

You may discuss these problems with your classmates, as it may be helpful to gain a good understanding of the topic. Nonetheless you should always submit your own work! Use a word processor to answer these questions, attach this questions page to your answers.

Question 1.

In your own words, explain in detail the steps that are followed in the “exception processing sequence”.

There are four general steps that are executed during an exception processing sequence:

1. *Save the old status register and change the processor state:*
 - Make an internal copy of the status register*
 - Set the ‘S’ bit in the status register.*
 - Disable tracing by clearing the ‘T’ bits in the status register.*
 - For interrupts and reset exceptions, update the priority mask bits ‘I2’, ‘I1’, and ‘I0’ in the status register.*
2. *Get the corresponding exception vector number:*
 - Use the internal CPU logic to determine the exception vector number.*
 - For interrupts, obtain the appropriate exception vector number during an “interrupt acknowledge cycle”.*
3. *Save the current processor state on the stack:*
 - Create an exception stack frame on the supervisor stack.*
 - Load the status register and program counter values into the stack frame.*
 - Depending on the exception, save some other relevant information in the stack frame.*
4. *Start the execution of the exception handling routine:*
 - Multiply the exception vector number times four to get the offset of the exception vector into the vector table.*
 - Go to the exception vector table, extract the exception vector, and load it into the program counter.*
 - If no other exception is pending, resume the instruction fetch-decode-execute cycle.*

Question 2.

- a) What makes the reset exception different from any other exception?
- b) Explain in your own words the reset sequence for the MC32 system.

The reset instruction does not cause a reset exception. It causes the CPU to drive the reset hardware signal active to reset all other microcomputer subsystems. The reset exception is caused by a hardware signal originating from outside the CPU. In the MC32 the signal can come from elsewhere in the chip. When the reset exception executes, no information is saved on the stacks. During the reset exception the following steps take place:

- The 'S' bit is set, the trace mode is disabled (all 'T' bits are cleared).
- The interrupt priority mask is set to binary 111.
- The vector base register is cleared (only in the case of the MC32).
- SP is loaded with vector 0 at address \$000.
- PC is loaded with vector 1 at address \$004.
- The first instruction of the reset routine is fetched.

The reset sequence for the MC32 follows the next steps:

- Initialize the submodules of the on-chip System Integration Module (SIM).
- Set the base address of the on-chip RAM. Enable on-chip RAM.
- Initialize the contents of the Exception Vector Table.
- Initialize the on-chip Queued Serial Module (QSM) and the Time Processing Unit (TPU).
- Initialize the off-chip peripheral interface chips.
- Initialize the operating system data structures.
- Initialize the User Stack Pointer (USP).
- Create the first exception handling frame on the supervisor stack. Load the starting user mode status register and program counter values.
- Execute a return from exception (RTE) instruction to start the execution of the first user mode program.

Question 3.

Explain in detail the interrupt handling sequence for the MC32.

1. An interrupt occurs with a specific priority level.
2. The CPU either finishes higher priority exception processing or reaches the end of the current instruction.
3. An internal copy of the status register is made. The 'S' bit is set. The trace mode is disabled by clearing all 'T' bits.
4. An interrupt acknowledge cycle is started by the CPU. At the end of the cycle, the CPU has obtained the exception vector number to be used.

-The MC32 echoes the corresponding interrupt priority level on the address bus. Depending on how the System Integration Module has been configured, the microcontroller may also assert an external chip select signal. Such signal can be used to drive an interrupt acknowledge signal to the Peripheral Interface Chip (PIC).

-The autovector (AVEC) signal is asserted low (either because it is connected permanently to GND, or because some internal or external decoder drove AVEC active), then the MC32 assumes that the autovectored method is to be used. In the later case, the interrupt priority level is used to determine the exception vector number, and the interrupt acknowledge (IACK) cycle end early.

If AVEC is inactive, then the microcontroller assumes that user interrupt method is to be used.

-The PIC receives the active chip select signal on its interrupt acknowledge input pin (IACK). The PIC receives the corresponding interrupt priority level over the address bus. The PIC uses this level to confirm which interrupt is about to be

handled. The PIC is capable of generating multiple interrupts at different levels, hence it needs to know which interrupt is being handled and acknowledge by the CPU.

-The PIC drives the appropriate exception vector number onto the data bus. This number is loaded into a register in the PIC by an initialization routine.

-The MC32 reads the exception vector number from the data bus. At this time the interrupt acknowledge cycle ends.

5. The address of the exception vector is computed from the exception vector number.

6. The program counter and the old status register are saved on the supervisor stack. Some additional information may also be saved depending on the CPU.

7. The exception vector is fetched from the exception vector table and loaded into the program counter of the CPU.

8. Normal instruction processing is resumed at the start of the exception handling routine.

Question 4.

Explain in your own words what the System Integration Module (SIM) is, what is its purpose, and how it interacts with the CPU32 in the MC32 microcontroller. Do some research in the literature (textbooks or internet) to backup your explanation. Mention your sources in a footnote.

The System Integration Module (SIM) consists of several submodules that allow the MC32 to interact with other devices or modules. Some important components of the SIM include the clock synthesizer, chip select, the external bus interface, and the system protection submodule^[1].

-The clock synthesizer generates the system clock signals using an external crystal oscillator or an external clock circuit. The clock synthesizer is also used to generate other clock signals for “watchdog” timers and a separate periodic interrupt timer on the chip. The periodic interrupt timer can be used to create a real-time clock that generates interrupts to the CPU at fixed time intervals.

-Chip selects and the external bus are used by to expand the system by adding circuits external to the microcontroller. A chip select enables a selected peripheral or a memory unit for data transfer. There are 12 chip select signals in the MC32. Five of these signals can be assigned as address signal lines on the external bus if these signals are not used as chip select signals.

-The external bus provides a 16-bit data path and up to 24 address signal lines to external devices. It also contains the controls signal lines for data transfers, interrupt requests, and other functions. The SIM signal lines are controlled by an executing program. Various signals can act as chip selects, as an external bus, or as input and output ports.

-The system protection submodule allows activity on the intermodule bus or the external bus to be monitored electrically. If an operation does not finish within a prespecified allotted time, an error is indicated to the CPU32. This hardware watchdog timer feature can monitor data transfers, interrupt acknowledgement

[1] Thomas L. Harman, *The Motorola MC68332 Microcontroller: Program Design, Assembly Language Programming, and Interfacing*, Prentice Hall, 1991.

cycles, and other types of bus activities. A software watchdog timer is also available to prevent a program from being trapped in a loop or otherwise exceeding the maximum time allotted to the program. If the program does not complete its execution in the predetermined time, the MC32 is reset.

Question 5.

In your own words explain what a kernel is in an operating system.

The kernel is the core of the operating system. Its responsibilities include managing the system's resources (the communication between hardware and software components). As a basic component of an operating system, a kernel provides the lowest-level abstraction layer for the resources (especially memory, processors and I/O devices) that application software must control to perform its function. The kernel provides the most basic services that are expected in a multitasking system. Some services include:

- processes and tasks creation, initiation, and termination.*
- processes and tasks suspension and unsuspending.*
- timer-controlled time slicing.*
- process priorities and pre-emption rules.*
- exception and interrupt handling mechanisms.*
- mechanisms for protecting critical sections.*
- Inter-process synchronization features.*

Question 6.

In your own words explain the two strategies for protecting critical sections (data structures or devices) in a multitasking environment.

There are two ways to protect critical sections:

1. Disable and re enable multitasking.

-Disabling multitasking:

-mask out all interrupts (disabling multitasking) at the start of the critical section.

-Restore interrupts (enable multitasking) at the end of the critical section.

The advantage of this approach is that is simple to implement.

The disadvantage is that all tasks in the system are affected and that it can become very disruptive in the multitasking environment.

2. Using semaphores.

-mask out interrupts, then check on the semaphore's data.

-If a flag in the semaphore is already set, then some other task is already in the critical section. Add the new task to a queue that are blocked on the semaphore. Switch to another ready-to-run task and restore interrupts.

-If a flag in the semaphore is cleared, then the data structure is available. Set the semaphore flag, restore interrupts, and proceed into the critical section. When exiting the critical section, mask out interrupts. If no task is blocked on the semaphore, then clear the flag in the semaphore. If there is a task blocked on the

semaphore, then unlock it by moving its Task Control Block (TCB) to the ready-to-run queue. Restore interrupts.

The advantage of this approach is that multitasking can continue while critical sections are being manipulated.

Question 7.

Mention three examples of preemptive multitasking operating systems.

Linux, Solaris, Windows XP, Mac OSX.