1. (CLO 5—Assy Lang.) Write a program in the space at the right that inputs the numbers data1 and data2 into the $t0 and $t1 registers. Square data1 and store in data3; square data2 and store in data4. AND data3 and data4, OR the result with data1, and XOR the result with data2. Take the 2’s complement and output the result using syscall 1.

```
.data
data1: .word 49583
data2: .word 21758
data3: .word 0
data4: .word 0

.text
main:

    li $v0,10
    syscall
```

2. (CLO 5—Assy Lang.) Compose a short program in the space on the right that will print out the smallest of the three numbers among num12, num2, and num3. Clearly, given the current data declaration, num3 should be printed out. To be sure that your program ALWAYS prints out the smallest number, exchange the values of num1-num3, so that the smallest value is in all 3 positions. If your program consistently prints out the smallest value, regardless of its location, it is correct.

```
.data
num1: .word 2146
num2: .word 3175
num3: .word 897

.text
main:
```
3. (CLO 5—Assy Lang.) **Loop Problem:** In the space to the right, write a brief loop program that will print out each of the bytes in w1, w2, w3, and w4 as ASCII characters. Note that w4 has three 0 bytes, so you can set the loop to quit when you load a null byte.

This is an extremely simple loop program if you think a bit about it before starting the program design. My program took ten instructions, including the .text declaration (but not including the data declarations).

```assembly
.data
w1: .word 0x6c6c6548
w2: .word 0x77202c6f
w3: .word 0x646c726f
w4: .word 0x00000021

.text
main:
```

4. (CLO 5—Assy Lang.) 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 positive, output it to the console. Ignore zero or negative values. Note that you have to examine 12 data words, and so you will need a counter that counts up to 12, so that you can exit the loop on count 12.

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?

```assembly
.data
n: .word 0x9750494d
p: .word 0x20697320
q: .word 0x61207573
r: .word 0x6566756c
s: .word 0x9073696d
u: .word 0x6f722066
v: .word 0x6f72206c
w: .word 0xff61726e
x: .word 0x696e6720
y: .word 0x8d495053
z: .word 0x20617373
```
5. (CLO 4—Seq. Logic) 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.

![Circuit Diagram]

<table>
<thead>
<tr>
<th>Count</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>Tx</th>
<th>Ty</th>
<th>Tz</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6. (CLO 4—Seq. Logic) In the space below, design a modulo 7 counter using the T flip-flops shown. You may use either the K-map or short-cut approach.
7. (CLO 4—Seq. Logic) 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. (CLO 4—Seq. Logic) 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.