Practice Questions (and Answers) for Final Exam
CS 64, Winter 2019, Matni

IMPORTANT NOTE: These questions are NOT representative of EVERYTHING you need to study for the midterm exam! You should also review your lab assignments questions and also all the examples and demos done in class.

1. Binary-to-decimal/hexadecimal conversion
    a. Convert **1001 0010 1100 0011** to 4-digit hexadecimal
    b. Convert the *signed* binary value **1001 1111** to decimal

2. Add the 2 following 8-bit numbers: **0110 0010** and **0011 0100** and indicate the status of the carry and overflow bits at the end of the addition. Interpret your findings.

3. Name one reason why **`li`** is a MIPS *pseudoinstruction*.

4. Translate this MIPS assembly code into C/C++ code.

```
.data
talk: .asciiz "blabla"
cs: .word 3
.text
main:
    li $t0, 5
    la $t1, cs
    lw $t2, 0($t1)
    blt $t0, $t2, gothere
    li $v0, 4
    la $a0, talk
    syscall
    j end
gothere:
    li $v0, 4
    la $a0, talk
    syscall
    syscall
end:
    li $v0, 10
    syscall
```

5. Write the following MIPS instructions in machine-language hexadecimals (show all work): **`addiu $t0, $s0, 17`** and **`sub $v0, $s4, $t5`**

6. Given a MIPS machine language instruction of **0x02088024**, and being told that it is an R-type, what is the assembly instruction?

7. What will the final value in register $s0 in this code be?

```
li $s0, 20
sll $s0, $s0, 2
add $s0, $s0, $s0
sra $s0, $s0, 4
```

8. Consider the C/C++ code below:

```
// arr is a globally accessible array of ints
// s0 already holds a value of type unsigned int
unsigned int s1 = arr[s0];
unsigned int s2 = arr[s0 - 1];
unsigned int s3 = arr[s0 + 1];
```

Using **no more than six instructions**, implement the above code snippet in MIPS. You **don't** have to follow the MIPS Calling Convention.

9. Consider the C/C++ code below.

```
int sum( int arr[], int size )
{
   if ( size == 0 )
      return 0;
   else
      return sum( arr, size - 1 ) + arr[size – 1];
}
```

a. Knowing that you **have to follow the MIPS Calling Convention**, which variables should be preserved either directly (via the stack) or indirectly (in an S-register) in order to maintain the intended program behavior?

b. Implement the previously shown C/C++ code using MIPS assembly, taking care to preserve the values you identified previously. Ignore the **.data** part and just focus on the **.text** part of the program.

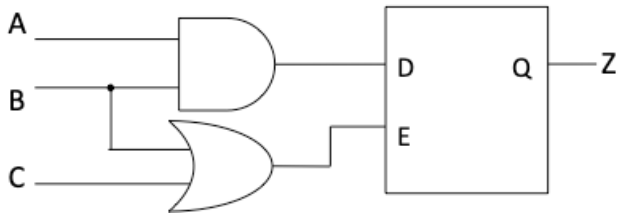
10. Show how a NOR function can be used as an AND function.

11. Simplify this expression using Boolean algebra: **F = NOT ((A NOR B) . (C + A.B))** and draw the resulting circuit.
12. Consider the following truth table, which includes don't cares:

A	B	C	D	R
0	0	0	0	1
0	0	0	1	0
0	0	1	0	X
0	0	1	1	0
0	1	0	0	X
0	1	0	1	1
0	1	1	0	X
0	1	1	1	X
1	0	0	0	X
1	0	0	1	0
1	0	1	0	1
1	0	1	1	X
1	1	0	0	1
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Simplify the output function **R** using a Karnaugh Map, and show the resulting sum-of-products representation. Show the map, along with the boxes you chose. For full credit, both the number of ORs (+) and the sizes of the products must be minimal.

13. Consider this circuit:



What does the output **Z** do with the following values of A, B, and C (suppose that these values happen in sequence: that is one after the other as shown in the table). Also explain why:

A	B	C	Z	Reason
0	0	1		
1	1	0		
0	0	0		
1	0	0		
1	0	1		

14. Consider a device that consists of three buttons labeled "UP" and "RESET", along with a light. The device internally counts the number of times "UP" is pressed, and when it is pressed two times, the device causes the light to illuminate. Additional presses of "UP" do nothing. Pressing "RESET" at any point will reset the internal counter back to zero, and will cause the light to go out. (Note that the light may have already been off, as when the user presses "UP" once followed by "RESET".)

    For this question, you will implement this device as a finite state machine. The machine has the following two external inputs:

    R: set to 1 whenever "RESET" is pressed

    U: set to 1 whenever "UP" is pressed

    The machine also has one external output:

    L: set to 1 whenever the light should be illuminated

    If both "RESET" and "UP" are pressed at the same time, then the behavior should be as if only "RESET" was pressed. Basically, if R is set, no matter what state you are in, you go back to the initial state.

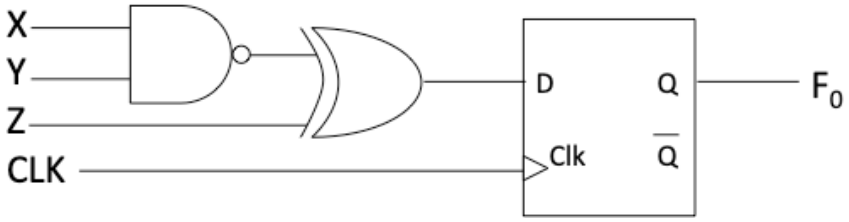
    a) Draw the finite state machine diagram corresponding to this task. All transitions should be drawn as products of R and U. For example, if a particular transition should be taken only if R = 1 and U = 0, then this should be drawn as R.$\overline{U}$ or R.!U.

    b) Using the "regular" method, how many D-FFs are necessary to implement this state machine? Draw the truth table for all 3 states, showing current state bits, input bits R and U, next state bits, and output bit L.

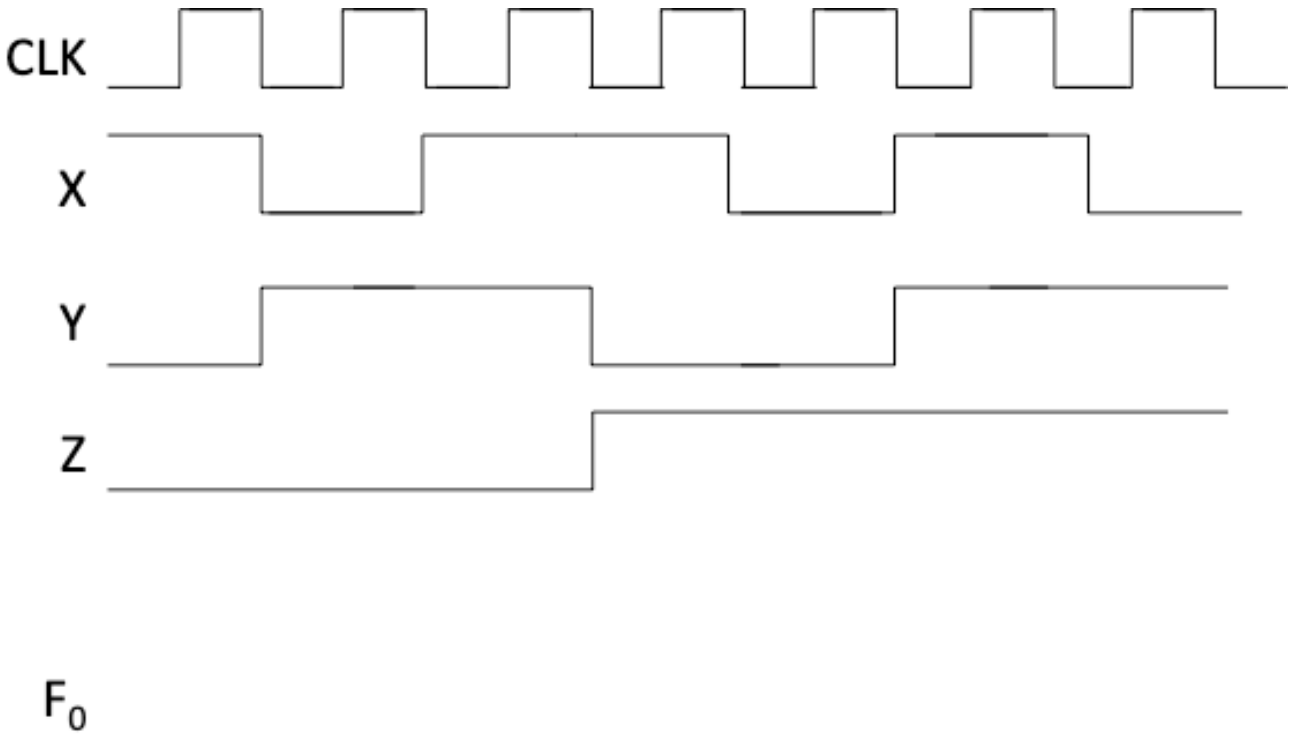
    c) Using K-Maps, write the optimal functions for the next state bits and the output.

    d) If we use the "one-hot method" instead, how many D-FFs are necessary to implement this state machine? Write all the logic formulas that describe all the states, as well as the output L.

15. Consider the following digital circuit:



a) Write the expression for the next-state bit F_0 as a sum-of-product.
b) Assuming that F_0 is initially 0, complete the timing diagram for F_0 based on your answer above. Make your drawing as accurate as you can. *Hint*: Draw the waveform to the input of the DFF as well as any other intermediate nodes/places in the circuit.



****** TO GET THE MOST OUT OF THIS REVIEW, DO THE QUESTIONS FIRST BEFORE LOOKING AT THE ANSWERS!******

ANSWERS TO THE REVIEW QUESTIONS FOR FINAL EXAM

1.

a. 1001 0010 1100 0011 /bin = 0x92C3

b. 1001 1111 /bin → 0110 0000 + 1 = 0110 0001 = $-(2^6 + 2^5 + 1) = -97$

2. 0110 0010
+ 0011 0100
1001 0110
The carry out bit = 0, the overflow bit is 1.
So, if these were 2 unsigned numbers, there would be no carry out, but if these were 2 signed numbers, then we'd have overflow.

3. `li` takes as second argument a 32-bit signed number. MIPS instructions themselves are 32-bit long, so loading this number into a register should actually be done in pieces: first load the upper 16-bits of the number, then load the lower 16-bits. Therefore the instruction `li` is really a macro for (at least) 2 regular instructions – i.e. it's pseudocode.

4. In C/C++:

```
char talk[] = "blabla";
int cs = 6;
int t0 = 5;
if (t0 >= cs) { printf(talk); }
else { printf(talk); printf(talk); }
```

5. **`addiu $t0, $s0, 17`** = 0x26080011
`sub $v0, $s4, $t5` = 0x028D1022

6. **0x02088024** = and $s0, $s0, $t0

7. $s0 = 20 → This is in decimal. So it's 0000 … 0001 0100 (in 32-bit binary)
sll $s0, $s0, 2 → $s0 becomes 0000 … 0101 0000
add $s0, $s0, $s0 → $s0 becomes 0000 … 1010 0000
sra $s0, $s0, 4 → $s0 becomes 0000 … 0000 1010 = **10 /dec**

8. In 6 or under instructions:

```
la $t0, arr
sll $s0, $s0, 2
addu $t0, $t0, $s0
lw $s1, 0($t0)
lw $s2, -4($t0)
lw $s3, 4($t0)
```

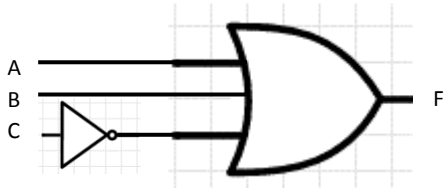
9. We assume **arr is in $a0 and size is in $a1**.

```
int sum( int arr[], int size ) {
  if ( size == 0 )
    return 0 ;
  else
    return sum( arr, size - 1 ) + arr[size-1]; }
```

```
.text
sum:
    addiu $sp, $sp, -12 # PUSH
    sw $ra, 8($sp)
    sw $s1, 4($sp)
    sw $s0, 0($sp)

    li $v0, 0
    beq $a1, $zero, return    # is size !=0?

    addi $a1, $a1, -1   # size is now: size - 1
    move $s0, $a0       # preserve &a0
    move $s1, $a1       # preserve a1 (size)

    jal sum             # recursive call

    sll $s1, $s1, 2     # multiply size by 4
    add $s0, $s0, $s1   # s0 is now the address of a[size-1]

    lw $t0, 0($s0)      # Get that array element
    add $v0, $v0, $t0   # add it to $v0

return:
    lw $ra, 8($sp)      # POP
    lw $s1, 4($sp)
    lw $s0, 0($sp)
    addiu $sp, $sp, 12
    jr $ra

main:
    la $a0, arr     # a0 = &a[]
    li $a1, 4       # a1 = size

    jal sum

exit:
    li $v0, 10
    syscall
```

10. Taking advantage of DeMorgan's theorm, you will not that if the inputs to the NOR are inverted, you get: F = NOT($\overline{A}$ + $\overline{B}$) = A.B

11. F = NOT ((A NOR B) . (C + A.B))
= NOT (($\overline{A}$. $\overline{B}$) . (C + A.B))
= NOT ($\overline{A}$. $\overline{B}$. C + $\overline{A}$. $\overline{B}$. A . B)
= NOT ($\overline{A}$. $\overline{B}$. C)
= A + B + $\overline{C}$



12. K-Map:

CD \ AB	**00**	**01**	**11**	**10**
00	1	X	1	X
01		1	X	
11		X	X	X
10	X	X	X	1

Since X's can be either 0 or 1, to maximize the size of our groupings and minimize the number of our groupings, we can transform the above to the following with 2 major groupings:

CD \ AB	**00**	**01**	**11**	**10**
00	1	1	1	1
01		1	1	
11		1	1	0
10	1	1	1	1

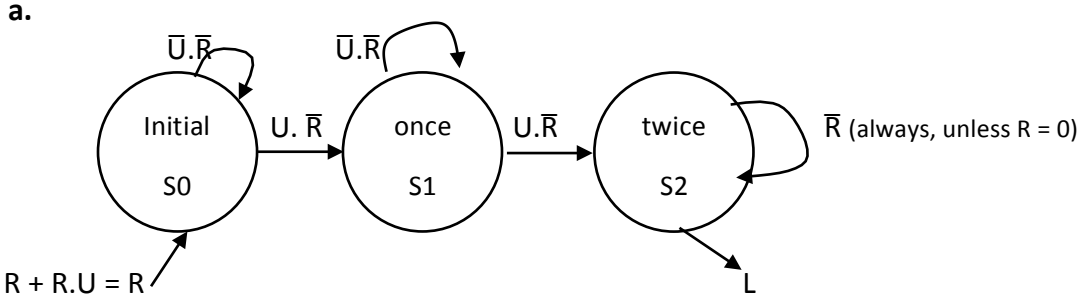
This gives us the formula: **F = $\overline{D}$ + B**
NOTE: I purposely made one of the Xs into a 0, so that I could minimize my groupings.

13.

A	**B**	**C**	**Z**	**Reason**
0	0	1	0	D-latch is enabled (E = 1). D = 0, so Q = 0.
1	1	0	1	D-latch is enabled (E = 1). D = 1, so Q = 1.
0	0	0	1	D-latch is not enabled (E = 0). So Q = Q_{old} = 1.
1	0	0	1	D-latch is not enabled (E = 0). So Q = Q_{old} = 1.
1	0	1	0	D-latch is enabled (E = 1). D = 0, so Q = 0.

14. State diagram:

a.



b. We'd need 2 bits (so 2 DFFs): B1 and B0.
S0 would be B1B0 = 00, S2 would be 01, S3 would be 10. The combination of B1B0 = 11 is undefined.

B1	**B0**	**U**	**R**	**B1***	**B0***	**L**
0	0	0	0	0	0	0
0	0	1	0	0	1	0
0	1	0	0	0	1	0
0	1	1	0	1	0	0
1	0	X	0	1	0	1
X	X	X	1	0	0	0

c. K-Maps:

For B1*:

B1B0 *UR*	00	01	11	10
00				**1**
01				
11				
10		**1**		**1**

For B0*:

00	01	11	10
	1		
1			

For L:

00	01	11	10
			1
			1

B1* = !B1.B0.U.!R + B1.!B0.!R
B0* = !B1.B0.!U.!R + !B1.!B0.U.!R
L = B1.!B0.!R

d. Using the one hot method, we'd need **3 DFFs** – one for every state. The formulas for each state and the output would be:

S0* = R + S0.$\bar{U}$.$\bar{R}$
S1* = S0.U.$\bar{R}$ + S1.$\bar{U}$.$\bar{R}$
S2* = S1.U.$\bar{R}$ + S2.$\bar{R}$
L = S2

15.

a. F_0 = !(X.Y) XOR Z = (X.Y).Z + !(X.Y).!Z = XYZ + !X!Z + !Y!Z

b.