Jonathan W. Valvano First Name: _______________ Last Name:____________________

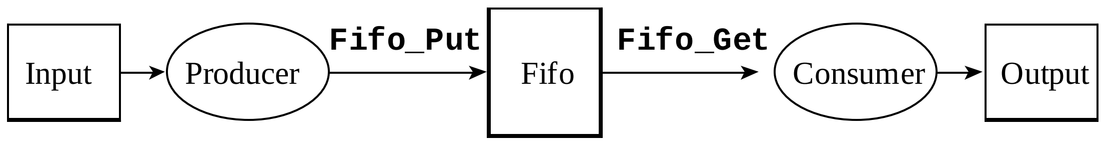
November 9, 2005, 1 to 1:50pm

This is an open book, open notes exam. You may put answers on the backs of the pages, but please don't turn in any extra sheets.

(10) Question 1. In a CAN network, what is the purpose of the **CRC** field? I.e., What is it used for?

(10) Question 2 Assuming transfer rate of 100,000 bits/sec on a CAN network, and each message contains 4 bytes, what is the maximum bandwidth of the network (in units of bytes of data per sec)? The important part of this question is the development of the equation, and the calculation of the specific number is of secondary importance.

(15) Question 3. Consider a producer/consumer problem linked by a FIFO queue. Both the producer thread and the consumer thread operate in the background using interrupt synchronization. The input device is a CAN receiver, and the output device is a SCI transmitter. When the CAN input is ready an interrupt-38 is generated, and the producer thread (CAN input ISR) reads the data and puts them into a FIFO. When the SCI output is idle, an interrupt-20 is generated, and the consumer thread (SCI output ISR) gets data from the FIFO and writes them to the output device.



Part a) The initialization software will clear the FIFO. Which threads should be armed at this time? *Circle your answer.*

A) The consumer (SCI output)

B) The producer (CAN input)

C) Both

D) Neither

Part b) After the producer thread puts data into the FIFO, it checks the FIFO status. It will disarm itself if it finds the FIFO is full. When should the producer thread be rearmed? *Circle your answer.*

A) Only by the ritual

B) On the next output interrupt (when the SCI output device is idle)

C) On the next input interrupt (when new CAN input is received)

D) The producer will call `Fifo_Put` over and over until it is not full

Part c) The consumer thread disarms itself if it finds the FIFO is empty. When should the consumer thread be rearmed? *Circle your answer*

A) Only by the ritual

B) On the next output interrupt (when the SCI output device is idle)

C) On the next input interrupt (when new CAN input is received)

D) The consumer will call `Fifo_Get` over and over until it is not empty

(25) Question 4. Consider a problem of running two foreground threads using a preemptive scheduler with semaphore synchronization (like Lab 17.) There is a shared 16-bit global variable:

```
short TheData;
```

The `writer` thread stores into `TheData`, and the `reader` thread reads from `TheData`. The goal is to create a 1-1 data transfer (repeating the pattern where one write is followed exactly one read). The basic shell of this operation is given. Define one or more semaphores, then add calls to the following three functions in order to properly synchronize the interactions between `writer` and `reader`.

```
int OS_InitSemaphore(Sema4Type *semaPt, short value);
void OS_Wait(Sema4Type *semaPt);
void OS_Signal(Sema4Type *semaPt);
```

You will define one or more semaphores and place calls to the three semaphore functions into the system, otherwise no other changes are allowed. Use descriptive names for the semaphores that describe what the semaphores mean. Assume `writer` is run first. You may assume the only accesses to `TheData` in the entire software system are explicitly shown here.

`void writer(void){`
 	
`  wInit();   // initialization`
 	
`  while(1){`
 	
`    TheData=wProcess(); // body`
 	
`  }`
`}`
	`void reader(void){`
	
	`  rInit();   // initialization`
	
	`  while(1){`
	
	`    rProcess(TheData);  // body`
	
	`  }`
	`}`

The purpose of the semaphores is to force the sequence of execution so that exactly one call to `wProcess` is followed by exactly one call to `rProcess`.

```
    TheData = wProcess(); // writer body
    rProcess(TheData);    // reader body
    TheData = wProcess(); // writer body
    rProcess(TheData);    // reader body
    TheData = wProcess(); // writer body
    rProcess(TheData);    // reader body
     ...
```

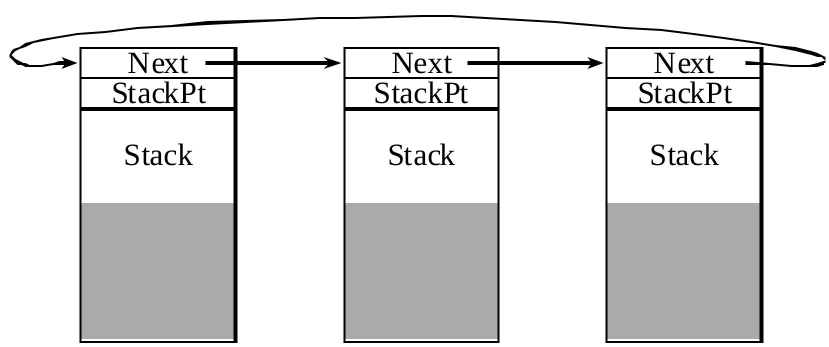
(30) Question 5. The goal of this problem is to design a cooperative thread switcher. There will be no interrupts whatsoever, just the **`SWI`** instruction that causes a software interrupt. The main program creates three threads and launches the first one. The threads are chained in a circle using the **`Next`** pointers in the TCB. All threads will cooperate by calling your **OS_Switch()** function regularly. The following thread control block will be used (like Lab17, except the **`Id`** removed.)

```
struct TCB{
 struct TCB *Next;
 unsigned char *StackPt;
 unsigned char MoreStack[99];
 unsigned char InitialCCR;
 unsigned char InitialRegB;
 unsigned char InitialRegA;
 unsigned short InitialRegX;
 unsigned short InitialRegY;
 void (*InitialPC)(void);
};
typedef struct TCB TCBType;
typedef TCBType * TCBPtr;

TCBType SystemTCB[3];

TCBPtr RunPt; // current

unsigned short NumThread=0;
```

```
void Thread1(void){
  Init1();
  while(1){
    Process1();
    OS_Switch();
  }
}
void Thread2(void){
  Init2();
  while(1){
    Process2();
    OS_Switch();
  }
}
void Thread3(void){
  Init3();
  while(1){
    Process3();
    OS_Switch();
  }
}
void main(void){
  OS_AddThread(&Thread1);
  OS_AddThread(&Thread2);
  OS_AddThread(&Thread3);
  OS_Launch();    //    doesn't
return
}
```

Execution Sequence

```
Process1();
Process2();
Process3();
Process1();
Process2();
Process3();
  •
  •
  •
```



NO hardware interrupts are allowed in this problem. You are not allowed to change the TCB structure or the code of the foreground threads **Thread1 Thread2 Thread3** or **main**. Code to add threads and launch are similar to Lab17 (except no **Id**), which you are also not allowed to change.

```
void OS_Launch(void){
    RunPt = &SystemTCB[0];  // Specify first thread to run
asm ldx RunPt
asm lds 2,x
asm rti                     // Launch First Thread
}
short OS_AddThread(void(*fp)(void)){
  if(NumThread >= 3) return 0;  // structure is full
  if(NumThread) SystemTCB[NumThread-
1].Next=&SystemTCB[NumThread];
  SystemTCB[NumThread].StackPt =
&SystemTCB[NumThread].InitialCCR;
  SystemTCB[NumThread].InitialCCR = 0x50;    // CCR I bit set
  SystemTCB[NumThread].InitialPC = fp;       // Initial PC
  SystemTCB[NumThread].Next = &SystemTCB[0];
  NumThread++;
  return 1; }
```

(10) Part a) Write the function **OS_Switch**, which issues a SWI.

(20) Part b) Write the SWI interrupt handler that suspends the current thread and runs the next thread in the circular linked list.

```
interrupt 4 void SWIhandler(void){
```

(10) Question 6. In Lab 17, we defined time-jitter, d**t**, as the difference between when a periodic task is supposed to be run, and when it is actually run. The goal of a real-time DAS is to start the ADC at a periodic rate, Δt. Let t_n be the nth time the ADC is started. In particular, the goal to make $t_n - t_{n-1} = \Delta t$. The jitter is defined as the constant, d**t**, such that

Δt-d**t** < $t_i - t_{i-1}$ < Δt+d**t** for all i.

Assume the input to the ADC can be described as **V(t) = A+Bsin(2**p**ft)**, where **A**, **B**, **f** are constants. Derive an estimate of the voltage error, d**V**, caused by time-jitter. Basically, solve for d**V** as a function of d**t**, **A**, **B**, and **f.**