ECE 153A Homework 2

1. Microblaze Processor: Read the slide set "MicroBlaze Overview" and "Xilinx Interrupt Handling..." from Canvas.

   (a) Describe how branch delay slot instructions can improve the code throughput.

   (b) If interrupt latency is of key importance to a design, why might support for hardware divide instructions be an issue?

   (c) Consider a Microblaze processor with stack pointer at address 0x0b0. At this point, the below fib() function is called with $n = 1000$. Estimate how many recursive calls you can support before serious trouble is likely. What trouble? (describe your assumptions)

   ```
   unsigned int fib(unsigned int n)
   {
      if (n <= 1)
          return n;
      return fib(n - 1) + fib(n - 2);
   }
   ```

   (d) Consider a function call whose parameter is a small 6x6 array of long ints which is declared in the parent as simply "long int". Where, physically (i.e. on the stack, in the heap, in initialized memory) is the storage for the array allocated? Where is it stored if declared as "static"?

   (e) A bug in an otherwise carefully checked embedded system is diagnosed as the stack growing too large during some rare execution traces. To fix this problem, the data segment of the program is moved from the shared BRAM to the much larger DRAM device on the FPGA Board. The larger space fixed the stack overflow, but the new version exhibits a variety of odd bugs including missing some interrupts. Why might this be happening?

2. Consider the following code fragment:

   ```
   static int var;
   void bar(void){
       var=0;
       while (var!=255);
   }
   ```

   Most compilers will notice that no other code can possibly change the value stored in var, and will assume that it will remain equal to 0 at all times. The compiler will therefore replace the function body with an infinite loop similar to this:

   ```
   void bar_optimized(void){
       var=0;
       while (True);
   }
   ```

However, `var` might represent a location that can be changed by other elements of the computer system at any time, such as a hardware register of a device connected to the CPU. The above code would never detect such a change. Rewrite to show how to prevent this from happening.

3. The attached code **interrupt_model.c** contains a model for a system that can be interrupted by 2 interrupts:

Interrupt	Run-time(clock cycles)	Probability	Priority
Interrupt0	3	0.1	1
Interrupt1	5	0.03	2

Rules for servicing the interrupts are as follows:

- The interrupt with the higher priority value will be serviced first, if the two interrupts arrive at the same time.
- The system has a mechanism to mark **one** interrupt of each kind as "pending", if it cannot be serviced immediately after its occurrence. This happens if any interrupt is being serviced when the current interrupt is fired.
- If any interrupt is "pending" and a new request for the same interrupt occurs, the request is "missed" and is not serviced at all. The program keeps track of the number of missed interrupts for each kind.
- If more than one interrupt is pending, the interrupt with the higher priority will be serviced first.

(a) Plot a CDF for the representative service time. What is the mean service time and indicate the 95% confidence interval. Estimate the probability of a service time longer than 35 cycles.

(b) Increase the probability of Interrupt0 to 0.2, roughly doubling the number of interrupt requests of that type. What happens to the number of missed interrupts and the worst case latency of both interrupt requests? Explain your findings.

(c) Augment the program to model an additional interrupt: 'Interrupt2' with run time of 7 and probability of 0.001, as well as the other two interrupts in part(a) above. Estimate this interrupt's mean service time when its priority is lower than the other 2 interrupts. What does this addition do to the miss rates for the other interrupts?