

REVIEW QUESTIONS FOR FINAL EXAM

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 \$r2, \$s4, \$t5**
6. 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
```

7. Consider the 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 C code snippet in MIPS. You don't have to follow the MIPS Calling Convention.

8. Consider the C code below.

```
int sum( int arr[], int size )
{
    if ( size == 0 )
        return 0;
    else
        return sum( 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 code using MIPS assembly, taking care to preserve the values you identified previously.

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

10. Simplify this expression using Boolean algebra: **$F = \text{NOT} ((A \text{ NOR } B) \cdot (C + A \cdot B))$** and draw the resulting circuit.

11. 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.

12. Using one or more D-Latch, one or more 2-to-1 mux, and any number of basic logic blocks (i.e. AND, OR, NOT, etc.), construct a circuit that takes 2 bits as inputs (B0, B1) and writes them to 2 registers called REG0 and REG1, respectively, but only when another input, E = 1. The registers retain their values if E = 0. The output of this circuit is the value stored in one of the registers as chosen by inputs S1 and S0. If S1=1 and S0=1, we should see the value in REG1 appear on the output (O) and if S1=0 and S0=0, we should see the value in REG0 appear at O. Any other combination of S1 and S0 should leave the output unchanged, that is the previous selection should remain the same.
13. 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.

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 $R.\bar{U}$ or $R.!U$.

14. If we use the “one-hot method”, how many D-latches are necessary to implement this state machine? Write all the logic formulas that describe all the states, as well as the output L.

ANSWERS TO THE REVIEW QUESTIONS FOR FINAL EXAM

1.
 - a. 1001 0010 1100 0011 /bin = 0x92C3
 - b. 1001 1111 /bin \rightarrow 0110 0000 + 1 = 0110 0001 = $-(2^6 + 2^5 + 1) = -97$
2.

$$\begin{array}{r} 0110\ 0010 \\ + 0011\ 0100 \\ \hline 1001\ 0110 \end{array}$$

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) { print(talk); }
else { print(talk); print(talk); }
```
5. `addiu $t0, $s0, 17` = 0x26080011
`sub $r2, $s4, $t5` = 0x028D1022
6. $\$s0 = 20$ /dec = 0000 ... 0001 0100 (in 32-bits)
`sll $s0, $s0, 2` \rightarrow $\$s0$ becomes 0000 ... 0101 0000
`add $s0, $s0, $s0` \rightarrow $\$s0$ becomes 0000 ... 1010 0000

sra \$s0, \$s0, 4 → \$s0 becomes 0000 ... 0000 1010 = 10 /dec

7. 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)
```

8. 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]; }
```

```
sum:    addi    $sp, $sp, -8      # Adjust sp so you can PUSH
        sw     $s0, 0($sp)      # Save $s0 to stack
        sw     $ra, 4($sp)      # Save return address

        bne    $a1, $zero, else # branch ! ( size == 0 )
        li     $v0, 0           # Set return value to 0
        addi    $sp, $sp, 8      # Adjust sp
        jr     $ra              # Return to caller of sum()

else:    addi    $a1, $a1, -1     # update size to (size - 1)
        jal     sum              # FIND arr[size-1]

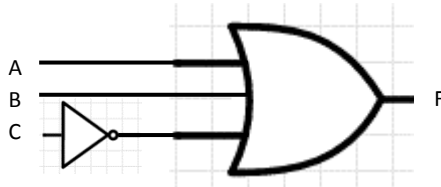
        sll     $a1, $a1, 2      # Multiply (size - 1) by 4
        add     $s0, $a0, $a1    # Compute &arr[ size - 1 ]
        lw     $t0, 0($s0)       # t2 = arr[ size - 1 ]

        add     $v0, $v0, $t0    # ret value = $v0 + arr[size - 1]

        lw     $ra, 4($sp)      # restore return address from stack
        lw     $s0, 0($sp)      # so you can POP
        addi    $sp, $sp, 8      # Adjust sp
        jr     $ra              # Return
```

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

$$\begin{aligned}
 10. F &= \text{NOT}((A \text{ NOR } B) \cdot (C + A.B)) \\
 &= \text{NOT}((\bar{A} \cdot \bar{B}) \cdot (C + A.B)) \\
 &= \text{NOT}(\bar{A} \cdot \bar{B} \cdot C + \bar{A} \cdot \bar{B} \cdot A \cdot B) \\
 &= \text{NOT}(\bar{A} \cdot \bar{B} \cdot C) \\
 &= A + B + \bar{C}
 \end{aligned}$$



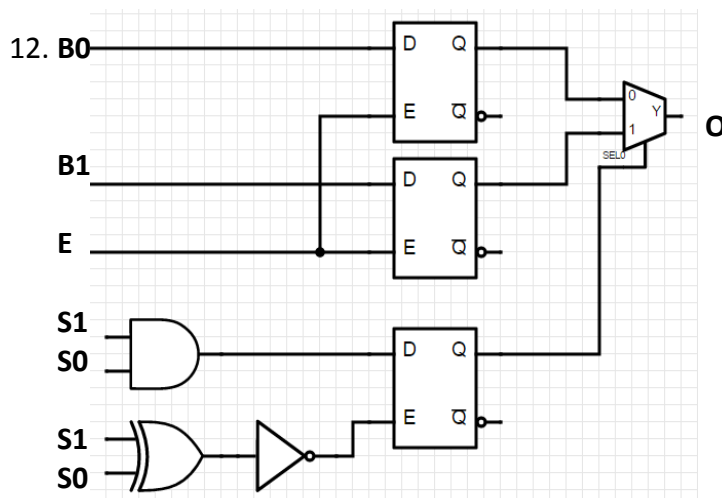
11. K-Map:

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 3 major groupings:

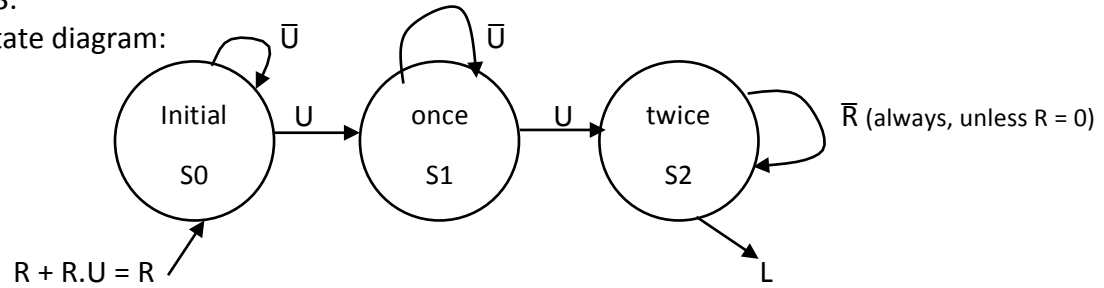
CD \ AB	00	01	11	10
00	1	1	1	1
01		1	1	
11		1	1	0
10	1	1	1	1

This gives us the formula: $F = \bar{C}.\bar{D} + C.\bar{D} + B$



13.

State diagram:



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

$$S0^* = R + S0.\bar{U}$$

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

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

$$L = S2.\bar{R} \text{ (or just } S2\text{)}$$