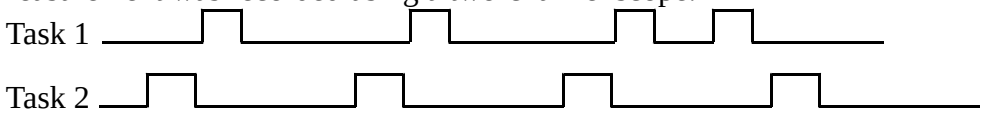
Jonathan W. Valvano First Name: _____ Last Name:_____
February 25, 2011, 11:00 to 11:50am

Quiz 1 is a closed book exam. You may have one 8.5 by 11 inch sheet of hand-written crib notes, but no books or electronic devices. You may put answers on the backs of the pages, and please staple the crib sheet to your exam.

(10) Question 1. A real-time system executes two periodic tasks. The goal is to execute both tasks at constant rates; although the rates for each are different. Both tasks are interrupt-timer driven. Each task sets an output pin high when it starts and clears that pin when it finishes. The following debugging measurement was recorded using a two-channel scope.



Part a) What type of error has occurred? Give the name of this error.

Part b) Which task has priority?

(10) Question 2. Consider this function and the assembly code created by the compiler.

```
sum:
   ADD  r0,r0,r1
   BX   lr
```

```
long sum(const long n, const long m){
static long result;
  result = m+n;
  return result;
}
```

Part a) Is this implementation of the function reentrant? Justify

Part b) How are the input parameters `n` and `m` passed?

Part c) What is in LR during the execution of the function?

(10) Question 3. Consider the following interrupt service routine and the resulting assembly code created by the compiler. These two **#define** definitions are bit-banded addresses for Port F.

```
SysTick_Handler
    LDR r0,[pc,#8]
    LDR r1,[r0,#0x04]
    EOR r1,r1,#0x01
    STR r1,[r0,#0x04]
    BX  lr
```

```
#define GPIO_PF0 (*((volatile unsigned long *)0x40025004))
#define GPIO_PF1 (*((volatile unsigned long *)0x40025008))
void SysTick_Handler(void) {
  GPIO_PF0 = GPIO_PF0^0x01;
}
void otherFunction(void) { // called from foreground
  GPIO_PF1 = GPIO_PF1^0x02;
}
```

Part a) Do these two code segments have a critical section? Justify.

Part b) Why did the compiler choose R0, R1 in the ISR rather than R4 and R5?

Part c) What is in LR during the execution of this ISR?

(10) Question 4. For each term, give a definition in 20 words or less.

Part a) Bounded waiting

Part b) Atomic

Part c) Slack time

Part d) Hook

Part e) Stabilization

(10) Question 5. There are three semaphores in this system, which are all initialized to 1. These are the only threads that access semaphores s1 s2 s3.

Thread1	Thread2	Thread3
wait(&s1) wait(&s2)	wait(&s2) wait(&s3)	wait(&s1) wait(&s3)
stuff1	*stuff2*	*stuff3*
signal(&s1) signal(&s2)	signal(&s2) signal(&s3)	signal(&s1) signal(&s3)

Can a deadlock occur? If so, give the order of execution resulting in a deadlock. If not, prove it.

(25) Question 6. Your Lab 2 operating system implements spin-lock counting semaphores. Assume the prototypes are

```
void OS_Wait(long *semaPt);
void OS_Signal(long *semaPt);
```

Implement an OS function that will wait on two semaphores. You will not be calling existing **`OS_Wait`** and **`OS_Signal`**, but rather you will be writing a new **`OS_Wait2`**. In particular, it will decrement both semaphores only if both are greater than or equal to 1. If either semaphore is 0, it will spin. You can use macros or inline assembly; just explain what they do in comments. The prototype is

```
void OS_Wait2(long *sema1Pt, long *sema2pt);
```

(25) Question 7. Consider an operating system that employs a priority scheduler without aging. Each thread has a unique priority (no two threads will have the same priority.) The TCBs are kept in a sorted single-linked list. If a thread is blocked or killed, it will be removed from the list. If it is sleeping its sleep counter is non zero. There is a **`StartPt`** that points to the first thread (highest priority thread). The list is always in sorted order by priority. The last thread in the single-linked list has its **`Next`** field null (0).

```
struct TCB {
  long *stackPointer;   // pointer to top of stack
  unsigned long Id;     // thread number, zero if this TCB is free
  struct TCB *Next;     // single-linked list
  long Priority;        // 0 highest, 1 is next highest...
  unsigned long Sleep;  // nonzero if this thread is sleeping
};
typedef struct TCB TCBType;
typedef TCBType * TCBPtr;
TCBPtr RunPt;   // Pointer to tcb of thread currently running
TCBPtr StartPt; // Pointer to first tcb in list
TCBPtr newPt;   // Pointer to tcb to be added
TCBPtr nextPt;  // Pointer to tcb to run next
```

Part a) Assume **`StartPt`** points to the list of threads. In particular, write code, given **`StartPt`**, which sets **`nextPt`** to point to the TCB with the highest priority non-sleeping thread. Set **`nextPt`** to null if there are no threads or if all threads are sleeping.

Part b) Assume **`newPt`** points to a TCB for a new thread that has every field set except the Next field. Show the C code that enters the new TCB into the linked list in priority-sorted order. If **`StartPt`** is null, the linked list is empty. This code would run as part of **`OS_AddThread`**.