Q: Is it possible to have one IDCMP port for two windows on two screens?

A: Yes. The standard technique for sharing one IDCMP port is:

```
sharedport = CreatePort( ... );
     ...
newwin.IDCMPFlags = 0;
     ...
win = OpenWindow( &newwin );
win->UserPort = sharedport;
ModifyIDCMP( win, DESIRED_IDCMPFLAGS );
     ...
/* Open other windows the same way */
     ...
/* You must use CloseWindowSafely() to close
windows with shared IDCMP ports -- CloseWindow()
alone is unsafe... */
CloseWindowSafely( win );
```

Notice that this code uses the function CloseWindowSafely() (see the CloseWindow() *Autodoc* for the code). This function is specially designed to clean up after a window that shares an IDCMP port either with a window or some other inter-process communications customer. *Don't modify the loop in the CloseWindow() Autodoc! It is correct.* Some people think they see a bug there (one iteration too many), but they're wrong.

Q: How do I provide a Zorro II 24-bit DMA device with a few bytes of memory on any machine?

A: Call the AllocMem() function using the new (for 2.0) MEMF_24BITDMA flag defined in <exec/memory.h>. (It is much like MEMF_CHIP and MEMF_FAST only for 24BITDMA...)

Q: but what if I'm using a 1.3 system?

A: If you call AllocMem() with the MEMF_24BITDMA flag, the pre-2.0 AllocMem() will still check for that flag. You could write a utility that, under pre-2.0 systems, adds the MEMF_24BITDMA flag to a memory header's MH_ATTRIBUTES field of the 24-bit memory areas.

Q: With some programs I get a lot of enforcer *hits (write-long) at addresses $180, $184, $188, etc. with PC in ROM at location $F830C. Why?*

A: See the *Autodoc* for *exec.library/Alert*. This is where exec stores the processor registers when there is an alert caused by a processor trap. You should be able to make some guesses about what's wrong with your program by looking at the register values being written. The Guru Number is in D7.

Q: Commodore has publicly stated that applications should not *make specific use of the PMMU. Is this correct?*

A: Yes. While tools/hacks sometimes use the PMMU (such as *Enforcer*) they can not be expected to work when the system itself starts to use the PMMU. The PMMU is a supervisor-space feature of the CPU and, for it to do what it does, it must remain a system controlled resource. If you wish to do MMU-related things in an OS and processor compatible manner, use the function calls available in 2.04 (such as CachePreDMA(), CacheControlU(), etc). These will do their best to ``do the right thing'' on all OS/CPU combinations.