + xy'$$

$$= y' + xy' + x(z + z')$$

$$= y' + x$$

2.



3.

$$f(w, x, y, z) = \sum m(0, 1, 2, 7, 8, 9, 10, 11)$$

(a) Sketch the Karnaugh Map for  $f$ .

(b) Write a minimal DNF realization for  $f$ .

(c) Suppose you could assign any one cell of the KM as don't-care, making  $f$  an incomplete Boolean function. Which cell would you assign in order to make the resulting minimal DNF realization as small as possible? Justify your choice.

3(a)

yz

wx

	00	01	11	10
00	1	1	0	1
01	0	0	1	0
11	0	0	0	0
10	1	1	1	1

4 PI subcubes:

4-corners ( $x'z'$ ), bottom row ( $wx'$ ), rows1&4cols1&2 ( $x'y'$ ), and  $w'xyz$

3(b)

All 4 PI's are irredundant so  $\min f = x'z' + wx' + x'y' + w'xyz$

3(c)

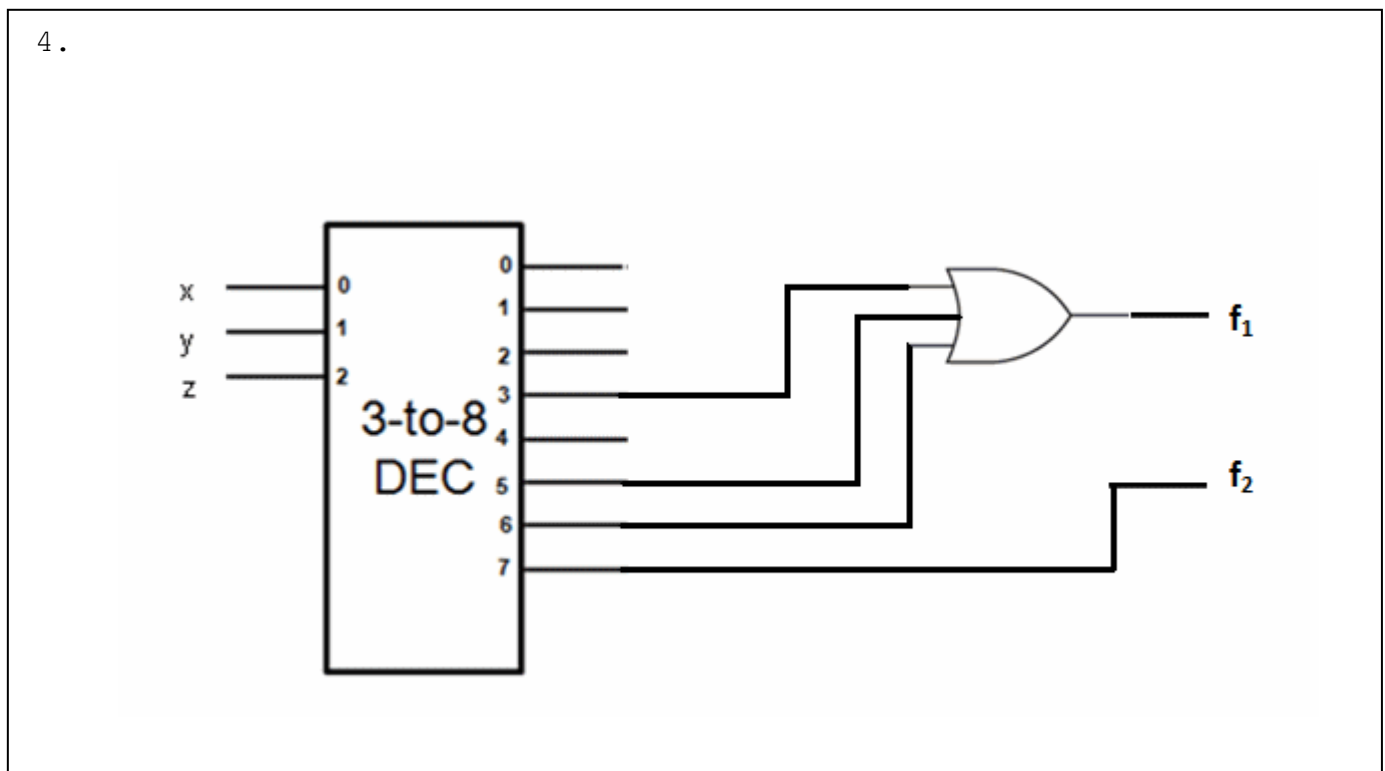
Make cell 0011 don't care, then assign it the value 1. Then  $f$  reduces down to just  $f = x' + w'yz$

4. Design a 3-input 2-output circuit so that its output  $f_1=1$  when exactly 2 of its 3 inputs are 1, and its other output  $f_2=1$  when all of its 3 inputs are 1 ( $f_2$  will be used to turn on a yellow warning lamp and  $f_1$  a red danger lamp). Use the given 3-8 decoder plus some additional gates to design this circuit.

x	y	z	$f_1$	$f_2$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

So  $f_1 = \sum m(3, 5, 6)$  and is therefore the OR of outputs 3, 5, 6.

$f_2 = \sum m(7)$  so it is just output 7.



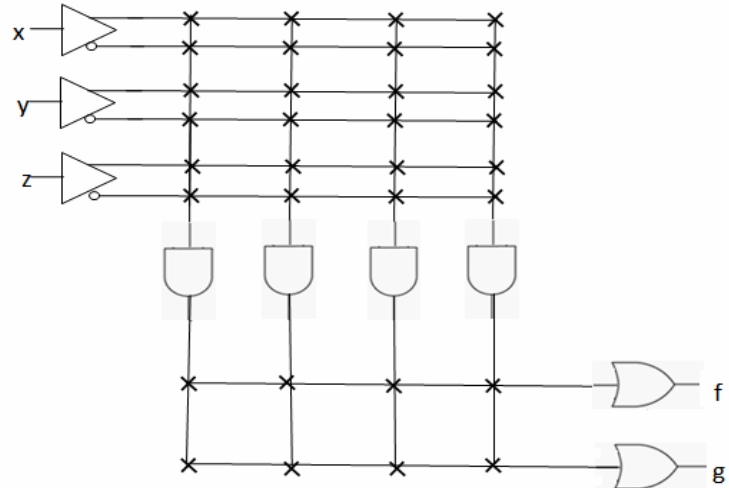
5. Design a circuit with 3 inputs  $x, y, z$  and 2 outputs  $f, g$  where  $f=x+y$ ,  $g=xy'+x'y+z$ . Use the  $3 \times 4 \times 2$  PLA chip shown immediately below. Recall that both the AND-array and OR-array are fuse programmable in PLAs. Show your answer in the answer box by putting x's in the locations where the fuses are not blown. This PLA chip does not include XOR gates at the outputs.

There are only 4 AND-gates so we are limited to 4 product terms. Must share terms between  $f$  and  $g$ .

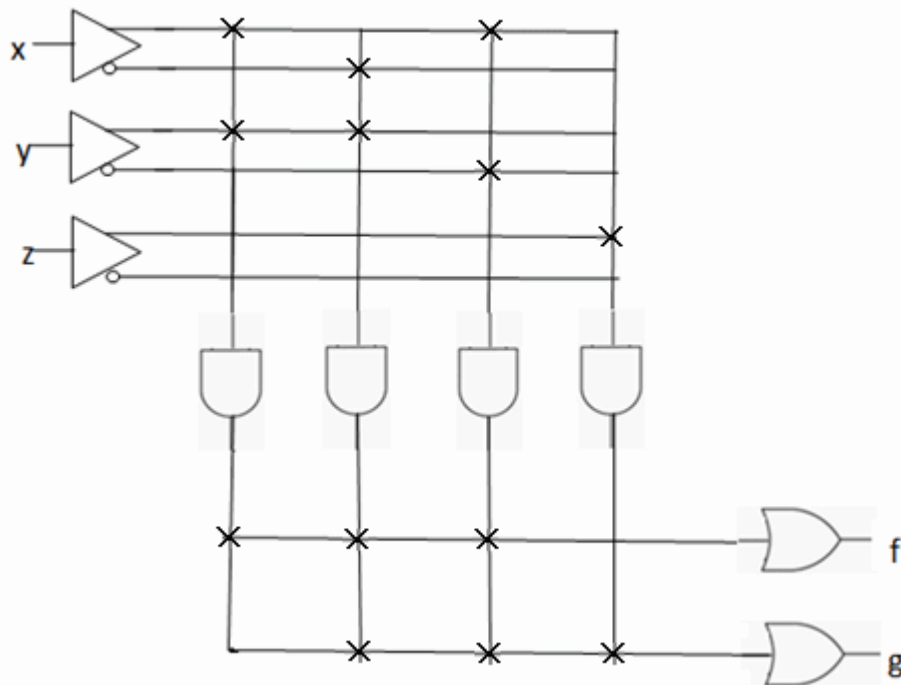
$$f = x(y+y') + y(x+x') = xy + xy' + x'y$$

$$g = xy' + x'y + z$$

This way, only 4 product terms needed.



5.



6. Write a MIPS assembler program that will store the positive numbers 0x00...0xA0 in successive half words starting at location 0x10010000, and store their sum in register \$t0.

6.

.text			
main:	li	\$t0 0	# initialize (optional)
	li	\$t1 0	# initialize (optional)
	li	\$t2, 0x10010000	# initialize address register
	li	\$t3, 0xA1	# initialize value register
loop:	sh	\$t1, (\$t2)	# store next halfword
	add	\$t0, \$t0, \$t1	# increment sum
	addi	\$t2, \$t2, 2	# increment address by 2 bytes
	addi	\$t1, \$t1, 1	# set next value to be stored
	bne	\$t1, \$t3, loop	# check if done yet
#end			(optional)

7. Design a synchronous mod-4 down-counter with counting sequence shown in the table to the left. Use 2 positive edge-triggered D flip-flops. The function table for this type of flip-flop is shown in the other table. Complete the circuit diagram begun in the answer box.

$Q_0$	$Q_1$
1	1
0	1
1	0
0	0

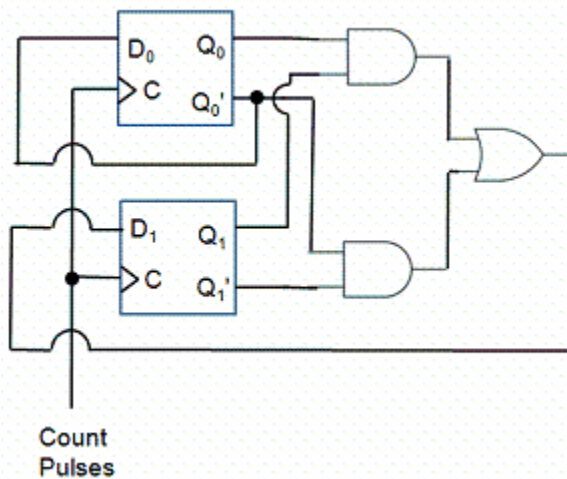
Inputs		Outputs	
D	C	$Q^+$	$Q'^+$
0	↑	0	1
1	↑	1	0
X	0	Q	Q'

Excitation table:

Present State		Next State		F-F Inputs	
$Q_0$	$Q_1$	$Q_0^+$	$Q_1^+$	$D_0$	$D_1$
1	1	0	1	0	1
0	1	1	0	1	0
1	0	0	0	0	0
0	0	1	1	1	1

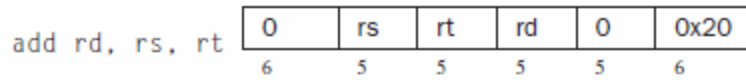
Looking at cols 1-2 vs 5, need  $D_0=Q_0'$ . From 1-2 vs 6,  $D_1=Q_0Q_1+Q_0'Q_1'$ .

7.

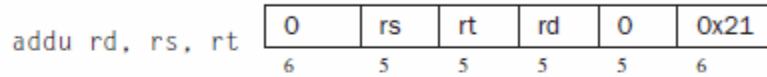


## Reference Sheet

### Addition (with overflow)

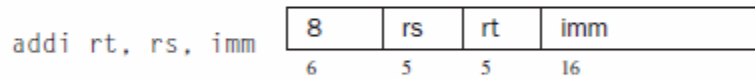


### Addition (without overflow)



Put the sum of registers *rs* and *rt* into register *rd*.

### Addition immediate (with overflow)

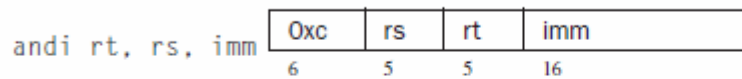


### Addition immediate (without overflow)



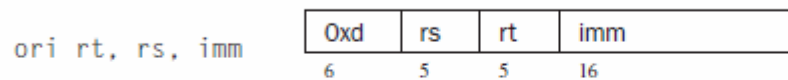
Put the sum of register *rs* and the sign-extended immediate into register *rt*.

### AND immediate

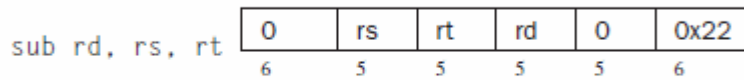
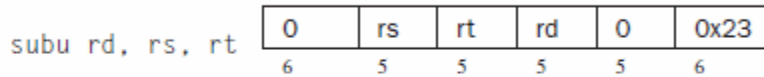


Put the logical AND of register *rs* and the zero-extended immediate into register *rt*.

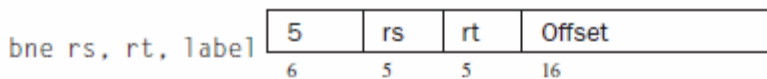
### OR immediate



Put the logical OR of register *rs* and the zero-extended immediate into register *rt*.

**Subtract (with overflow)****Subtract (without overflow)**

Put the difference of registers *rs* and *rt* into register *rd*.

**Branch on not equal**

Conditionally branch the number of instructions specified by the offset if register *rs* is not equal to *rt*.

**Branch on equal zero**

beqz *rsrc*, label *pseudoinstruction*

Conditionally branch to the instruction at the label if *rsrc* equals 0.

**Branch on greater than equal**

bge *rsrc1*, *rsrc2*, label *pseudoinstruction*

**Branch on greater than equal unsigned**

bgeu *rsrc1*, *rsrc2*, label *pseudoinstruction*

Conditionally branch to the instruction at the label if register *rsrc1* is greater than or equal to *rsrc2*.

**Jump**

Unconditionally jump to the instruction at target.

### Load immediate

`li rdest, imm` *pseudoinstruction*

Move the immediate `imm` into register `rdest`.

### Load word

`lw rt, address`

0x23	rs	rt	Offset
6	5	5	16

Load the 32-bit quantity (word) at `address` into register `rt`.

### Load address

`la rdest, address` *pseudoinstruction*

Load computed `address`—not the contents of the location—into register `rdest`.

### Load halfword

`lh rt, address`

0x21	rs	rt	Offset
6	5	5	16

### Load unsigned halfword

`lhu rt, address`

0x25	rs	rt	Offset
6	5	5	16

Load the 16-bit quantity (halfword) at `address` into register `rt`. The halfword is sign-extended by `lh`, but not by `lhu`.

### Store halfword

`sh rt, address`

0x29	rs	rt	Offset
6	5	5	16

Store the low halfword from register `rt` at `address`.

### Store word

`sw rt, address`

0x2b	rs	rt	Offset
6	5	5	16

Store the word from register `rt` at `address`.