EE 2310 Test Review #2 – Complex Sequential Logic and Writing Simple SPIM Programs

Name: _____

1. **Write a program in the space at the right that inputs the numbers data1 and data2 into the $t0 and $t1 registers and calculates their product. The product should be stored in data3 and also output to the console using a syscall 1. End the program with a syscall 10 as usual.**

```
     .data
data1:  .word 7394
data2:  .word 1758
data3:  .word 0

     .text
main:
```

2. **Compose a short program in the space on the right that will square the number stored in data1 and check to see that it is between 255 (0xff) and 65535 (0x ffff). If between 0xff and 0xffff, it stores the number in data2, prints it out using syscall 1, and ends the program. If greater than 0xffff or less than 0xff, it stores the number in data2 and simply ends the program.**

```
     .data
data1: .word 0x3a
data2: .word 0
```

3. **<u>Loop Problem</u>: In the space to the right, write a brief loop program that will search through the ASCII sequence "Hello, world!\n" and capitalize all the lower-case letters, then print it out. Helpful hints: (1) The upper case letters go from 0x41 to 0x5a, the lower case letters from 0x61 to 0x7a. Thus, if you subtract 0x20 from the ASCII code of any lower-case letter, you capitalize it! (2) This program must be a loop, but it is easy to test for when you are done – just look for a zero (i.e., a null character). (3) Remember, you will need a pointer register that has the address of the current letter you are examining, which can be incremented by 1 each time the program starts to go through the loop again, thereby automatically pointing to the next letter.**

```
    .text
main:




    .data
str: .asciiz "Hello, world!\n"
```

4. **Construct a program in the space at the right that loads data words n-z and tests them. If the 32-bit data word is negative, output it to the console. Ignore the other values. When the program encounters the word "end," which is a zero, close it with a syscall 10 (i.e., you can look for a 0, as in an asciiz character string, to tell when to end the loop).**

   Each time you output a number, follow it with a CR/LF, to separate the numbers.

   How many numbers are output and what are they?

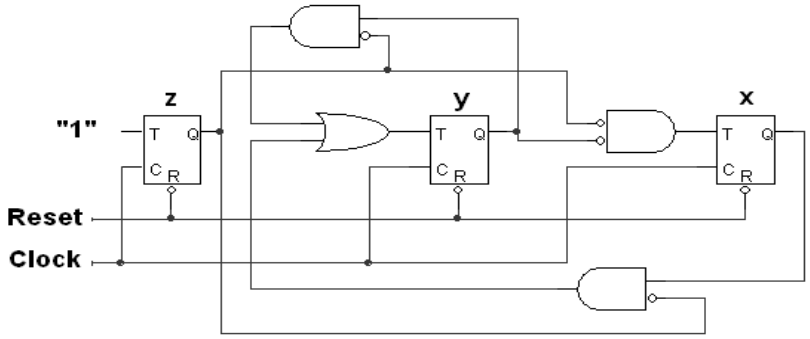
   Four numbers are output:

   0x9750494d
   0x9073696d
   0xff61726e
   0x8d495053

```
    .data
n:  .word 0x9750494d
p:  .word 0x20697320
q:  .word 0x61207573
r:  .word 0x6566756c
s:  .word 0x9073696d
t:  .word 0x756c6174
u:  .word 0x6f722066
v:  .word 0x6f72206c
w:  .word 0xff61726e
x:  .word 0x696e6720
y:  .word 0x8d495053
z:  .word 0x20617373
end: .space 4
```

Sequential Logic

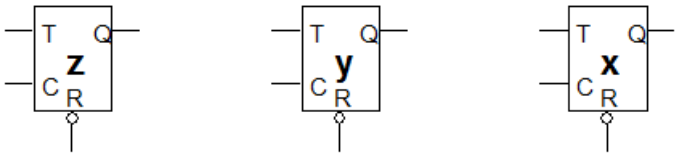
5. **The counter below counts in an unusual cycle, so simply finding "m-1" won't help you in this case. Determine the Boolean expressions for Ty and Tx from the circuit below (clearly, Tz = 1). Using those and the count chart shown, you can determine the T's at count 0, then let the clock tick. You can then determine the T's at count 1 and repeat, etc. Using this method, you can find the count cycle relatively quickly.**



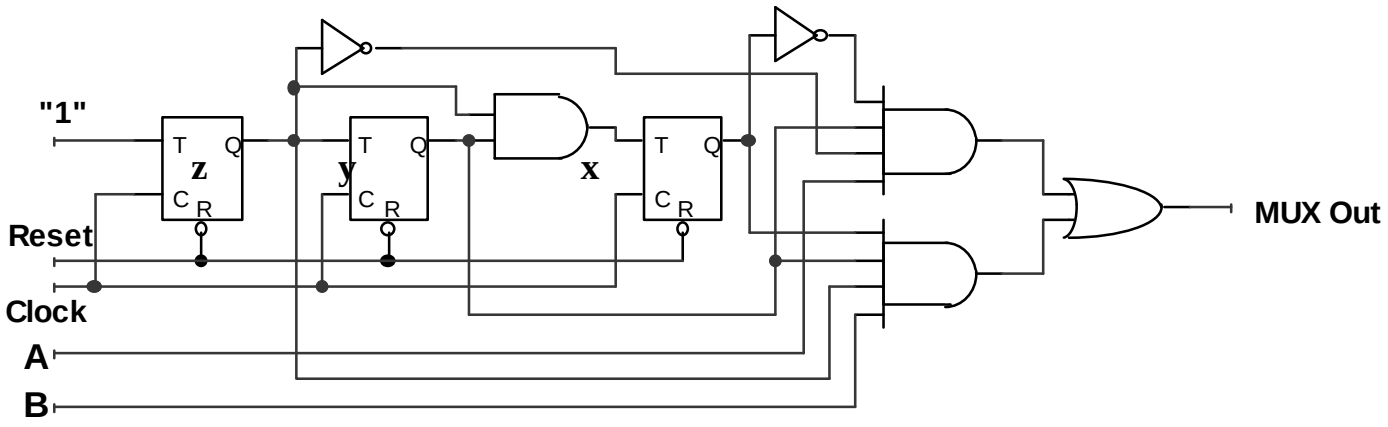
Count					
x	**y**	**z**	**Tx**	**Ty**	**Tz**
0	**0**	**0**			

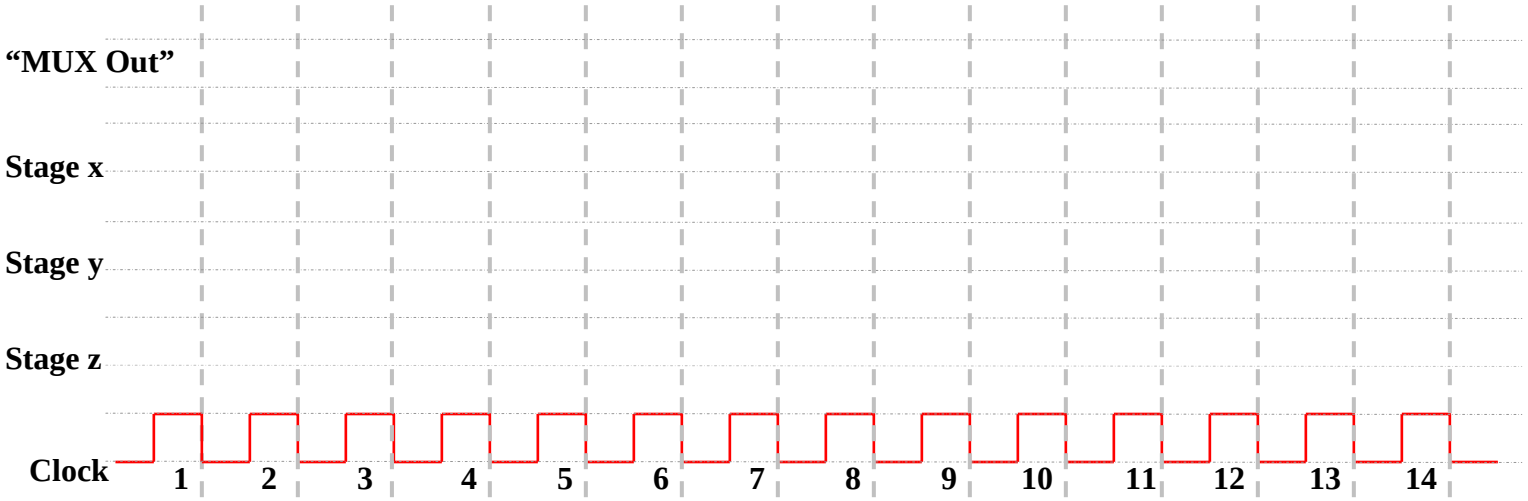
Etc.

6. **In the space below, design a modulo 7 counter using the T flip-flops shown. You may using either the K-map or short-cut approach.**



7. **Develop a timing diagram for the MUX on the chart shown below. Plot the timing of FF's x, y, and z, and also the MUX Out signal. Note that the "MUX Out" signal plots shows when inputs a and b are output.**





8. **In the shift-register ring-counter circuit below, on the master "Reset" signal, two of the ff's (1 and 3) are set to 1 rather than 0. The clock then starts, generating a signal pattern on the output f. The use is unspecified but you can assume signal f is used by another circuit for timing purposes. Plot the timing of the output f on the diagram below for the number of clock pulses shown.**