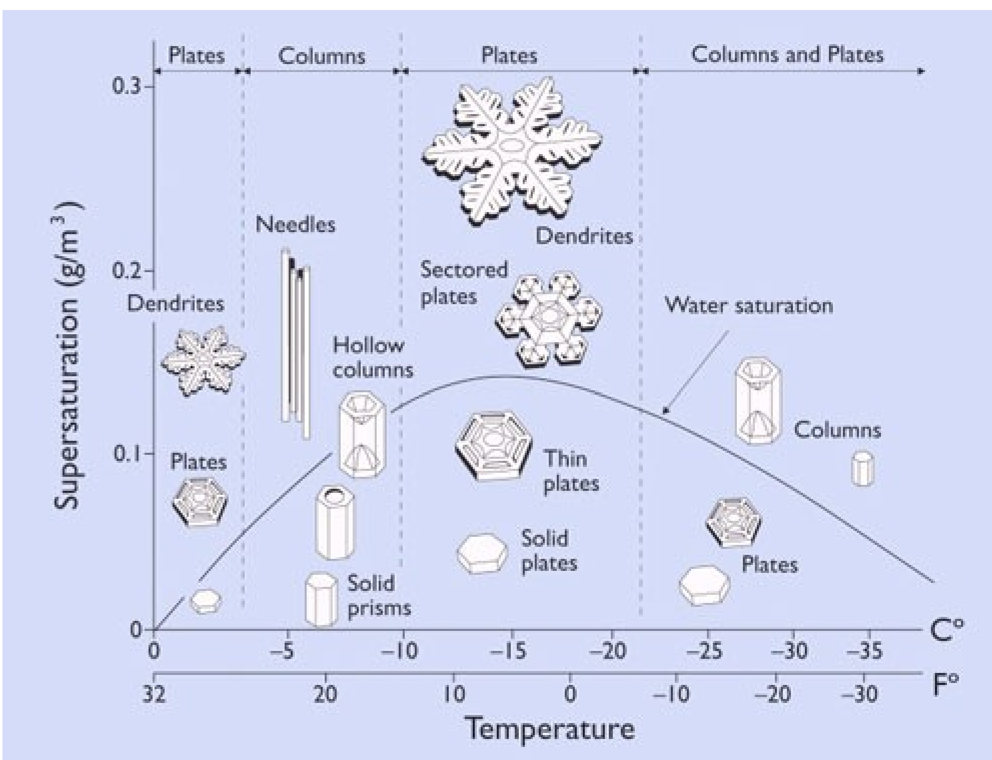
ECE 2030 2:00pm | Computer Engineering | Fall 2010

5 problems, 9 pages | Final Exam | 17 December 2010

Instructions: This is a closed book, closed note exam. Calculators are not permitted. If you have a question, raise your hand and I will come to you. Please work the exam in pencil and do not separate the pages of the exam. For maximum credit, show your work.
Good Luck!

Your Name (***please print***) ____

1	2	3	4	5		total
28	32	24	28	32		144



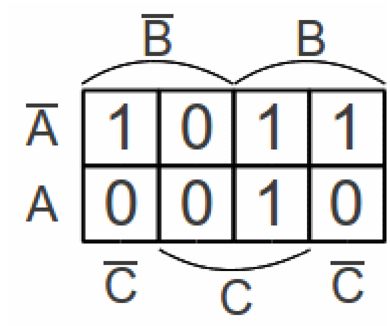
http://blogs.static.mentalfloss.com/blogs/archives/20735.html

Problem 1 (3 parts, 28 points) **Instruction Formats, Etc.**

Part A (8 points) Suppose a datapath has three operand busses (two source, one destination), 244 different instruction types, and 128 registers where each register is 32 bits wide. Immediate operands can be in the range of ±8K. Label the fields of an I-type instruction format and indicate the maximum number of bits needed for each field.

Label:	**Label:**	**Label:**	**Label:**
# bits:	**# bits:**	**# bits:**	**# bits:**

Part B (8 points) Derive the **simplified POS expression** from the following Karnaugh map.



Simplified POS expression: ______________________

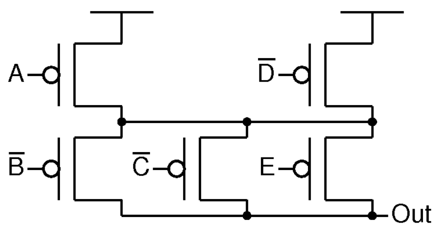
Part C (12 points) For each problem below, compute the operations using the rules of arithmetic, and indicate whether an overflow occurs assuming all numbers are expressed using a **four bit unsigned** and **four bit two's complement** representations.

	1010 + 110	101 + 100	1011 - 1110	1010 - 101
result				
unsigned error?				
signed error?				

Problem 2 (4 parts, 32 points) Dueling Designs

Complete each design below. Be sure to label all signals.

Part A: Complete the following CMOS design. Also express its behavior.



Out = ______________________________

Part B: Implement the following expression using NOR gates. Use proper mixed logic design. Determine # of switches needed.

$$Out = (A+B)\cdot\overline{C}\cdot\overline{\overline{D}+E}$$

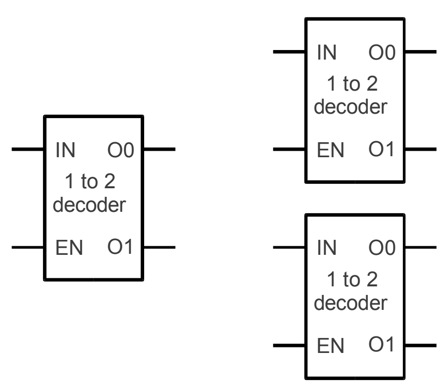
switches = ______________________________

Part C: Complete the truth table for even parity. Then implement the behavior using **only one 2 to 4 decoder and one OR gate**. *Label all inputs and outputs of the decoder*.

A	B	$\overline{A \oplus B}$
0	0	
1	0	
0	1	
1	1	

Part D: Complete the behavior table for a 2 to 4 decoder. Then implement it using three 1 to 2 decoders.

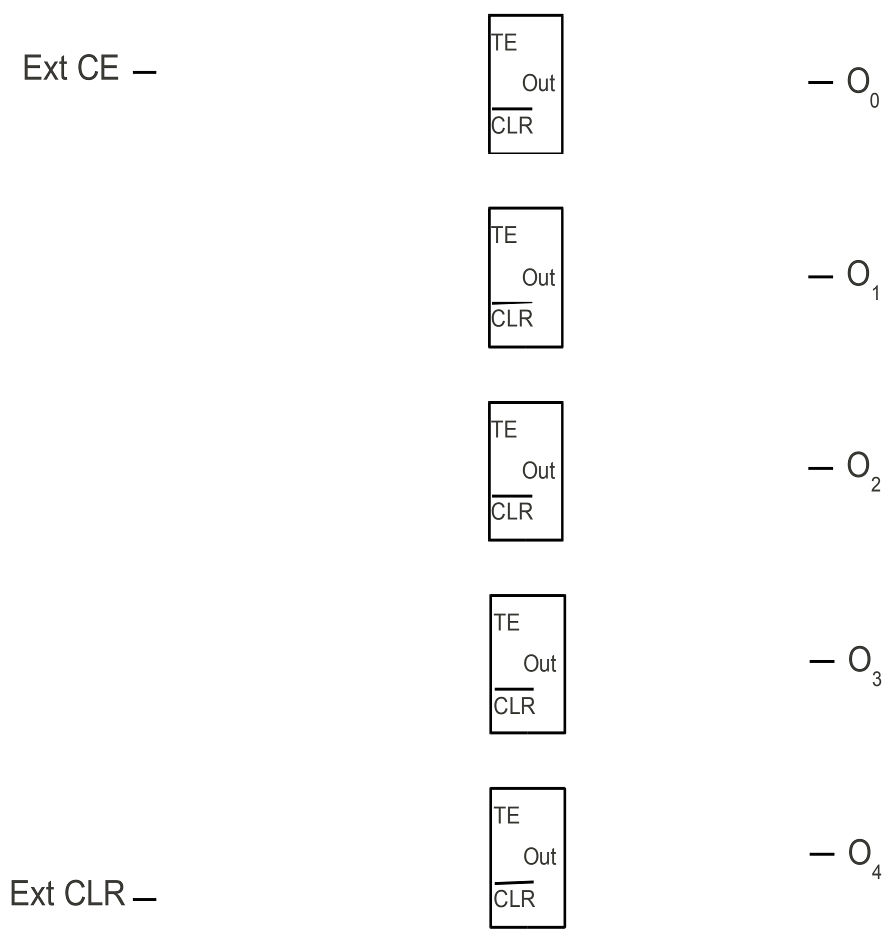
IN_1	IN_0	En	O_0	O_1	O_2	O_3



Problem 3 (3 parts, 24 points) Counters

Part A (7 points) Implement a toggle cell using ***only transparent latches and basic gates (XOR, AND, OR, NAND, NOR, NOT)***. Use an icon for the transparent latches. Label the inputs **TE**, $\overline{CLR}$, **Φ_1**, **Φ_2** and the output **Out**.

Part B (8 points) Now combine these toggle cells to build a **divide by 24** counter. Your counter should have an external clear, external count enable, and five count outputs O_4, O_3, O_2, O_1, O_0. Use any basic gates (AND, OR, NAND, NOR, & NOT) you require. Assume clock inputs to the toggle cells are already connected. *Your design should support multi-digit systems*.



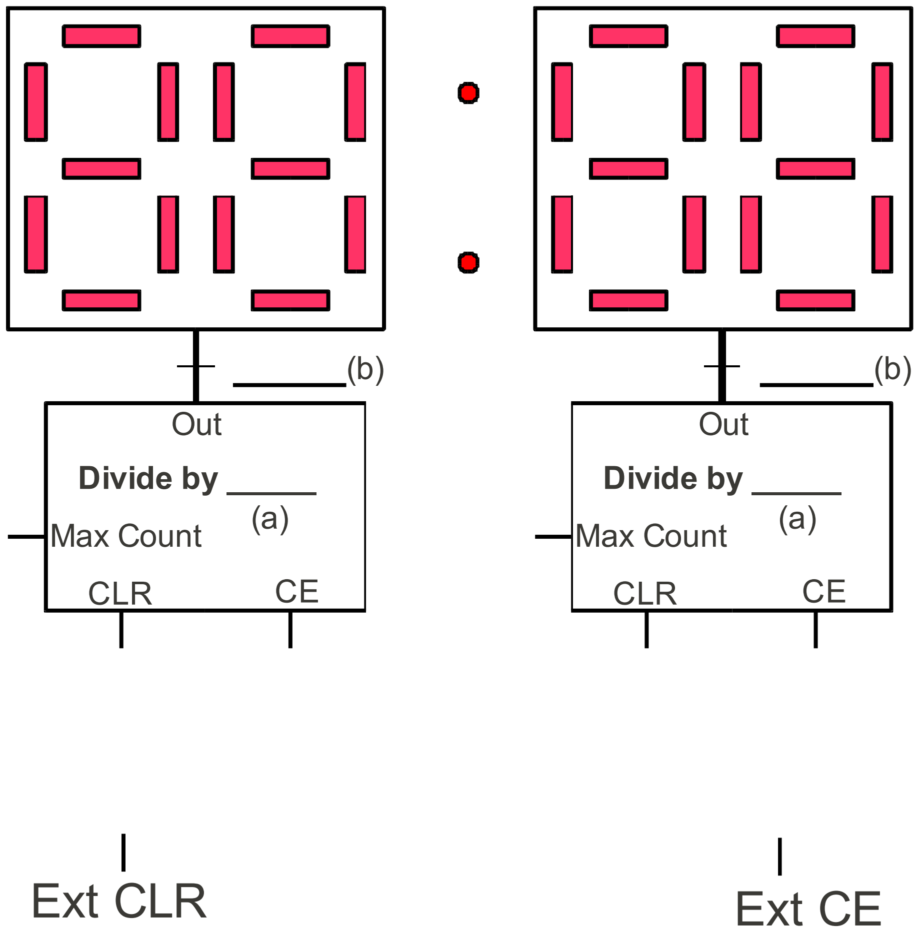
Part C (9 points) Build a military timer (HH:MM) which displays hours (0...23) on the left and minutes (0...59) on the right as follows. In the diagram below:

a) Fill in the label "Divide by ___" on each counter.

b) Label the number of output wires coming from each counter to its attached display.

c) Draw the appropriate wiring connections to allow this military timer to correctly respond to external clear (Ext CLR) and count enable (Ext CE) signals, and to correctly increment the hour count when the maximum number of minutes have passed while the clock is still running.

Use any basic gates you require. Assume clock inputs are already connected.



Problem 4 (3 parts, 28 points) **Storage**

Part A (12 points) Consider a **256 Mbit** DRAM chip organized as **8 million addresses** of **32-bit words**. Assume both the DRAM cell and the DRAM chip are square. The column number and offset concatenate to form the memory address. Using the organization approach discussed in class, answer the following questions about the chip. *Express all answers in decimal (not powers of two)*.

total number of bits in address ____________________

number of columns ____________________

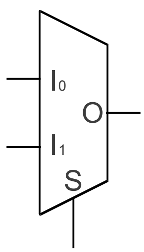
column decoder required (*n* to *m*) ____________________

number of words per column ____________________

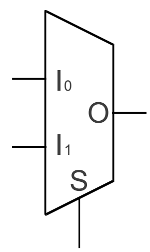
type of mux required (*n* to *m*) ____________________

number of address lines in column offset ____________________

Part B (10 points) Implement a ten transistor transparent latch (left) and a register with write enable (right) using the 2 to 1 mux plus other devices. Label all inputs and outputs.

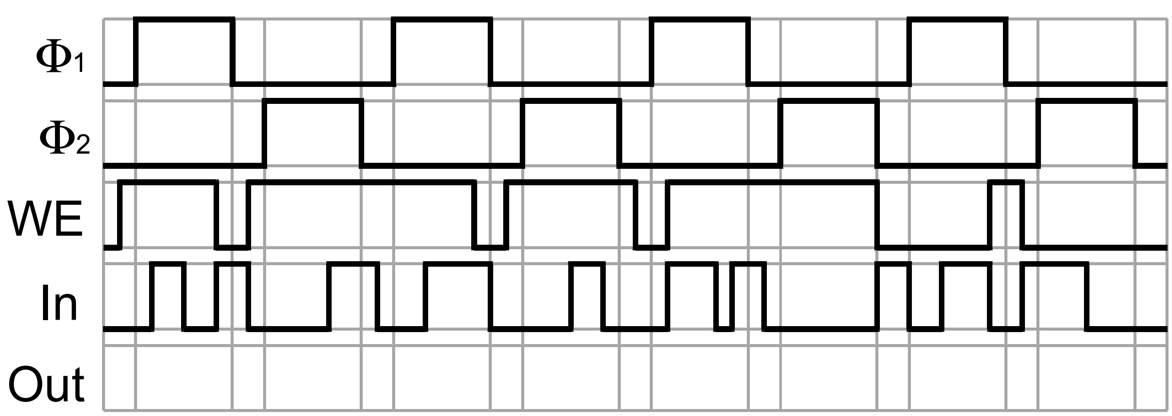


10T Transparent Latch



register with write enable

Part C (6 points) Assume the following signals are applied to a register with write enable. Draw the output signal **Out**. Draw a vertical line where **In** is sampled. *Assume **Out** is initially zero.*



Problem 5 (5 parts, 32 points) **Assembly Language Programming**

Part A (14 points) Write a MIPS subroutine `SumMags` that reads in a vector of integers and sums up the magnitude (absolute value) of each element, placing the sum of magnitudes in register \$3. Assume the length of the vector (# of integer elements) is given in register \$2 and is > 0, and the base address of the vector is in register \$1. Your code calls the subroutine `Abs`, which computes the absolute value of an integer x given in register \$4; it returns $|x|$ in register \$4. Follow the steps outlined in the comments in the rightmost column below. **You may modify only registers \$1 through \$4**.

label	instruction	comment
SumMags:		# initialize running sum (\$3 = 0)
Loop:		# load current vector element x into \$4
B:	**[leave blank for part A]**	# code to be written in part B to # preserve registers on stack
	jal Abs	# call Abs (\$4 = |x|)
C:	**[leave blank for part A]**	# code to be written in part C to # restore registers on stack
		# add |x| to running sum
		# increment vector pointer to next element
		# decrement number of elements by 1
		# if number of elements ≠ 0, loop back
		# return to caller

Part B (5 points) To ensure that `SumMags` can be properly called by another subroutine and that `SumMags` can call `Abs` without losing any of the intermediate values it computes, you must add code before and after the "`jal Abs`" instruction. Write MIPS code to preserve registers before the `jal` by pushing them on the stack. Assume `Abs` can modify *any* registers, not just \$4.

label	instruction	comment
B:		
	jal Abs	# call Abs (\$4 = |x|)

Part C (5 points) Write MIPS code to restore registers after the `jal` by popping them from the stack. Assume `Abs` can modify *any* registers, not just $4.

label	instruction	comment
	jal Abs	# call Abs ($4 = |x|)
C:		

Part D (4 points) Write the MIPS instruction that is equivalent to the following microinstruction.

#	X	Y	Z	rwe	im en	im va	au en	s/$\overline{a}$	lu en	lf	su en	st	ld en	st en	r/$\overline{w}$	msel	description
6	2	8	7	1	0	x	0	x	1	8	0	x	0	0	x	0	

Equivalent MIPS Instruction: ______________________________

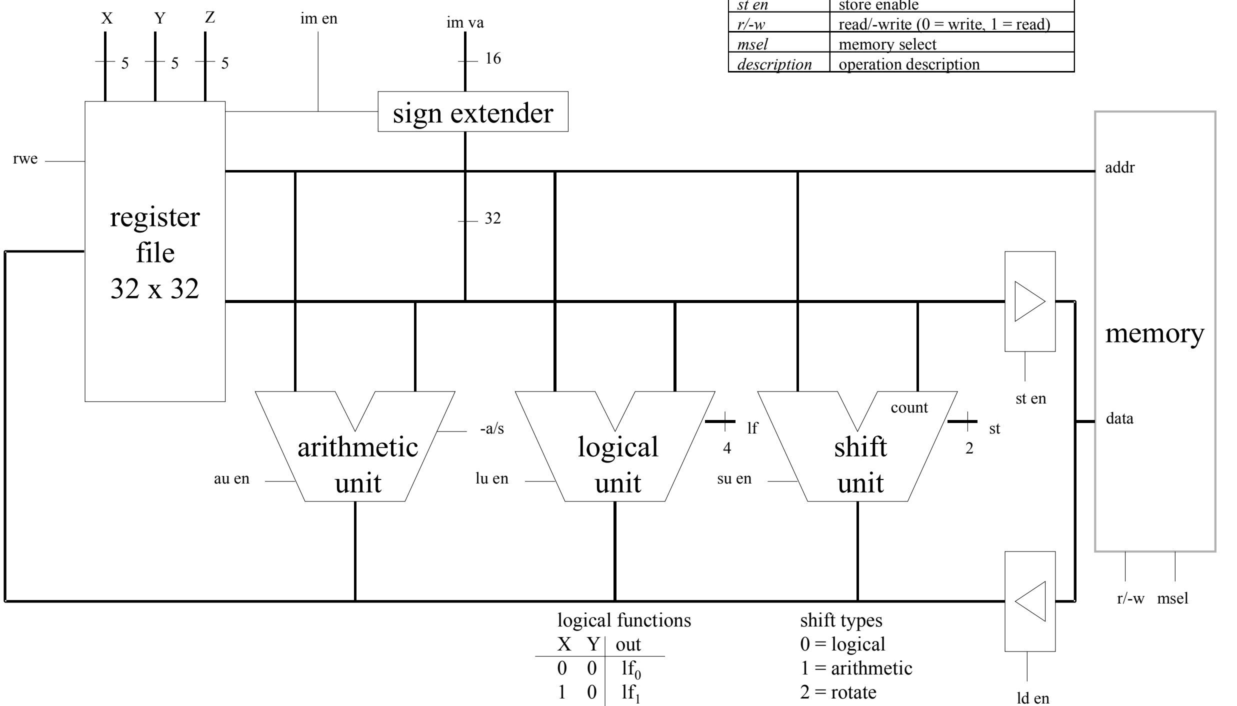
Part E (4 points) Write the MIPS instruction that is equivalent to the following microinstruction.

#	X	Y	Z	rwe	im en	im va	au en	s/$\overline{a}$	lu en	lf	su en	st	ld en	st en	r/$\overline{w}$	msel	description
7	3	x	6	1	1	FFFA	0	x	0	x	1	0	0	0	x	0	

Equivalent MIPS Instruction: ______________________________

cycle	cycle number
X	register driven onto X bus
Y	register driven onto Y bus
Z	register written from Z bus
rwe	register write enable
im en	immediate enable on Y bus
im va	immediate value

au en	arithmetic unit enable
-a/s	-add / sub (0 = add, 1 = subtract)
lu en	logical unit enable
lf	logical function
su en	shift unit enable
st	shift type
ld en	load enable
st en	store enable
r/-w	read/-write (0 = write, 1 = read)
msel	memory select
description	operation description



logical functions

X	Y	out
0	0	lf_0
1	0	lf_1
0	1	lf_2
1	1	lf_3

shift types
0 = logical
1 = arithmetic
2 = rotate
+ count shifts right
- count shifts left

MIPS Instruction Set

instruction	example	meaning
add	`add $1,$2,$3`	`$1 = $2 + $3`
subtract	`sub $1,$2,$3`	`$1 = $2 - $3`
add immediate	`addi $1,$2,100`	`$1 = $2 + 100`
multiply	`mul $1,$2,$3`	`$1 = $2 * $3`
divide	`div $1,$2,$3`	`$1 = $2 / $3`
and	`and $1,$2,$3`	`$1 = $2 & $3`
or	`or $1,$2,$3`	`$1 = $2 | $3`
xor	`xor $1,$2,$3`	`$1 = $2 xor $3`
and immediate	`andi $1,$2,100`	`$1 = $2 & 100`
or immediate	`ori $1,$2,100`	`$1 = $2 | 100`
xor immediate	`xori $1,$2,100`	`$1 = $2 xor 100`
shift left logical	`sll $1,$2,5`	`$1 = $2 << 5 (logical)`
shift right logical	`srl $1,$2,5`	`$1 = $2 >> 5 (logical)`
shift left arithmetic	`sla $1,$2,5`	`$1 = $2 << 5 (arithmetic)`
shift right arithmetic	`sra $1,$2,5`	`$1 = $2 >> 5 (arithmetic)`
load word	`lw $1, ($2)`	`$1 = memory [$2]`
store word	`sw $1, ($2)`	`memory [$2] = $1`
load upper immediate	`lui $1,100`	`$1 = 100 x` 2^{16}
branch if equal	`beq $1,$2,100`	`if ($1 = $2), PC = PC + 4 + (100*4)`
branch if not equal	`bne $1,$2,100`	`if ($1 ≠ $2), PC = PC + 4 + (100*4)`
set if less than	`slt $1, $2, $3`	`if ($2 < $3), $1 = 1 else $1 = 0`
set if less than immediate	`slti $1, $2, 100`	`if ($2 < 100), $1 = 1 else $1 = 0`
jump	`j 10000`	`PC = 10000`
jump register	`jr $31`	`PC = $31`
jump and link	`jal 10000`	`$31 = PC + 4; PC = 10000`