JHU Computer Organization

Module 6 Assignment - Lottery Odds

There are many different lottery schemes across the world. In this assignment you will write a MIPS assembly program to calculate the odds of the following lotteries:

1. The Euro Millions lottery which requires matching 5 numbers from a pool of 50 in addition to 2 numbers from a second pool of 12.
2. The Mega Millions lottery which requires matching 5 numbers from a pool of 70 in addition to 1 number from a second pool of 25.
3. The Thunderball lottery which requires matching 5 numbers from a pool of 39 in addition to 1 number from a second pool of 14.
4. The Powerball lottery which requires matching 5 numbers from a pool of 69 in addition to 1 number from a pool of 26.

Lottery Math

Keep in mind how a lottery works: it uses something called sampling without replacement. This means once a number is drawn from a pool of numbers it is kept aside and not put back into the pool. Mathematically we can count the number of ways to draw r balls from a pool of n balls using a "combination":

$$\frac{n!}{r!(n-r)!}$$

So, choosing r = 2 balls from a pool of n = 4 ball, we get: 4!2!(4-2)!=4*3*2*12*1*(2*1)=244=6, which means there are 6 ways to choose 2 balls from a pool of 4. Specifically: (0, 1) (0, 2) (0, 3) (1, 2) (1, 3) (2, 3). Notice that we are not concerned about the order. That is, we consider the pair (0, 1) and (1, 0) to be the same. If we cared about order we would use permutations rather than combination. So, the odds we win a lottery by selecting 2 balls from a pool of 4 is 1 in 6 = ~16%.

You probably noticed we used the factorial function in our calculation. This function is a perfect candidate for the use of recursion to calculate the final value. On the following page we've given you a recursive function called ***'factrl'***. You pass in the value for which you'd like to calculate the factorial using register $a0 and the function will return the result to you in register $v0.

Before you implement your full solution, I recommend using the following simple test case to make sure your code is working correctly: calculate the odds of choosing 8 balls from a pool of 15 and 1 ball from a pool of 2. The solution should be 1 in 12870.

Programming Assignment

Please write a MIPS program that does the following for each of the 4 lotteries listed above:

Prompts the user for 4 numbers

a. An integer representing the size of the large pool
b. An integer representing the number of balls to be pulled from the large pool
c. An integer representing the size of the small pool
d. An integer representing the number of balls to be pulled from the small pool

Calculates the odds of winning the lottery

Displays the odds to the console in the following format

- "The odds are 1 in N" where N is the value you computer in Step 2

Note: As you know, MIPS can only store values up to a single 32-bit word. That means MIPS cannot store a single precision integer larger than 12!. Fortunately, you can make some algebraic simplifications to make this work. Please see the posted hint for details, but I also encourage you to think about it on your own first. Namely, how would you simplify the formula for calculating the number of combinations? Try working through a simple example and see if anything consistently cancels out in the numerator and denominator.

Please submit your factorial.s file, along with screenshots of the outputs for each of the lotteries.

Factorial Function Code

```
#   Given n, in register $a0;
#   calculate n!, store and return the result in register $v0

factrl:    sw   $ra, 4($sp)    # save the return address
           sw   $a0, 0($sp)    # save the current value of n
           addi $sp, $sp, -8   # move stack pointer
           slti $t0, $a0, 2    # save 1 iteration, n=0 or n=1; n!=1
           beq  $t0, $zero, L1 # not less than 2, calculate n(n-1)!
           addi $v0, $zero, 1  # n=1; n!=1
           jr   $ra            # now multiply

L1:        addi $a0, $a0, -1   # n = n-1

           jal factrl          # now (n-1)!

           addi $sp, $sp, 8    # reset the stack pointer
           lw   $a0, 0($sp)    # fetch saved (n-1)
           lw   $ra, 4($sp)    # fetch return address
           mul  $v0, $a0, $v0  # multiply (n)*(n-1)
           jr   $ra            # return value n!
```