1. (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: li $t9,0
la $a0,ldr
li $v0,4
syscall
li $v0,5
syscall
blt $v0,1,main
bgt $v0,30,main
move $t0,$v0
sw $t0,num
li $t1,200000
mul $t2,$t1,$t0
nxt: move $t3,$t2
count: sub $t3,$t3,1
beqz $t3,out
j count
out: lw $a0,num
li $v0,1
syscall
la $a0,secs
li $v0,4
syscall
addi $t9,$t9,1
beq $t9,20,done
j nxt
done: li $v0,10
syscall

.data
num: .word 0
secs: .asciiz " secs."
ldr: .asciiz "Input number of seconds."
```
2. (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.
3. (CLO 5—Assy Lang.) Compose a program to initialize an array with a series of numbers. This program will be very similar to one that we did in class, but with a twist or two. The array is to be 20 X 20 words, so you will need to use the “.space” command to initialize the array size. It should be 400 words (20 X 20), but remember, using the “.space” directive, you have to reserve bytes.

In this case, you are going to load each row of the array with slightly different numbers. You will be loading rows 0-19, and columns 0-19, or 20 X 20. Remember that the first row is 0! For even rows (including 0, that is, 0, 2, 4, 6, etc.), load the numbers 1-20 consecutively in the 20 columns. For odd rows (1, 3, 5, 7, etc.) load the numbers 20-1 consecutively, that is, in reverse order.

Thus, you will load even rows with the sequence 1-20, and odd rows with the sequence 20-1.

As in problem one, you can print out and paste a copy of the program in the space at the right for your records.

Comments: This program will take a little time, but it is relatively simple if you take the time to review the array program shown in class. Note that, as in the classroom problem, you will need nested loops to do the program most easily.

However, unlike the classroom example, you are NOT storing a simple value. The value you store must be incremented every time a column element is stored.

Further, adjacent rows have elements that “go in the opposite direction.” Even rows count UP, odd rows count DOWN.

You may have found (as the example to the right shows) that having an “even loop” and an “odd loop” simplifies bookkeeping in the inner loop storage routine. That is, setting up TWO (2) inner loops helps.

---

```assembly
.data
array: .space 1600

.text
main: la $t9,array # load array address
      li $t0,0  # initializing row counter
row:  beq $t0,20,done # done yet?
      li $t1,0  # initializing column counter
      mul $t4,$t0,80 # initializing row address
      add $t9,$t9,$t4 # 
      div $t2,$t0,2 # odd or even row?
      mfhi $t2 # 
      bnez $t2,orow # 
      erow: li $t5,1  # intialize even-row number
      eloop: sw $t5,0($t9) # store element
        addi $t5,$t5,1 # increment element value
        j eloop  # do it again
      urow:  addi $t0,$t0,1 # increment row counter
        la $t9,array # re-initialize array address
        j row      # next row
orow:  li $t5,20 # initialize even-row number
        oloop: sw $t5,0($t9) # store element
        addi $t1,$t1,1 # increment column counter
        beq $t1,20,urow # stored 20 columns?
        addi $t9,$t9,4 # no, calc. add. or next pos.
        addi $t5,$t5,1 # increment element value
        j oloop    # do it again
done:  li $v0,10  # done; QUIT!
syscall

# Array Program -- Loads a 20 Word X 20 Word Array With Numbers
```

---