CPU_R4300

The CPU_R4300 module contains the machine-dependent plug-in for the MIPS R4300 family of processors. In addition to the standard core functions, the module also provide machine-dependent routines and support for machine-specific operations such as reading the floating point registers.

Module Options

CPU_KPRINTF_TRACES	Causes all trace-table entries generated by calls to the *rome_add_trace* routine to be displayed at the time they are entered into the table. This option is only useful for debugging problems during system initialisation as it otherwise generates a large volume of interrupt-disabled output.
CPU_ENABLE_DEBUG_CODE	Enables the code for the debugger and the disassembler. Activating this option is useful during development. Disabling this option can significantly reduce code size.
CPU_PW_DEBUG	If this symbol is defined,use of the system debugger is only possible by entering a ‘password’ at the prompt. The password is compiled in to the debugger source, so this is not much of a security measure, but it does offer some protection in the system.

Target File Definitions

The values required in the target file depend on the model of CPU on the board.

CPU_CACHED_PTR	A macro which converts a cached address into an uncached address referencing the same data area.
CPU_UNCACHED_PTR	A macro which converts an uncached address into a cached address referencing the same data area, or the identity mapping if this feature is not present on the machine (identity mapping on I960 machines).
CPU_PRIV_RAM_BASE	The base address of the private (main) RAM in the system.
CPU_RAMSIZE	The size of the available memory (in bytes) for the ROME system.

Data Definitions

cpu_plugin.h contains the following type definitions:

cpu_dep_mips_t	The data structure representing the machine-specific register accesses. It contains the 32 general purpose registers as well as the *hi*, *lo*, *imsk* and the *epc* register.
jmp_buf	The data structure used to hold an 'environment' for *setjmp* and *longjmp*. The *pfp* value contains the old stack pointer and the*rip* value contains the return address.

stdtypes.h contains definitions for the C standard **div_t** and **ldiv_t** types.

Module Operation

The CPU_R4300 module contains the initial entry of the ROME system at the head of the *link_first.s* assembler file. The routine clears bss storage, initializes the low-level interrupt handlers and calls the machine-independent *rome_start* routine.

The module also handles the first-level interrupt scheduling, dispatching interrupts to the handlers registered through the *rome_exception_handlers* array.

Shared Library Macros and Routines

Variable Arguments to routines

The *stdarg.h* file, which is copied from the *gcc* distribution, contains the macros for processing variable numbers of routine arguments: *va_alist*, *va_arg*, *va_dcl*, *va_end*.

I/O Accesses

The following macros provide cpu-dependent access to I/O space locations. These macros are provided for 'portable' drivers to make architecture-dependent access to locations where device registers may be placed. On the I386 machines, as there is a special I/O space, these macros generate calls to routines within the CPU module:

CPU_IOCLEAR*n*(_*a*, _*v*)	$n = 1, 2, 4$ clears the bits specified by _*v* in the *n*-byte wide IOSpace address _*a*.
CPU_IORD*n*(_*a*)	$n = 1, 2, 4$ returns the value of the *n*-byte wide location at IOSpace address _*a*.
CPU_IOSET*n*(_*a*, _*v*)	$n = 1, 2, 4$ sets the bits specified by _*v* in the *n*-byte wide IOSpace address _*a*.
CPU_IOWR*n*(_*a*, _*v*)	$n = 1, 2, 4$ sets the *n*-byte wide location at IOSpace address _*a* to the value _*v*.

Endianness

The following four macros are defined through the Target file to convert between network-endian and CPU-endian byte orderings.

uint *htonl*(
    uint *_dword*)
ushort *htons*(
    ushort *_word*)
uint *ntoh*l(
    uint *_dword*)
ushort *ntohs*(
    ushort *_word*)

As these macros may evaluate their arguments more than once, they should not be used with auto-incrementing arguments.

cpu_epilogue

void *cpu_epilogue*(**void**)

The *cpu_epilogue* performs any final initialisation of the processor environment before the scheduler is called. In this case, it does nothing except ensure that the *rome_this_ptr* variable contains a valid machine address.

cpu_longjump

void *cpu_longjump*(
    jmp_buf *env*,
    int *val*)

The *cpu_longjump* routine implements the standard *longjump* function, by causing a procedure return to the code location saved in the *env* buffer, with return code *val*.

cpu_prologue

void *cpu_prologue*(**void**)

The *cpu_prologue* routine performs C-level initialisation of the processor environment, by calling the *icu_setup_default_handlers* routine and setting the *cpu_freemem* variable to point to the end of the currently-used memory. It also turns off A20 emulation to prevent address wraparound at 1M, and turns off the floppy-drive motor which was left on after the boot completed.

cpu_scheduler

void *cpu_scheduler*(**void**)

The *cpu_scheduler* routine transfers control to the first process on the run queue. This routine is the exit point of the system initialisation procedure from which there is no return.

cpu_setjmp

int *cpu_setjmp*(
  jmp_buf *env*)

The *cpu_setjmp* routine implement the C standard *setjmp* function, creating a context in *env* for a subsequent call to *longjump*. The routine always returns 0. The *env* parameter is a pointer to a **struct _jmp_buf** data structure, which must remain in scope for the duration of the context.

cpu_setup_process

void *cpu_setup_process*(
  ROME_PROCESS **here*,
  ROME_INIT_PROC **proc*)

The *cpu_setup_process* routine initialises the machine-dependent information in the process structure *here* using the information supplied through the init module *proc* entry. This routine allocates the stack for the process and allocates memory for the *cpu_dep* field of the process control block. It then initializes a suitable set of values in that structure for a initial *eret* to enter the process.

cpu_suspend

void *cpu_suspend*(**void**)

The *cpu_suspend* routine saves the state of the currently-executing process and executes a context switch to the process at the head of the run queue. This routine is called explicitly during message processing by the machine-independent ROME code, and by the machine-dependent interrupt handler when an interrupt makes a higher-priority process runnable.

rome_add_trace

void *rome_add_trace*(
  ptr *a0*,
  int *type*,
  ptr *a2*)

The *rome_add_trace* routine adds a trace record to the circular trace buffer. The *type* parameter identifies the type of the trace record which determines how the two opaque parameters, *a0* and *a2* are to be interpreted.

rome_debug

void *rome_debug*(**void**)

The *rome_debug* routine enters the system-wide debugger. The following commands are supported in the I386 version of the debugger:

address *symbol*	print address of symbol
backtrace	trace process call stack
call *name*	call user-provided routine
continue	resume execution
cp *name*	change current process to *name*

di *addr len*	disassemble instructions
dm.[w|s|b] *addr len*	display memory [word, short or byte]
help	print this text
lp	list all processes
mem	memory-manager trace
message *addr*	format memory as a ROME message
pinfo	display info for current process
symbol *addr*	print symbol at address
symbols	print global symbol table
trace	display process trace log
wm.[w|s|b] *addr val*	write memory [word, short or byte]
[escape]	repeat last command

The *call* and *symbol* commands only work when a symbol table is present in the system.