Invalid register number.

Task not created from local node.

NOTES

Can be called from within an ISR, except when the task was not created on the local node.

Will not cause a preempt.

1.1.15 DB\_SETREG

NAME

db\_setreg – “Set a task’s register”

SYNOPSIS

uint db\_setreg ( tid, regnum, &regptr )

uint tid; /\* task id as returned from t\_create ot t\_ident \*/

uint regnum; /\*register number \*/

union regval \*regptr; /\* pointer to register value \*/

union regval {  
 uint i;

float f;

}

The *regnum* field values are:

D\_REG0 Task’s Processor Register D0

D\_REG1 Task’s Processor Register D1

D\_REG2 Task’s Processor Register D2

D\_REG3 Task’s Processor Register D3

D\_REG4 Task’s Processor Register D4

D\_REG5 Task’s Processor Register D5

D\_REG6 Task’s Processor Register D6

D\_REG7 Task’s Processor Register D7

A\_REG0 Task’s Processor Register A0

A\_REG1 Task’s Processor Register A1

A\_REG2 Task’s Processor Register A2

A\_REG3 Task’s Processor Register A3

A\_REG4 Task’s Processor Register A4

A\_REG5 Task’s Processor Register A5

A\_REG6 Task’s Processor Register A6

A\_REG7 Task’s Processor Register A7

H\_SR Status Register

H\_PC Program Counter

H\_VOR Vector Offset Register

H\_USP User Stack Pointer

H\_ISP Interrupt Stack Pointer

H\_MSP Master Stack Pointer

H\_VBR Vector Base Register

H\_CACR Cache Control Register

H\_CAAR Cache Address Register

H\_VBR Vector Base Register

H\_CACR Cache Control Register

H\_CAAR Cache Address Register

FP\_REG0 Task’s Processor Register FP0

FP\_REG1 Task’s Processor Register FP1

FP\_REG2 Task’s Processor Register FP2

FP\_REG3 Task’s Processor Register FP3

FP\_REG4 Task’s Processor Register FP4

FP\_REG5 Task’s Processor Register FP5

FP\_REG6 Task’s Processor Register FP6

FP\_REG7 Task’s Processor Register FP7

FPCR Task’s Coprocessor Control Register

FPSR Task’s Coprocessor Status Register

FPIAR Task’s Coprocessor Instruction Address Register

DESCRIPTION

The executive sets the register identified in the *regnum* field for the task identified by the *tid* with the value in the *regptr* field.

The task identified in the *tid* field may exist on the local processor, or any remote processor in the multiprocessing configuration if the task was created with the GLOBAL flags value set (see *t\_create)*.

RETURN VALUE

If *db\_setreg* successfully set the register value, then 0 is returned.

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

ERROR CONDITIONS

Invalid *tid*.

Invalid register number.

Task not created from local node.

NOTES

Can be called from within an ISR, except when the task was not created on the local node.

Will not cause a preempt.

1.2 Debugging systems

Debugging a system is much more complex than debugging a task or collection of tasks. In order to debug a system, it should be possible to debug the interrupt service routines (ISR’s) which are part of the system. This causes several problems. The interrupt mask must not be lowered outside of an ISR. Additionally, an exception in an ISR may come at any time, and may occur when any task (with a low enough interrupt mask) is executing. Since the ISR must be blocked from further execution, the current task is also blocked.

1.2.1 Controlling Systems

The control over a system is established through the use of the *db\_system* directive. This will assert debug control over the entire system of tasks and ISR’s executing on that particular cpu board. In order to issue this command, the debugger must not be a task on the cpu board being debugged1.

When control is established, the type of control is specified by the *mode* parameter. If *all* is specified, then all activity, except for processing directives, is suspended when an exception occurs in an ISR. If *level* is specified, then the executive will block further dispatching at the current level and below (see the *db\_level* command) and continue dispatching tasks whose interrupt mask is greater than the current level.

1.2.2 Exceptions in ISR’s

When a controlled ISR issues an exception, such as a bus error, the execution of the entire system must be examined. Further activity of the ISR is suspended and further task dispatching on the system is performed based on the *mode* specified in the *db\_system* directive. The executive on the controlled system will format a message containing information about the exception and place it on a message queue associated with the debug of the cpu. Note that even if the execution of a system is blocked, the execution of the directives must still be processed. Since the execution of directive continues, the debug task may issue a *db\_remote* directive which will permit further execution of the controlled system.

\_\_\_\_\_\_\_\_\_\_\_\_

1. Alternatively, the debugger could be a “higher order” entity, such as the resident debug monitor, on a single cpu system. This “higher order” entity would perform as a system debugger and be able to issue requests to the executive as if it were a remote task.

1.2.3 Directives

The following directives are used for system debugging:

|  |  |
| --- | --- |
| Directive | Function |
| db\_system  db\_level | Control a system  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 specified 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 the 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 task 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 services by the executive. The executive will become unblocked when the debug task (remotely) 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  db\_get\_item | Get identifier for an item  Get information about an item |