CPU_I386

The CPU_I386 module contains the machine-dependent plug-in for the Intel 80x86 family of processors of class 80386 and above (80486 and Pentium). In addition to the standard core functions, the module also provide machine-dependent 'fastmove' routines and support for machine-specific operations, for example to manipulate the special-purpose registers for memory cache control.

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_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 I386 machines).
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_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_VIDEO_MODE	The mode number to which the video controller should be set.
CPU_VIDEO_SET	A video mode-set option to the set the video mode to the desired mode.

Data Definitions

cpu_plugin.h contains the following type definitions:

CPU_MSRR	The data structure representing the machine-specific register accesses. The *eax* and *edx* fields contain the values of the two registers, either set or returned, by the write and read machine-specific register instructions respectively.
jmp_buf	The data structure used to hold an 'environment' for *setjmp* and *longjmp*. The *pfp* value contains the old stack pointer and the*ip* value contains the return address.

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

Module Operation

The CPU_I386 module contains the initial entry of the ROME system at the head of the *link_first.s* assembler file. The routine first discovers the graphics system video modes using a BIOS interrupt, and tries to set the current mode to the *CPU_VIDEO_MODE* value. It then clears low storage and switches into protected mode. It initialises the low-level interrupt handlers and sets up the (single) task for ROME operation. It then clears the blank-storage of the system and executes a task switch to 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 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*.

The low-level routines implementing I/O are:

uchar *cpu_inbyte*(
    int *address*)
uint *cpu_indword*(
    int *address*)
ushort *cpu_inword*
    int *address)*

which read a byte, int or short from an IOspace address, respectively, and:

void *cpu_outbyte*(
    int *address*,
    uchar *value*)
void *cpu_outdword*(
    int *address*,
    uint *value*)
void *cpu_outword*(
    int *address*,
    ushort *value*)

which write a byte, int or short to the IOspace address, respectively.

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, the 32-bit versions of these macros may use the assembler *BSWAP* instruction.

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_getcpuid

void *cpu_getcpuid*(
uint **buffer*)

The *cpu_getcpuid* routine returns the cpu information in the supplied buffer. The first four words contain the returned data from the *cpuid 0* instruction. The next word contains the returned data from the *cpuid 1* instruction, and the final four words contain the returned data from the *cpuid 2* instruction. These data are available for applications or devices that need to know the precise architecture on which they are running.

cpu_getcr

uint *cpu_getcr*(
int *crno*)

The *cpu_getcr* routine returns the value of control register *crno. crno* must be in the range 0..4.

cpu_getfd

uint *cpu_getfd*(**void**)

The *cpu_getfd* routine returns the flags register.

cpu_gettr

uint *cpu_gettr*(**void**)

The *cpu_gettr* routine returns the segment selector for the current task register.

cpu_interrupt_uh

void *cpu_interrupt_uh*(**void**)

The *cpu_interrupt_uh* routine is connected to all interrupts in the vector table as the default handler. Supported interrupts (i.e. faults and IRQs) may be overriden by software, but this routine acts as the unhandled-interrupt handler. The routine passes the interrupt number to the internal machine-dependent fault handler for analysis.

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_memcpy1, cpu_memcpy2, cpu_memcpy4

void *cpu_memcpy{1,2,4}*(
ptr *to*,
ptr *from*,
int *count*)

The *cpu_memcpy* routines copy *count* quantities (char, short or int as 1, 2, 4 respectively) from the memory area addressed by *from* to the memory area addressed by *to* using machine-dependent fast instructions.

cpu_memset4

void *cpu_memset4*(
ptr *to*,
int *value*,
int *wordcount*)

The *cpu_memset4* routine sets *wordcount* words (32bit quantities) at the memory area addressed by *to* to the value *value* using machine-dependent fast instructions.

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_readmsr

void *cpu_readmsr*(
int *regno*,
CPU_MSRR *data)

The *cpu_readmsr* routine returns the contents of model-specific register *regno* in the *eax* and *edx* fields of the supplied *data* structure, as set in the *%eax* and *%edx* registers from the *RDMSR* machine instruction.

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_setcr

void *cpu_setcr*(
int *crno*,
uint *value*)

The *cpu_setcr* routine sets control register *crno* to *value*. No checks are made as to whether the value is sensible or not. *crno* must be in the range 0..4.

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 I386 CPUs, this routine allocates the stack for the process and stores the current stach pointer in the *cpu_dep* field of the process control block. It then pushes a suitable set of values onto the stack for a initial *ret* 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.

cpu_writemsr

void *cpu_writemsr*(
int *regno*,
CPU_MSRR *data)

The *cpu_writemsr* routine sets the contents of model-specific register *regno* from the *eax* and *edx* fields of the supplied *data* structure, as passed in the *%eax* and *%edx* registers to the *WRMSR* machine instruction.

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.