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 or 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 debugged[1].

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 directives 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.