UNIVERSITY OF ALBERTA

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ECE 315 – Computer Interfacing

Midterm Examination

Instructor: B. F. Cockburn
Exam date: March 1, 2017
Exam duration: 50 minutes
Aids permitted: A paper or electronic copy of the course lecture slides can be freely consulted.
The Internet cannot be accessed by any device.
Electronic calculators of any kind are permitted.

Instructions:
1. Please enter your printed name, signature and I.D. number on this page.
2. Verify that this booklet contains 6 pages (including the cover page).
3. Neatly enter your answers in the spaces provided.
4. Use the reverse sides of the pages for rough work.
5. Take into account the marks per question when budgeting your time.

Student name: _____**Model**_____**Solutions**_____
Family name **Given name**

Signature: _____

Student I.D.: _____

Question	Time	Worth	Mark	Subject
1.	10	20		Basic Concepts
2.	10	20		Software in Embedded Systems
3.	10	20		Multitasking Software
4.	10	20		Interrupt-driven Data Transfer
5.	10	20		Communication Interfaces
Total	**50 mins**	**100**		**—**

Question #1 (Basic Concepts)

Carefully explain the differences among the following pairs of concepts:

(a) Automatic memory allocation vs. dynamic memory allocation [4 & 4 marks]

Automatic memory allocation is the process of allocating memory space from the stack to implement objects (including variables) that exist for the execution lifetime of a context, such as function call or a braced block in C. Automatically allocated memory is recycled (automatically) when execution leaves the same context. Note that static local variables in a context are not allocated memory automatically; instead, they are assigned memory when the system image is linked together so that the variable persists between invocations of the function.

Dynamic memory allocation is the process when, in response to a request from a task, the operating system obtains a block of memory of a requested size from the heap of available memory, and returns a pointer to that block to the task. The understanding is that the task will release the block of memory back to the operating system when it has finished using it.

(b) Internally initiated events vs. externally initiated events [3 & 3 marks]

Internally initiated events are (usually potentially interrupt-triggering) hardware conditions that are produced by systems within a microcomputer. Examples would be when a hardware timer counts down to zero, or when an internal DMA-controlled data transfer has completed.

Externally initiated events are (usually potentially interrupt-triggering) hardware conditions that correspond to signal changes that are controlled by systems that are external to the microcomputer. E.g., Internet message arrival, or the human operator pushes buttons on a control panel.

(c) Blocked task vs. interrupted task [3 & 3 marks]

A **blocked task** is a task that cannot execute on the CPU because it is waiting for some condition to be true. There are many possible blocking conditions including semaphores, timers, and message queues. Once a blocking condition goes true, the blocked task is moved to the ready-to-run queue.

An **interrupted task** is a task whose execution has been temporarily stopped to allow an interrupt service routine to execute instead on the CPU. The saved context of the interrupted task is saved on the supervisor stack.

Question #2 (Software in Embedded Systems)

(a) What are the fundamental problems with using global variables in an embedded multitasking system? What can be done within a C programming environment to guard against those problems?

[7 marks] Unless there is some way of ensuring that only one task can write to a global variable, such a variable can be written and changed by any software in the system. This situation can easily lead to complicated interactions and quite possibly erroneous interactions between the software that reads and writes the global variable. The situation is even more complicated in multitasking systems, when the sequence of task executions (and accesses to global variables) can be difficult or even impossible to predict.

[3 marks] In the C programming environment, there is no way of enforcing restrictions on which software can write or read from global variables.

(b) How could one use a software object to provide functionality that is very similar to that of a global variable, but safer? What features would the software object have to have to make the situation safer? Explain how those features would improve safety.

[3 marks] In C++ or in another object-oriented programming language, access to global variables can be protected to a large extent by hiding global variables inside global software objects as private variables, and then restricting access to the private variables to the object's member functions.

[7 marks] Those object member functions can restrict access to the variable. For example, only some tasks (maybe only one task) might be allowed to write to the variable with a set member function; all other tasks could be restricted to read-only access through a get member function. Access to the member functions could be enforced according to task. Also, all accesses to the variable through the object's member functions could be logged for debugging purposes.

Question #3 (Multitasking Software)

(a) A critical section is a section of software that must be run to completion once started; otherwise, data structures and/hardware registers may be placed into unintended and possibly incorrect states. When would it be appropriate to protect a critical section by simply disabling interrupts during the execution of the section? In what situations is it more appropriate to protect a critical section using a binary semaphore?

[5 marks] Protecting a critical section by disabling all interrupts would be acceptable if the critical section is relatively short and if the resulting short time delays to the system interrupts would be safe for the system to tolerate. This method would only work of the task code runs in supervisor mode, in M68000 and ColdFire systems.

[5 marks] If disabling interrupts to protect a critical section is too disruptive to system performance, then a semaphore can be used instead. When the semaphore is 1, the first task to arrive at the start of the critical section is allowed inside, and the semaphore is decremented to 0, thereby blocking all other tasks from entering the critical section. When the first task leaves the critical section, the semaphore is changed back to 1 to allow the next task to enter the critical section. Note that the changes to the semaphore must be protected, and this is usually done by locking out all interrupts.

(b) Cooperative multitasking was described in the lectures as being potentially very efficient. Why would its efficiency be potentially higher than that of a multitasking system that uses time slicing? In a real-time application, why would hardware interrupts be required for both a cooperative multitasking system and a time sliced multitasking system?

[5 marks] A cooperative multitasking system can potentially be very efficient because the number of context switches can be minimized. A task, once it starts to execute on the CPU, is allowed to execute without interruption or blocking for as long as required. The overhead of unnecessary context switches can thus be avoided. In a time sliced system, the context switches are triggered by a hardware timer that is unaware of any implications of causing unnecessary context switches. The main advantage of a time-sliced system is that fair sharing of the CPU, according to some predetermined schedule, is enforced by the hardware-triggered interrupts.

[5 marks] A real-time system, whether it be cooperative or time-sliced, still benefits from having hardware interrupts so that time-critical responses can still be provided by the system for interrupt-causing events. Neither cooperative multitasking nor time-sliced multitasking can provide very fast or deterministic servicing for interrupts.

Question #4 (Interrupt-driven Data Transfer)

(a) Briefly explain what is meant when one refers to interrupt-driven data transfer in the transmit direction? Assume that the transmitted data is transmitted as individual bytes.

[6 marks] In interrupt-driven data transfer in the transmit direction, interrupts occur when the transmitter hardware can be reloaded with more data to transmit out of a communications interface. If the transmitted data blocks are bytes, then a transmit interrupt will occur whenever a new byte of data can be loaded into the transmitter hardware by the CPU. Typically the interrupt-triggering event is when the transmit data register becomes empty. Thus the occurrence of the interrupts drives the transmission process, byte-by-byte.

(b) In interrupt-driven data transfer in the transmit direction, when are the interrupts for the transmit data register empty condition enabled and disabled? What would happen if these transmit interrupts were to be enabled, but never disabled?

[8 marks] The transmit data register empty interrupts are enabled by the device driver software when a new transmission needs to be started. The transmit data register empty interrupts need to be disabled later when there is no more data left to be transmitted.

[6 marks] If the transmit data register empty interrupts are never disabled, then once the supply of data to be transmitted is exhausted, the interrupts will continue to fire when there is no more data to be sent. This will likely cause a continuous stream of needless interrupt firings that will waste CPU time, and quite likely prevent the system from progressing with any useful work.

Question #5 (Communication Interfaces)

(a) Ethernet interfaces in microcomputers, like the MCF5234 and the MCF54415, provide independent data buffer systems in both the transmit and receive directions. Using buffers is compatible with dynamic memory allocation, which has benefits with respect to flexibility. What other benefits are provided by the ability to express a transmitted data frame as data stored in a series of buffers of possibly different sizes?

[10 marks] In the transmit direction, the flexibility of having data buffers of different sizes allows the different layers of a layered communication interface to have independent control of their own portions of a data packet. So a first buffer can be used to store the Ethernet header, a second buffer can be used to store the IP header, a third buffer can be used to store the TCP header, a sequence of one or more buffers can be used to store the application payload, and a last buffer can be used to store the checksum. Using separate buffers in this way allows the software, and the associate data buffers, to be strictly partitioned making the software more secure, modular and maintainable.

(b) In the context of RS232 connections, what is meant by a "null modem"? What problem does the null modem solve when a terminal (or another peripheral, like a printer) is connected directly to a computer?

[10 marks]
A null modem is a bundle of wires that implements the connections between two data terminal equipment (DCE) interfaces. The null modem must implement both straight-through power connections as well as cross-over connections for the data signals and the control signals. So the Tx data line on the left-side DTE needs to be connected to the Rx data line on the right-side DTE, and the Tx on the right-side DTE needs to be connected to the Rx on the left-side DTE. The RTS and CTS signals can also be crossed over.

The null model provides an elegant way of encapsulating the wiring connections. The cross-over connections are required to overcome the fact that the original DTE-DCE connection is asymmetrical. This means that a DTE cannot be directly connected to a second DTE. The cross-over connections provide the turn each DTE into a DCE with respect to the other DTE.