CPU_I960

The CPU_I960 module contains the machine-dependent plug-in for the Intel 960 family of processors.

Module Options

CPU_FASTMOVE	Indicates that the plugin has provided machine-dependent 'fastmove' routines for data movement, specifically *cpu_memset*, and *cpu_memcpy*.
CPU_HAS_FLOAT	Enables the generation of code to support floating-point operations, for example the 'e' format effector in *printf*.
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_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_*xx*	The CPU model, 'xx' is currently one of CX, HX or JX.
CPU_BIG_ENDIAN	This symbol should be defined if the main RAM is configured in big-endian addressing mode, and be undefined otherwise.
CPU_CACHED_PTR	A macro which converts a cached address into an uncached 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_FREQ_REGISTER	The address of the memory-mapped register containing the CPU operating frequency.
CPU_IMAP*n*	Initial value of the Interrupt Map registers (*n* 0..2)

CPU_IMASK_ADDR	(Hx and Jx series) The address of the memory-mapped interrupt-mask register.
CPU_INIT_AC	Initial value for the Arithmetic Controls register
CPU_INIT_CACHE	Initial value for the Cache Control register
CPU_INIT_FC	Initial value for the Fault Controls register
CPU_INIT_RC	Initial value of the Register Cache size
CPU_INTERRUPTSTACK	The address of an area of main RAM to be used as the stack during interrupt handling.
CPU_IPND_ADDR	(Hx and Jx series) The address of the memory-mapped interrupt-pending register.
CPU_MANUAL_INTERRUPT	An interrupt vector representing the manual switch on a motherboard, used to force entry to the debugger on systems which support it.
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.
CPU_REGION*h*	Initial values for the region control words (*h* 0..F).
CPU_SUPERVISORSTACK	The address of an area in main RAM to be used as the stack during system initialisation,
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).

Data Definitions

cpu_plugin.h contains the following type definitions:

CPU_I960_REGISTERS	The data structure representing the machine-specific context information associated with each process. It contains the 16 global registers in the *gregs* array, the process' *pc*, *ac*, and *pfp* registers and the *imsk* value at the time of the context switch.
jmp_buf	The data structure used to hold an 'environment' for *setjmp* and *longjmp*. It contains the *pfp* and *rip* values to restore the stack.

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

Module Operation

The CPU_I960 module contains the initial entry of the ROME system at the head of the init.S assembler file. The routine clears the blank-storage of the system and executes a processor reset to select the processor control tables from ROME memory. It sets up a stack for the rest of the initialisation procedeure 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 *icu_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 I960 machines, as there is no special I/O space, these are indirections through suitably-cast **volatile** pointers:

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. In the usual case where the CPU is operating in little-endian mode, these macros are defined to byte-swap their arguments.

cpu_def_fault_handler

void *cpu_def_fault_handler*(**void**)

The *cpu_def_fault_handler* routine is connected to all the fault interrupts in the vector table. The routine passes the fault record and frame pointer into the internal machine-dependent fault handler for analysis.

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_pre_debug_int

void *cpu_pre_debug_int*(**void**)

The *cpu_pre_debug_int* routine calls the debugger from an unhandled interrupt.

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.

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. For I960 CPUs, this routine allocates the per-process **CPU_I960_REGISTERS** structure and places a pointer to it in the *cpu_dep* field of the proess structure.

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 I960 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
dlr	display local registers
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.