1.2.3 Directives

The following directives are used for system debugging:

Directive	Function
db_system	Control a system
db_level	Set minimum Processor mask level

1.2.4 DB_SYSTEM

NAME

db_system -- "Control a System During Debug"

SYNOPSIS

uint db_system (cpu, mode)

```
uint cpu;     /* Designates a cpu in the system */
uint mode;    /* new mode */
```

DESCRIPTION

The *cpu* parameter uniquely identifies a cpu in the system.

The *mode* parameter indicates what processing may continue in the system after an exception occurs at some point within the system. Valid *mode* settings are:

DB_SYSTEM_CONTROL	to establish control over system
DB_SYSTEM_RELEASE	to remove control over system
DB_LEVEL	block tasking at level of ISR
DB_ALL	block all task dispatching
DB_CONTINUE	continue execution on the system

If an exception occurs while a task is executing, then that task is blocked and a message is sent to the debug task. If **DB_LEVEL** was specied as the mode, then only this task will be blocked. If **DB_ALL** was specified as the mode, then all dispatching will be suspended until a *db_system* command is specified with mode set to **DB_CONTINUE.**

If an exception occurs while an ISR is executing, further system activity is indicated by the mode parameter. If **DB_LEVEL** is specified for the *mode* parameter, then when an exception occurs in an ISR, the executive will issue a *db_level* directive with the level set to that of the current interrupt priority mask. This will keep the executive from dispatching tasks whose interrupt priority mask is less than this value, and will also block interrupts at this level or less. Interrupts and tasks whose level is greater will occur normally.

If the *mode* parameter is **DB_ALL** and an exception occurs within an ISR, then all further activity on this system will be blocked. The only exception to this is that remote requests for RTEID directives (including debug extensions) will be serviced by the executive. The executive will become unblocked when the debug task (remotly) issues a *db_unblock* for the *cpu_id* corresponding to the system. At this point, the ISR that caused the exception will continue execution.

Issuing a *db_system* directive with *mode* set to **DB_CONTINUE** will cause the execution of the system to continue.

RETURN VALUE

If *db_system* is successful, then 0 is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

Invalid *cpu*.

Invalid *mode*.

NOTES

When first establishing control over a system, the *mode* parameter must include **DB_SYSTEM_CONTROL** and may also include either **DB_ALL** or **DB_LEVEL**.

Once control has been established, the type of control may be changed by specifying a different mode.

1.2.5 DB_LEVEL

NAME

db_level -- "Set the Minimum Mask Level"

SYNOPSIS

uint db_level (level, &plevel)

```
uint level;     /* Minimum Processor Interrupt mask level*/
uint plevel;    /* Previous level - returned by this call */
```

DESCRIPTION

The *db_level* directive specifies a minimum interrupt priority mask level for further execution of the tasks and ISR's executing on the local cpu.

The *level* value is the minimum interrupt level for all tasks in the system. The executive will never set the status register's interrupt mask to a value less than *level*. Furthermore, the executive will never dispatch a task whose status register's interrupt mask is less than *level*.

RETURN VALUE

If *db_level* is successful, then the previous minimum level is returned in *plevel* and 0 is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

Level is not in a valid range (0..7).

The interrupt mask of the current task is less than *level*.

NOTES

May cause a preempt.

1.3 System Monitoring

Debugging a system involves more than debugging a collection of tasks; the performance of the entire system needs to be monitored and tuned. The *db_get_id* directive will return a unique identifier for items of particular types, or items in particular queues. The *db_get_item* directive will get information about items specified by the identifier. The information block will contain data about the system as well as some history (such as total number of calls to a directive) about the execution of the system. It is important to note that gathering statistics about the system will add a small amount of overhead to all of the calls.

The *db_get_id* directive requires an item_id as an input parameter. If the value of item_id is zero, then the first item of the specified class would be returned. If the item is non-zero, then the next item past the specified item_id will be returned. This can be used to loop through all items in a particular class. For example, to examine all tasks in the system, the following C code could be used:

```
for( item_id=0; item_id=get_item(item_id, TASK, 0); )
{
    process(item_id);
}
```

The class parameter specifies what type of item id to return and the third parameter is used to specify additional information (such as which message queue).

1.3.1 Directives

The directives provided by the system monitoring are:

Directive	Function
db_get_id	Get identifier for an item
db_get_item	Get information about an item