Jonathan W. Valvano

First Name: ________________ Last Name:_____________________

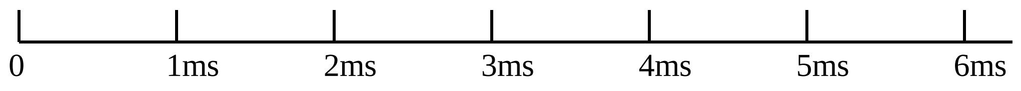
November 8, 2006, 1 to 1:50pm

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.

(15) Question 1. Consider a situation where 4 microcontrollers are connected together using a CAN network. Assume for this question that each frame contains 100 bits. Also assume the baud rate is 100,000 bits/sec, therefore it takes 1ms to send a frame. Initially, the CAN controllers are initialized (i.e., all computers have previously executed **`CAN_Open`**).

At time = 0	computer A calls **`CAN_Send`** with ID=1000
At time = 300us	computer B calls **`CAN_Send`** with ID=800
At time = 500us	computer C calls **`CAN_Send`** with ID=900
At time = 700us	computer D calls **`CAN_Send`** with ID=600

Specify the time sequence in which the four frames occur on the CAN network. Clearly define the begin and end times when each message is visible on the CAN network.



(10) Question 2. Assume the CAN transfer rate is 100,000 bits/sec, the system uses 11-bit IDs, there are 8 nodes on the network, each node sends a message every 0.1 sec, and each message contains 5 bytes of data, what is the ***actual*** bandwidth of this network (in units of bytes of data per sec)? You may neglect stuff bits. The total number of bits in a CAN frame can be calculated as the number of ID bits plus the number of data bits plus 36 bits. The important part of this question is the development of the equation, and the calculation of the specific number is of secondary importance.

(25) Question 3. The CAN physical channel defines one bit at a time in such a way that 0 dominates over 1. In particular, if one node is attempting to drive CANH to 2.5 and a second node tries to make CANH 3.75, then the actual CANH signal goes to 3.75. Similarly, if one node is attempting to drive CANL to 2.5 and a second node tries to make CANL 1.25, then the actual CANL signal goes to 1.25. Notice also that CANH + CANL always equals 5.0 volts.

Part a) Define a new voltage protocol that transmits two bits at a time in such a way that 00 dominates over 01, which dominates over 10, which dominates over 11. There will still be two signals, CANH and CANL, such that CANH + CANL always equals 5.0 volts. In order for the interface to operate on a single 5 V supply, all voltages must be between 1 and 4 volts.

Digital input	CANH	CANL	CANH+CANL
0 0			5.00 V
0 1			5.00 V
1 0			5.00 V
1 1			5.00 V

Part b) Demonstrate how your system will implement dominance by considering a simple case with two nodes simultaneously transmitting. You will show just the functionality of CANH, but a corresponding operation will occur on CANL. Since the system will be symmetric (fair) with respect to nodes A and B, the situations with A dominating over B were removed from the table.

Node A[1]	CANH (A)[2]	Node B[3]	CANH (B)[4]	CANH[5]	
0 0		0 0			**A=B**
0 1		0 0			**B>A**
1 0		0 0			**B>A**
1 1		0 0			**B>A**
0 1		0 1			**A=B**
1 0		0 1			**B>A**
1 1		0 1			**B>A**
1 0		1 0			**A=B**
1 1		1 0			**B>A**
1 1		1 1			**A=B**

A=B means A and B continue transmitting, and dominance occurs later
B>A means B asserts dominance over A, and A stops transmitting

[1] This is the digital logic that Node A is attempting to send
[2] Fill in with the value from your Table in part a)
[3] This is the digital logic that Node B is attempting to send
[4] Fill in with the value from your Table in part a)
[5] This will be the actual voltage on the CANH signal

(25) Question 4. Assume the binary semaphore type is a simple 8-bit integer, e.g.,

```
char s1;  // binary semaphore can be 0 or 1
```

The following implementation of the binary spinlock semaphore is proposed.

```
void OS_InitSemaphore(char *semaPt, char value){
  *semaPt = value;
}
void OS_bWait(char *semaPt){ // semaPt in RegD
asm tfr  d,x    // RegX->semaphore
asm ldaa #0     // RegA=0
asm minm 0,x    // replace memory with the smaller of 0 and memory
                // minm sets the C bit if it changes from 1 to 0
asm bcc *-3     // execute minm until changes from 1 to 0
}
void OS_bSignal(char *semaPt){ // semaPt in RegD
asm tfr  d,x    // RegX->semaphore
asm ldaa #1     // RegA=1
asm staa 0,x    // semaphore is set to 1
}
```

Part a) Does this implementation have any critical sections? If so, identify the specific place(s) where the critical section exists.

Part b) In this part, assume the semaphore is initially 1, and two or more threads have started to execute **`OS_bWait`**. Fill in the blank:

The first thread to execute the _______________ instruction will be the one to return from **`OS_bWait`**, *while the other thread(s) will spin in the* **`minm/bcc`** *loop.*

Part c) Why is this implementation better than the spinlock implement you created in Lab 18?

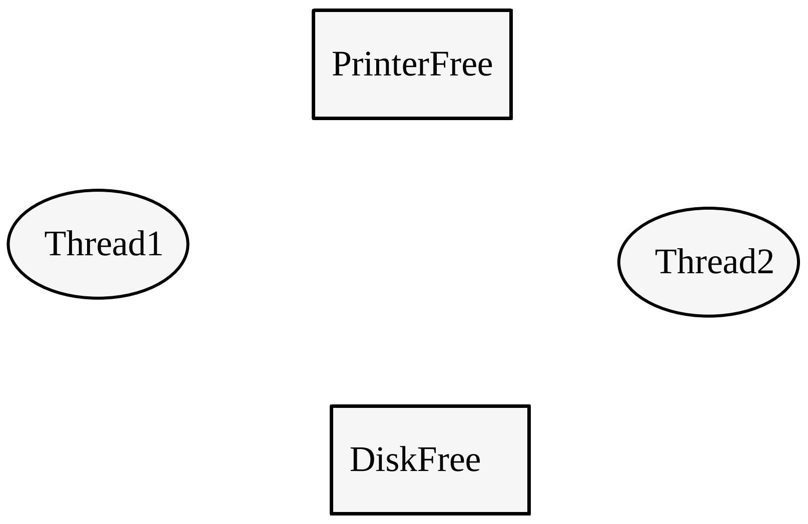
(25) Question 5. Consider a problem of deadlocks that can occur with semaphore synchronization. The following is a classic example that might occur if two threads need both the disk and the printer. In this example, the disk has a binary semaphore **`DiskFree`**, which is 1 if the disk is available, and similarly the printer has a binary semaphore **`PrinterFree`**, which is 1 if the printer is available. A deadlock occurs if each thread gets one resource then waits (on each other) for the other resource. In this example, we assume there is one disk and one printer.

```
void thread1(void){
  OS_bWait(&DiskFree);
  OS_bWait(&PrinterFree);

// use disk and printer

  OS_bSignal(&DiskFree);
  OS_bSignal(&PrinterFree);
}
```
	```
	void thread2(void){
	  OS_bWait(&PrinterFree);
	  OS_bWait(&DiskFree);

	// use printer and disk

	  OS_bSignal(&PrinterFree);
	  OS_bSignal(&DiskFree);
	}
	```

In this problem we will develop a graphical method (called a **resource allocation graph**) to visualize/recognize the deadlock. Draw each thread in your system as an oval, and each binary semaphore as a rectangle. If a thread calls **`OS_bWait`** and returns, then draw an arrow (called an *allocation edge*) from the semaphore to the thread. An arrow from a semaphore to a thread means that tread owns the resource. If a thread calls **`OS_bSignal`**, then erase the previously drawn allocation edge. If a thread calls **`OS_bWait`** and spins or blocks because the semaphore is not free, then draw an arrow from the thread to the semaphore (called a *request edge*). An arrow from a thread to a semaphore means that tread is waiting for the resource associated with the semaphore.

Part a) Draw the resource allocation graph that occurs with the deadlock sequence

1) thread1 executes **`OS_bWait(&DiskFree);`**
2) thread2 executes **`OS_bWait(&PrinterFree);`**
3) thread2 executes **`OS_bWait(&DiskFree);`**
4) thread1 executes **`OS_bWait(&PrinterFree);`**

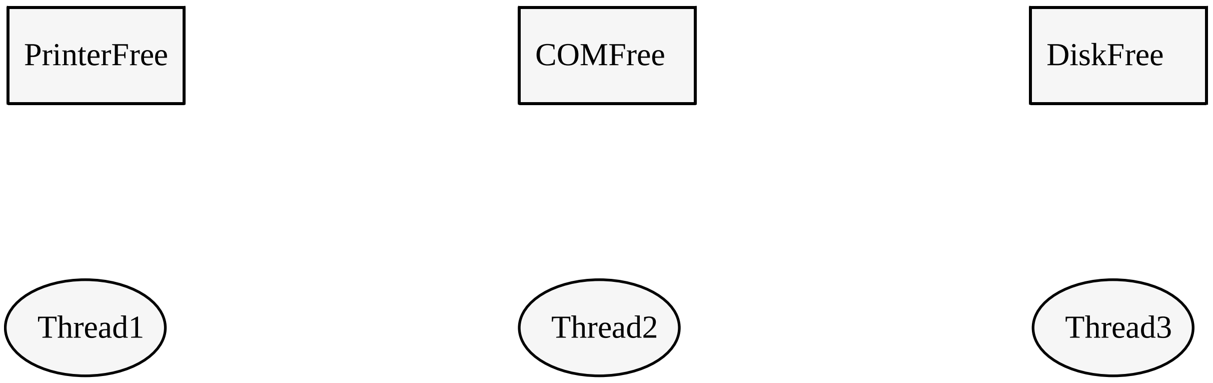


Part b) This method can be generalized to detect that a deadlock has occurred with an arbitrary number of binary semaphores and threads. What shape in the resource allocation graph defines a deadlock? In other words, generalize the use of this method such that you can claim

"There is a deadlock if and only if the resource allocation graph contains a shape in the form of a ______________________".

Part c) Justify your answer by giving a deadlock example with three threads and three semaphores. In particular, give 1) the C code; 2) the execution sequence; 3) the resource allocation graph

`void thread1(void){`	`void thread2(void){`	`void thread3(void){`
`}`	`}`	`}`



MINM — Place Smaller of Two Unsigned 8-Bit Values in Memory — MINM

Operation: MIN ((A), (M)) $\Rightarrow$ M

Description: Subtracts an unsigned 8-bit value in memory from an unsigned 8-bit value in accumulator A to determine which is larger and leaves the smaller of the two values in the memory location. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 1, the value in accumulator A has replaced the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is \$00; cleared otherwise

V: $A7 \bullet \overline{M7} \bullet \overline{R7} + \overline{A7} \bullet M7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{A7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{A7}$
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

Condition codes reflect internal subtraction (R = A – M)

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
MINM *oprx0_xysp*	IDX	`18 1D xb`	`OrPw`	`OrPw`
MINM *oprx9,xysp*	IDX1	`18 1D xb ff`	`OrPwO`	`OrPwO`
MINM *oprx16,xysp*	IDX2	`18 1D xb ee ff`	`OfrPwP`	`OfrPwP`
MINM [D,*xysp*]	[D,IDX]	`18 1D xb`	`OfIfrPw`	`OfIfrPw`
MINM [*oprx16,xysp*]	[IDX2]	`18 1D xb ee ff`	`OfIPrPw`	`OfIPrPw`

BCC — Branch if Carry Cleared (Same as BHS) — BCC

Operation: If C = 0, then (PC) + \$0002 + Rel ⇒ PC

Simple branch

Description: Tests the C status bit and branches if C = 0.

LDAA — Load Accumulator A — LDAA

Operation: (M) ⇒ A

Description: Loads the content of memory location M into accumulator A. The condition codes are set according to the data.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is \$00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
LDAA #*opr8i*	IMM	`86 ii`	`P`	`P`
LDAA *opr8a*	DIR	`96 dd`	`rPf`	`rfP`
LDAA *opr16a*	EXT	`B6 hh ll`	`rPO`	`rOP`
LDAA *oprx0_xysp*	IDX	`A6 xb`	`rPf`	`rfP`
LDAA *oprx9,xysp*	IDX1	`A6 xb ff`	`rPO`	`rPO`
LDAA *oprx16,xysp*	IDX2	`A6 xb ee ff`	`frPP`	`frPP`
LDAA [D,*xysp*]	[D,IDX]	`A6 xb`	`fIfrPf`	`fIfrfP`
LDAA [*oprx16,xysp*]	[IDX2]	`A6 xb ee ff`	`fIPrPf`	`fIPrfP`

STAA Store Accumulator A STAA

Operation: (A) ⇒ M

Description: Stores the content of accumulator A in memory location M. The content of A is unchanged.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise
Z: Set if result is $00; cleared otherwise
V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
STAA *opr8a*	DIR	`5A dd`	`Pw`	`Pw`
STAA *opr16a*	EXT	`7A hh ll`	`PwO`	`wOP`
STAA *oprx0_xysp*	IDX	`6A xb`	`Pw`	`Pw`
STAA *oprx9,xysp*	IDX1	`6A xb ff`	`PwO`	`PwO`
STAA *oprx16,xysp*	IDX2	`6A xb ee ff`	`PwP`	`PwP`
STAA [D,*xysp*]	[D,IDX]	`6A xb`	`PIfw`	`PIfPw`
STAA [*oprx16,xysp*]	[IDX2]	`6A xb ee ff`	`PIPw`	`PIPPw`

TFR Transfer Register Content to Another Register TFR

Operation: See table.

Source Form	Address Mode	Object Code(1)	Access Detail	
			HCS12	M68HC12
TFR *abcdxys,abcdxys*	INH	`B7 eb`	`P`	`P`

1. Legal coding for `eb` is summarized in the following table. Columns represent the high-order source digit. Rows represent the low-order destination digit (MSB is a don't-care). Values are in hexadecimal.