Question: 1. Which register number is used for the stack pointer (sp) in OS/161 and in which file did you find this information?

$29 is used for the stack pointer in OS/161 and this information can be found in *kern/arch/mips/include/kern/regdefs.h*

Question: 2. What bus/busses does OS/161 support?

OS/161 supports LAMEbus only. This information can be found in *kern/arch/sys161/include/bus.h*

Question: 3. What is the maximum number of CPUs that can be configured in SYS/161?

Maximum is 32. This information can be found in *kern/arch/sys161/include/maxcpus.h*

Question: 4. How many times per second is the kernel's hardclock() function invoked when executing a kernel compiled for ASST1? Hint: You may need some information from some of the files in kern/compile/ASST1 and kern/conf.

hardclock() is invoked for every 10000 times per second when executing a kernel compiled for ASST1.

Question: 5. How many times per second is the kernel's hardclock() function invoked when executing a kernel compiled for assignments other than ASST1?

hardclock() is invoked for every 100 times per second when executing a kernel compiled for assignments other than ASST1.

Question: 6. Explain how you can control whether or not OS/161 debugging statements are printed. When referring to files give the path name starting with kern.

Debugging statements (printing) can be toggled on and off by changing the dbflags variable either through editing source file or during runtime using the debugger.

Question: 7. Explain how you would add the ability to add and control your own new set of debugging statements (using DB_CATMOUSE).

In kern/include/lib.h add the line "#define DB_CATMOUSE 0xYYY" where YYY is a unique bit identifying this flag. Now whenever you need to debug a statement under this new flag, first make sure dbflags is set to DB_CATMOUSE or 0xYYY, then write the actual message (e.g. DEBUG(DB_CATMOUSE, "message")).

Question: 8. Give an example of using the OS/161 debugging statement to print "Hello World\n" in conjunction with DB_CATMOUSE

Recall the hello.c from assignment 0, we will now add:
"dbflags = DB_CATMOUSE;"
"DEBUG(DB_CATMOUSE, "Hello World\n");"

Question: 9. Describe how would you enable the debugging statements that use DB_CATMOUSE or DB_THREADS and only those debugging statements.

Use the bitwise-or operation and set dbflags = DB_CATMOUSE | DB_THREADS

Question: 10. Explain why you can use neither the debugging statements provided by OS/161 nor kprintf inside of lock acquire. in the later part of this assignment.

kprintf() calls lock_acquire. Thus having kprintf() or debugging statements inside a lock_acquire will create a deadlock.

Question: 11. Explain what a bitmap is and give an example of how and why it might be used.

A bitmap is a fixed-size array of bits and it is used for storage management. One can for example use it to emulate a series of switches and turn each one on/off.

Question: 12. What are the possible states that a thread can be in?
S_RUN, S_READY, S_SLEEP, and S_ZOMBIE

Question: 13. When do "zombie" threads finally get cleaned up?
They are finally cleaned up when exorcise() is called.

Question: 14. Which function is used to put a thread to sleep?
wchan_sleep(struct wchan *wc)

Question: 15. What is the purpose of the kernel's curthread global variable?
It is a pointer to the currently running thread on CPU.

Question: 16. : Explain what uw-locktest1 does.
uw-locktest1 tests if the lock is implemented correctly. Specifically, it initiates multiple threads executing add or subtract operations on a variable and checks if the final value is 0 (i.e. whether or not the operations are synchronized).

Question 17:
-59910, -28892, 14200, -6733, -6498

Question: 18. : Why is this test failing?
Multiple threads are accessing and editing the test_value in unordered fashion instead of the pre-determined order of operations. This happens because no actual implementation of mutual exclusion is in place (e.g. lock_acquire and lock_release).

Question 19:
-3248, 2271, 13613, 9390, 4383

Question: 20. : What happens to the final value of test value when you change the number of CPUs?
There are more positive/larger numbers than the last instances with the original CPU value, however these numbers are still randomized between –N and N (where N = the total from all the add ops) due to the absence of mutual exclusion.