- **Set #4** (Due Wed Nov 2)

1) Interrupt Flags
a) What happens if an interrupt service routine doesn't clear its interrupt flag?

The interrupt would get stuck in a loop since once it finished it would think there is another interrupt and cause it to go into the routine again.

b) Why <u>must</u> the clear of the interrupt flag be an atomic operation? Be complete -- list the two things that can happen and give a clear explanation of how each can occur if the flag clear is not atomic.

If the interrupt flag clears are not atomic you can have a higher priority interrupt occur during the execution of the Read instruction. This makes writing the flags image trigger a second entry into the higher priority ISR but there was only 1 interrupt. This only happens when the higher priority interrupt must occur during the instruction to read the flag.

If a lower priority interrupt occurs after the read instruction writing the flags image causes the lower priority flag to be overwritten and lost.

c) What PIC32 feature is used to make interrupt flag clears atomic (happen in one instruction)?

The I/O registers have a corresponding CLR(clear), SET(set), and INV(invert) register designed to provide atomic bit manipulations. So we would use the IFS0CLR register.

2) Epilogue
a) In general, what could go wrong with the SRSCtl, EPC or Status registers if there wasn't a `di` instruction prior to the restore of these registers in the epilogue and an interrupt occurred in this section of the code?

That section of the epilogue is considered a critical section. If an interrupt were to occur during this time we would not be able to properly restore the SRSCtL, EPC or Status registers. This is because in the prologue of an interrupt those are saved so we would be effectively overwriting the information we are trying to restore.

b) Assuming that

- the di instruction is not removed
- the `addiu sp,sp,...` instruction (which releases the stack space) follows the `mtc0 k1, Status` instruction in the epilog,
- and that the restored Status has its original IPL, but has IE set and EXL (mistakenly) cleared,

i) Why can there now be interrupts between the `mtc0 k0, Status` instruction, and prior to the `addiu sp,sp,...` instruction which releases the ISR's stack space?

Since the IE bit is 1 and EXL bit is now 0 it will not block any interrupts.

ii) How could this lead to stack overflows?

When we make room on the stack we only account for enough space for interrupts of a higher priority. But in this instance we can now be interrupted by an interrupt of lower priority which could take up valuable space we didn’t account for. If we run out of stack space, we will have an overflow.

c) Why can't we get stack overflows from interrupts in the body of the service routine?

Because we set up the stack to have enough space for the interrupts of higher priority so as long as the interrupts are higher priority we are prepared for that.

d) We ensure that there are no interrupts handled between the Status restore and the eret in the interrupt epilogue by making sure that the restored Status has its EXL bit set. Why couldn't we do this by instead of setting EXL, just leaving it clear, but also clearing the IE bit in the restored Status? (Note that eret clears the EXL bit when it runs.)

The EXL bit blocks interrupts from happening. By leaving the IE bit enabled and only blocking interrupts during critical portions using the EXL we insure that interrupts will be enabled when they need to be and the eret command clears the EXL bit for us so when we leave a ISR we know that interrupts will be enabled.

3) PRISS
a) In order to get the advantage of using a shadow set for a priority x interrupt, what must be done in addition to assigning a non-zero value to the PRIxSS field in the PRISS register?

Have the correct priority assigned to the interrupt in the IPCx, and ensure the IPLxSRS of the interrupt matches the priority of the PRIxSS.

b) Describe what can go wrong if a programmer accidentally assigns the same non-zero value to two different PRIxSS fields in the PRISS register, and how things go wrong it this happens.

One can overwrite the other shadow set which would let to an exception because two interrupts of different priorities would be trying to use the same registers.

c) Why is it okay to assign all the interrupts at one group priority to a given shadow set, even though the service routines for these interrupts would ideally not be saving and restoring registers?

Because if they are of the same priority level, they would not be interrupting each other so there is no risk of overwriting the registers before it was done.

4) SFR access
a) Why shouldn't SFRs be cached?

Because SFRs change regularly so it doesn't make sense to cache them.

b) Why should code running at user level privilege be prevented from accessing SFRs? Note that having code running at user level privilege usually implies that there is an operating system running at kernel level privilege, managing (possibly) several independent user level tasks. Also, for this question, assume that all the user level code is reliable and trustworthy.

Because SFR's are only used by Kernel level code. If the code at user level could access these registers they could interfere with the OS and cause failures.

5) Kseg0 and Kseg1 virtual addresses both map to the same physical addresses.
a) What is the difference between Kseg0 and Kseg1?

KSEG0 is chacheable and KSEG1 is not cacheable.

b) Why are the SFRs only found in Kseg1 and not in Kseg0?

Because it doesn't make sense to cache SFR's so you would want them to be in the non-cacheable KSEG1.

6) Memory Mapping
a) In what segment (Ksegn) and type (Program Flash or RAM) of memory would you expect to find each of the following virtual addresses:
i) `0x9d07073c`

`KSEG0 and Program Flash`

ii) `0xA0000344`

`KSEG1 and Data RAM`

b) To what physical address does each of the above addresses map?

`0x9D07073C = 0x1D07073C`

`0xA0000344 = 0x00000344`

c) Give a bitwise logical expression in C that can translate a KSEG0 or a KSEG1 address to its physical address.

Var = Virtualaddress & 0x1FFFFFFF

7) System Bus
a) Why does the DMA read function, DMA write function and CPU (read and write) functions each have two initiators?

One is for higher priority.

b) Under what bit settings and conditions is initiator id 2 used?

CFGCON bit 24