I/O Permission Bit Map

The I/O instructions that directly refer to addresses in the processor's I/O space are IN, INS, OUT, OUTS. The 80386 has the ability to selectively trap references to specific I/O addresses. The structure that enables selective trapping is the I/O Permission Bit Map in the TSS segment (see Figure 8-2). The I/O permission map is a bit vector. The size of the map and its location in the TSS segment are variable. The processor locates the I/O permission map by means of the I/O map base field in the fixed portion of the TSS. The I/O map base field is 16 bits wide and contains the offset of the beginning of the I/O permission map. The upper limit of the I/O permission map is the same as the limit of the TSS segment.

In protected mode, when it encounters an I/O instruction (IN, INS, OUT, or OUTS), the processor first checks whether CPL <= IOPL. If this condition is true, the I/O operation may proceed. If not true, the processor checks the I/O permission map. (In virtual 8086 mode, the processor consults the map without regard for IOPL . Refer to Chapter 15.)

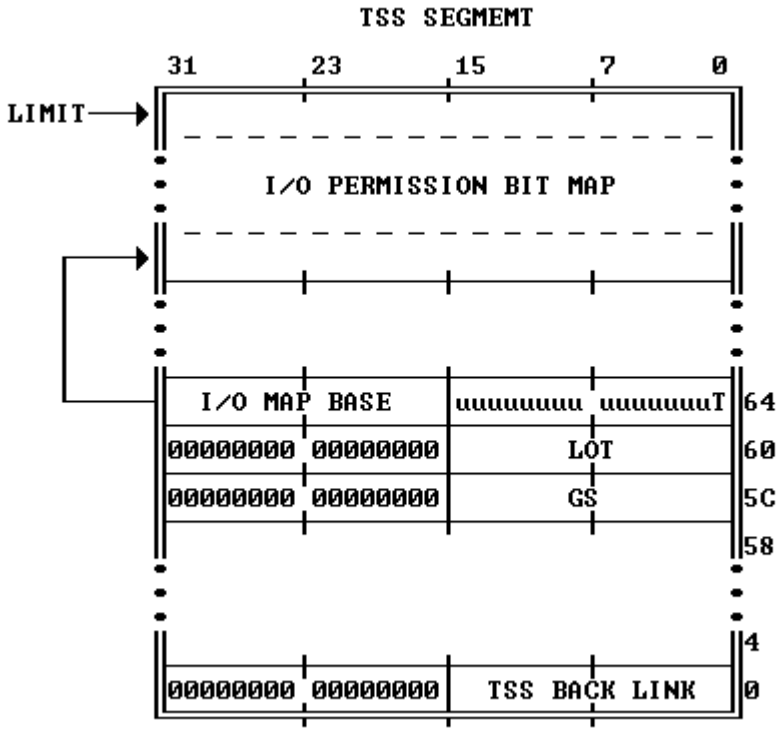
Each bit in the map corresponds to an I/O port byte address; for example, the bit for port 41 is found at I/O map base + 5, bit offset 1. The processor tests all the bits that correspond to the I/O addresses spanned by an I/O operation; for example, a doubleword operation tests four bits corresponding to four adjacent byte addresses. If any tested bit is set, the processor signals a general protection exception. If all the tested bits are zero, the I/O operation may proceed.

It is not necessary for the I/O permission map to represent all the I/O addresses. I/O addresses not spanned by the map are treated as if they had one bits in the map. For example, if TSS limit is equal to I/O map base + 31, the first 256 I/O ports are mapped; I/O operations on any port greater than 255 cause an exception.

If I/O map base is greater than or equal to TSS limit, the TSS segment has no I/O permission map, and all I/O instructions in the 80386 program cause exceptions when CPL > IOPL.

Because the I/O permission map is in the TSS segment, different tasks can have different maps. Thus, the operating system can allocate ports to a task by changing the I/O permission map in the task's TSS.

Figure 8-2. I/O Address Bit Map



How Multitasking Works (TSS)

The 386, as mentioned earlier, has support for *multitasking*, i.e. running several processes concurrently. In reality, however, they do not run concurrently. It only appears to the user as though they were all running at the same time.

The 386 uses the Task State Segments (TSSs) to support multitasking. The TSS descriptor points to a buffer which must be at least 104 bytes long. In addition to multitasking, TSSs can also be used for hardware interrupt handling (using task gates in the IDT). The TSS selector is neither readable nor writeable. Generally, a TSS alias is created, which is nothing but a "data type" segment pointing to the TSS buffer. TSS selectors *always* appear in the GDT, never in LDT or IDT. However, as mentioned above, task gates may appear in the IDT. The processor uses the TSS selector internally.

Since a multitasking switch requires the processor state to be saved, the buffer mostly contains the contents of the hardware registers. When a task switch occurs, the processor saves various details in the TSS buffer automatically. This process is very quick and hence not many CPU cycles are wasted in switching to a new task. Before a task is initially started, the operating system has to fill in certain entries in the TSS buffer.

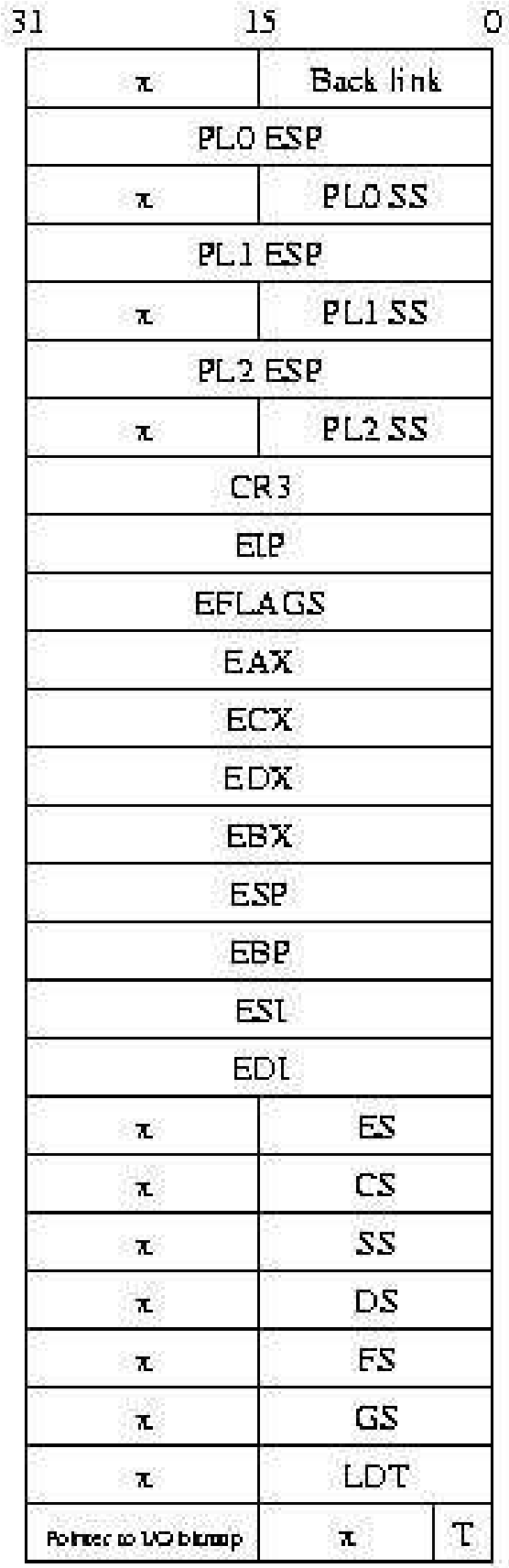
Notice the slots for stack segments and pointers for PL0, PL1 and PL2 near the top of the TSS. A program must maintain separate tasks for each privilege ring it may use. In particular the PL0 stack segments and pointers are a must. Otherwise, the processor may shut down due to a double fault in case of an interrupt.

A task switch may occur during a "far jump" or a "far call". The offset of the call or jump is simply ignored. The values in the TSS are loaded into the registers and the new task beings to execute. The selector referred by the jump or call must be a TSS selector or a task gate. The task gate contains the TSS selector. But unlike the TSS selector, the task gate may occur in the GDT, IDT or LDT. The EPL must always be less than or equal to the TSS's DPL as in case of the normal segment selectors.

The 386 also supports nested tasks. It handles nested tasks using the NT bit in the EFLAGS register. Whenever a task "calls" another task, the 386 stores the old task's TSS selector in the "back-link" field of the new TSS. Also, the NT bit in the EFLAGS register is set. When the new task wishes to return to the old one, it issues an IRET instruction[4].

The TSS aren't reentrant. Whenever a task is running, the 386 sets the BUSY bit in the TSS selector to indicate this. This is done to prevent recurive calling of tasks.

As mentioned earlier, the TSS must be at least 104 bytes long. The size of the TSS may extend to any size. The 386 contains a pointer to an I/O bitmap in the last field. The size of this bitmap is usually 8K, but may be lesser. The I/O bitmap is optional. If the size of the I/O bitmap is lesser, entries past the end of this bitmap are considered as "1s". Each bit in the I/O bitmap corresponds to one I/O port. If a task tries to do I/O, the 386 checks the task's CPL against the IOPL. If the CPL is less than or equal to the IOPL, access is granted, otherwise the 386 checks the I/O bitmap. If the bit corresponding to the I/O port used is zero, the 386 allows access, otherwise it denies access. A bitmap should always end with 0FFh.



Task state segment