1. (CLO 5—Assy Lang.) Construct a brief loop program to print the following ASCII bytes, ONLY, in the words aa-hh: Capital letters, commas, spaces, and exclamation points. Note that the last word is a 0, so you will not need a counter in the loop. Simply test for 0, and that will tell when the loop is completed.

What is printed out?

```assembly
.text
main:

.data
aa: .word 0x30557b6a
bb: .word 0x54396162
cc: .word 0x7a743844
dd: .word 0x67372079
ee: .word 0x71724561
ff: .word 0x32334366
gg: .word 0x3579457b
hh: .word 0x3921777a
ii: .word 0
```
2. (CLO 5—Assy Lang.) A factorial program (a program that calculates $n!$, $n$ a whole number) can be easily constructed using a recursive approach.

However, it is fairly easy to make a non-recursive program that can also calculate factorials.

Construct such a program that calculates the factorials of 1 to 12 (anything larger would require floating-point calculations). The program should ask for a whole number $n$ to be input between 1 and 12, check its size, and if it is within the limits, find $n!$

If the number is not within the limits, output a bad number warning and restart the program. Give the program the capability to do more than one number without restarting.

You may use the output statements shown in the data declaration to the right. Also, since you will output a statement that “$n$ factorial is xxx,” a space is provided “num”) to store the number until you are outputting the answer.

```
data
str: .asciiz "Input whole number between 0 and 12.\n"
str2: .asciiz "Bad number. Starting over."
str3: .asciiz "factorial is 
str4: .asciiz "\nDo another factorial (y = yes)?\n"
num: .word 0
```
3. (CLO 5—Assy Lang.) Write a brief loop program to print out the ASCII characters for numbers in the data words “data1” and “data2” in the data statement to the right. Note that since the two words have 8 bytes, the loop needs to be traversed eight times, so you will need a counter. The hex values for ASCII characters 0-9 are 0x30-0x39.

You do not have to print a leader or other statement—simply print out the sequence of numbers.

What number sequence is printed out?

```
.data
data1: .word 0x327a2131
data2: .word 0x38413735
```
4. (CLO 5—Assy Lang.) Create a timer program that will output a timed statement of a number of seconds. That is, it will put out a phrase “X seconds,” for an interval from one to thirty seconds. It should time the interval 20 times, then stop.

For example, if you want to time 5 second intervals, the program would let you input a 5, then time 5 seconds twenty times, outputting the statement “5 seconds” every five seconds for twenty intervals. Then it would automatically stop.

This means that you need to come up with a countdown time for seconds. In my computer, a countdown from 200,000 produces a 1-second interval, so you need only multiply the number of seconds desired by 200,000 to get the correct interval. Make sure to output a CR/LF between outputs for neatness. You may need to experiment a bit to come up with the number that will produce a 1-second interval, as every computer system is a bit different.

You may use the data declaration shown if you wish.

```
.text
main:

.data
num: .word 0
secs: .asciiz " secs."
ldr: .asciiz "Input number of seconds."
```
5. (CLO 5—Assy Lang.) In the early 13th century, the mathematician Fibonacci developed a formula for a number series that predicted the reproduction rate of rabbits. This series turned out to be an important mathematical development.

The general formula for a Fibonacci number $F(n)$ is: $F(n) = F(n-1) + F(n-2)$, $n$ an integer $> 0$, where $F(0)$ is defined as 0, and $F(1)$ is defined as 1.

It turns out that a very neat recursive routine to calculate $F(n)$, where $F(n)$ is the Fibonacci number for the integer $n$, can be developed. As a matter of fact, such a problem was on a previous version of this test review. However, consider a non-recursive program to calculate $F(n)$. Such a program is a bit simpler, requiring only that you keep count of how many $F(m)$'s ($m < n$) you need to calculate to get the desired $F(n)$. Due to our using fixed point calculations, restrict the input $n$ to 2-40. Use the data declarations shown to the right.

**Hints:**
(1) Note that $F(2)$ is the smallest $F(n)$ that would have to be calculated, as $F(0)$ and $F(1)$ are defined.
(2) All you must do is calculate successively larger $F(m)$'s, until you arrive at $n$. That is, $F(2) = F(1) + F(0)$, $F(3) = F(2) + F(1)$, $F(4) = F(3) + F(2)$, etc., until you arrive at $F(n) = F(n-1) + F(n-2)$.

When the program is working properly, calculate $F(5)$, $F(10)$, $F(20)$, $F(30)$, and $F(40)$.

$F(5) = 5$, $F(10) = 55$, $F(20) = 6765$, $F(30) = 832,040$, $F(40) = 102,334,155$.

Note: Although not recursive, this problem requires some thought and a well-planned approach.