```
**********************************************************
   65575 Sun Mar  2 12:33:06 2014
new/usr/src/uts/i86pc/io/apix/apix.c
4664 CPU->cpu_pri_data hasn't been used for years
**********************************************************
______unchanged_portion_omitted_

#if defined(__amd64)
static unsigned char dummy_cpu_pri[MAXIPL + 1] = {
        0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0
};
#endif

static void
apix_init()
{
        extern void (*do_interrupt_common)(struct regs *, trap_trace_rec_t *);

        APIC_VERBOSE(INIT, (CE_CONT, "apix: psm_softinit\n"));

        do_interrupt_common = apix_do_interrupt;
        addintr = apix_add_avintr;
        remintr = apix_rem_avintr;
        get_pending_spl = apix_get_pending_spl;
        get_intr_handler = apix_get_intr_handler;
        psm_get_localapicid = apic_get_localapicid;
        psm_get_ioapicid = apic_get_ioapicid;

        apix_softinit();

#if !defined(__amd64)
#if defined(__amd64)
        CPU->cpu_pri_data = dummy_cpu_pri;
#else
        if (cpuid_have_cr8access(CPU))
                apic_have_32bit_cr8 = 1;
#endif
#endif  /* __amd64 */

        /*
         * Initialize IRM pool parameters
         */
        if (irm_enable) {
                int     i;
                int     lowest_irq;
                int     highest_irq;

                /* number of CPUs present */
                apix_irminfo.apix_ncpus = apic_nproc;
                /* total number of entries in all of the IOAPICs present */
                lowest_irq = apic_io_vectbase[0];
                highest_irq = apic_io_vectend[0];
                for (i = 1; i < apic_io_max; i++) {
                        if (apic_io_vectbase[i] < lowest_irq)
                                lowest_irq = apic_io_vectbase[i];
                        if (apic_io_vectend[i] > highest_irq)
                                highest_irq = apic_io_vectend[i];
                }
                apix_irminfo.apix_ioapic_max_vectors =
                    highest_irq - lowest_irq + 1;
                /*
                 * Number of available per-CPU vectors excluding
                 * reserved vectors for Dtrace, int80, system-call,
                 * fast-trap, etc.
                 */
```



```
                apix_irminfo.apix_per_cpu_vectors = APIX_NAVINTR -
                    APIX_SW_RESERVED_VECTORS;

                /* Number of vectors (pre) allocated (SCI and HPET) */
                apix_irminfo.apix_vectors_allocated = 0;
                if (apic_hpet_vect != -1)
                        apix_irminfo.apix_vectors_allocated++;
                if (apic_sci_vect != -1)
                        apix_irminfo.apix_vectors_allocated++;
        }
}
______unchanged_portion_omitted_
```

```
**********************************************************
   35863 Sun Mar  2 12:33:06 2014
new/usr/src/uts/i86pc/io/pcplusmp/apic.c
4664 CPU->cpu_pri_data hasn't been used for years
**********************************************************
______unchanged_portion_omitted_
 137         /*
 138          * The ipl of an ISR at vector X is apic_vectortoipl[X>>4]
 139          * NOTE that this is vector as passed into intr_enter which is
 140          * programmed vector - 0x20 (APIC_BASE_VECT)
 141          */

 143 uchar_t apic_ipltopri[MAXIPL + 1];      /* unix ipl to apic pri */
 144         /* The taskpri to be programmed into apic to mask given ipl */

 146 #if defined(__amd64)
 147 static unsigned char dummy_cpu_pri[MAXIPL + 1];
 148 #endif

 146 /*
 147  * Correlation of the hardware vector to the IPL in use, initialized
 148  * from apic_vectortoipl[] in apic_init().  The final IPLs may not correlate
 149  * to the IPLs in apic_vectortoipl on some systems that share interrupt lines
 150  * connected to errata-stricken IOAPICs
 151  */
 152 uchar_t apic_ipls[APIC_AVAIL_VECTOR];

 154 /*
 155  * Patchable global variables.
 156  */
 157 int     apic_enable_hwsoftint = 0;      /* 0 - disable, 1 - enable      */
 158 int     apic_enable_bind_log = 1;       /* 1 - display interrupt binding log */

 160 /*
 161  *      Local static data
 162  */
 163 static struct   psm_ops apic_ops = {
 164         apic_probe,

 166         apic_init,
 167         apic_picinit,
 168         apic_intr_enter,
 169         apic_intr_exit,
 170         apic_setspl,
 171         apic_addspl,
 172         apic_delspl,
 173         apic_disable_intr,
 174         apic_enable_intr,
 175         (int (*)(int))NULL,             /* psm_softlvl_to_irq */
 176         (void (*)(int))NULL,            /* psm_set_softintr */

 178         apic_set_idlecpu,
 179         apic_unset_idlecpu,

 181         apic_clkinit,
 182         apic_getclkirq,
 183         (void (*)(void))NULL,           /* psm_hrtimeinit */
 184         apic_gethrtime,

 186         apic_get_next_processorid,
 187         apic_cpu_start,
 188         apic_post_cpu_start,
 189         apic_shutdown,
 190         apic_get_ipivect,
 191         apic_send_ipi,
```



```
 193         (int (*)(dev_info_t *, int))NULL,       /* psm_translate_irq */
 194         (void (*)(int, char *))NULL,    /* psm_notify_error */
 195         (void (*)(int))NULL,            /* psm_notify_func */
 196         apic_timer_reprogram,
 197         apic_timer_enable,
 198         apic_timer_disable,
 199         apic_post_cyclic_setup,
 200         apic_preshutdown,
 201         apic_intr_ops,                  /* Advanced DDI Interrupt framework */
 202         apic_state,                     /* save, restore apic state for S3 */
 203         apic_cpu_ops,                   /* CPU control interface. */
 204 };
______unchanged_portion_omitted_

 274 void
 275 apic_init(void)
 276 {
 277         int i;
 278         int     j = 1;

 280         psm_get_ioapicid = apic_get_ioapicid;
 281         psm_get_localapicid = apic_get_localapicid;
 282         psm_xlate_vector_by_irq = apic_xlate_vector_by_irq;

 284         apic_ipltopri[0] = APIC_VECTOR_PER_IPL; /* leave 0 for idle */
 285         for (i = 0; i < (APIC_AVAIL_VECTOR / APIC_VECTOR_PER_IPL); i++) {
 286                 if ((i < ((APIC_AVAIL_VECTOR / APIC_VECTOR_PER_IPL) - 1)) &&
 287                     (apic_vectortoipl[i + 1] == apic_vectortoipl[i]))
 288                         /* get to highest vector at the same ipl */
 289                         continue;
 290                 for (; j <= apic_vectortoipl[i]; j++) {
 291                         apic_ipltopri[j] = (i << APIC_IPL_SHIFT) +
 292                             APIC_BASE_VECT;
 293                 }
 294         }
 295         for (; j < MAXIPL + 1; j++)
 296                 /* fill up any empty ipltopri slots */
 297                 apic_ipltopri[j] = (i << APIC_IPL_SHIFT) + APIC_BASE_VECT;
 298         apic_init_common();

 300 #if !defined(__amd64)
 303 #if defined(__amd64)
 304         CPU->cpu_pri_data = dummy_cpu_pri;
 305 #else
 301         if (cpuid_have_cr8access(CPU))
 302                 apic_have_32bit_cr8 = 1;
 303 #endif
 308 #endif  /* __amd64 */
 304 }
______unchanged_portion_omitted_
```

```
**********************************************************
    7619 Sun Mar  2 12:33:07 2014
new/usr/src/uts/i86pc/ml/offsets.in
4664 CPU->cpu_pri_data hasn't been used for years
**********************************************************
```

```
\
\ Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
\ Copyright 2012 Garrett D'Amore <garrett@damore.org>.  All rights reserved.
\
\ CDDL HEADER START
\
\ The contents of this file are subject to the terms of the
\ Common Development and Distribution License (the "License").
\ You may not use this file except in compliance with the License.
\
\ You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
\ or http://www.opensolaris.org/os/licensing.
\ See the License for the specific language governing permissions
\ and limitations under the License.
\
\ When distributing Covered Code, include this CDDL HEADER in each
\ file and include the License file at usr/src/OPENSOLARIS.LICENSE.
\ If applicable, add the following below this CDDL HEADER, with the
\ fields enclosed by brackets "[]" replaced with your own identifying
\ information: Portions Copyright [yyyy] [name of copyright owner]
\
\ CDDL HEADER END
\



\
\ offsets.in: input file to produce assym.h using the ctfstabs program
\

#ifndef _GENASSYM
#define _GENASSYM
#endif

#define SIZES   1

#include <sys/types.h>
#include <sys/bootsvcs.h>
#include <sys/systm.h>
#include <sys/sysinfo.h>
#include <sys/user.h>
#include <sys/thread.h>
#include <sys/proc.h>
#include <sys/cpuvar.h>
#include <sys/tss.h>
#include <sys/privregs.h>
#include <sys/segments.h>
#include <sys/devops.h>
#include <sys/ddi_impldefs.h>
#include <vm/as.h>
#include <sys/avintr.h>
#include <sys/pic.h>
#include <sys/rm_platter.h>
#include <sys/stream.h>
#include <sys/strsubr.h>
#include <sys/sunddi.h>
#include <sys/modctl.h>
#include <sys/traptrace.h>
#include <sys/ontrap.h>
#include <sys/lgrp.h>
#include <sys/dtrace.h>
#include <sys/brand.h>
```



```
#include <sys/fastboot.h>
#include <sys/cpr_wakecode.h>

proc            PROCSIZE
        p_link
        p_next
        p_child
        p_sibling
        p_sig
        p_flag
        p_tlist
        p_as
        p_lockp
        p_user
        p_ldt
        p_ldt_desc
        p_model
        p_pctx
        p_agenttp
        p_zone
        p_brand
        p_brand_data

_kthread        THREAD_SIZE
        t_pcb                   T_LABEL
        t_lock
        t_lockstat
        t_lockp
        t_lock_flush
        t_kpri_req
        t_oldspl
        t_pri
        t_pil
        t_lwp
        t_procp
        t_link
        t_state
        t_mstate
        t_preempt_lk
        t_stk                   T_STACK
        t_swap
        t_lwpchan.lc_wchan      T_WCHAN
        t_flag                  T_FLAGS
        t_ctx
        t_lofault
        t_onfault
        t_ontrap
        t_cpu
        t_lpl
        t_bound_cpu
        t_intr
        t_forw
        t_back
        t_sig
        t_tid
        t_pre_sys
        t_preempt
        t_proc_flag
        t_startpc
        t_sysnum
        t_intr_start
        _tu._ts._t_astflag      T_ASTFLAG
        _tu._ts._t_post_sys     T_POST_SYS
        _tu._t_post_sys_ast     T_POST_SYS_AST
        t_copyops
```

```
ctxop
        save_op                 CTXOP_SAVE

as
        a_hat

user    USIZEBYTES
        u_comm
        u_signal

_label_t
        val     LABEL_VAL

\#define        LABEL_PC        LABEL_VAL
\#define        LABEL_SP        _CONST(LABEL_VAL + LABEL_VAL_INCR)
\#define        T_PC            _CONST(T_LABEL + LABEL_PC)
\#define        T_SP            _CONST(T_LABEL + LABEL_SP)

_klwp
        lwp_thread
        lwp_procp
        lwp_brand
        lwp_eosys
        lwp_regs
        lwp_arg
        lwp_ap
        lwp_cursig
        lwp_state
        lwp_mstate.ms_acct      LWP_MS_ACCT
        lwp_mstate.ms_prev      LWP_MS_PREV
        lwp_mstate.ms_start     LWP_MS_START
        lwp_mstate.ms_state_start LWP_MS_STATE_START
        lwp_pcb
        lwp_ru.sysc             LWP_RU_SYSC

\#define        LWP_ACCT_USER   _CONST(LWP_MS_ACCT + _MUL(LMS_USER, LWP_MS_ACCT_
\#define        LWP_ACCT_SYSTEM _CONST(LWP_MS_ACCT + _MUL(LMS_SYSTEM, LWP_MS_ACC

fpu_ctx
        fpu_regs                FPU_CTX_FPU_REGS
        fpu_flags               FPU_CTX_FPU_FLAGS
        fpu_xsave_mask          FPU_CTX_FPU_XSAVE_MASK

fxsave_state    FXSAVE_STATE_SIZE
        fx_fsw                  FXSAVE_STATE_FSW
        fx_mxcsr_mask           FXSAVE_STATE_MXCSR_MASK


autovec         AUTOVECSIZE
        av_vector
        av_intarg1
        av_intarg2
        av_ticksp
        av_link
        av_prilevel
        av_dip

av_head
        avh_link
        avh_hi_pri
        avh_lo_pri

cpu
        cpu_id
        cpu_flags
        cpu_self
```



```
        cpu_thread
        cpu_thread_lock
        cpu_kprunrun
        cpu_lwp
        cpu_fpowner
        cpu_idle_thread
        cpu_intr_thread
        cpu_intr_actv
        cpu_base_spl
        cpu_intr_stack
        cpu_stats.sys.cpumigrate        CPU_STATS_SYS_CPUMIGRATE
        cpu_stats.sys.intr              CPU_STATS_SYS_INTR
        cpu_stats.sys.intrblk           CPU_STATS_SYS_INTRBLK
        cpu_stats.sys.syscall           CPU_STATS_SYS_SYSCALL
        cpu_profile_pc
        cpu_profile_upc
        cpu_profile_pil
        cpu_ftrace.ftd_state            CPU_FTRACE_STATE
        cpu_mstate
        cpu_intracct

\#define        CPU_INTR_ACTV_REF       _CONST(CPU_INTR_ACTV + 2)

cpu
        cpu_m.pil_high_start    CPU_PIL_HIGH_START
        cpu_m.intrstat          CPU_INTRSTAT
        cpu_m.mcpu_current_hat  CPU_CURRENT_HAT
        cpu_m.mcpu_gdt          CPU_GDT
        cpu_m.mcpu_idt          CPU_IDT
        cpu_m.mcpu_tss          CPU_TSS
        cpu_m.mcpu_softinfo     CPU_SOFTINFO
        cpu_m.mcpu_pri          CPU_PRI
        cpu_m.mcpu_pri_data     CPU_PRI_DATA
#if defined(__xpv)
        cpu_m.mcpu_vcpu_info    CPU_VCPU_INFO
#endif

machcpu
        mcpu_pri_data   MCPU_PRI_DATA

standard_pic
        c_curmask
        c_iplmask

ddi_dma_impl
        dmai_rflags
        dmai_rdip

dev_info
        devi_ops                DEVI_DEV_OPS
        devi_bus_ctl
        devi_bus_dma_ctl
        devi_bus_dma_allochdl
        devi_bus_dma_freehdl
        devi_bus_dma_bindhdl
        devi_bus_dma_unbindhdl
        devi_bus_dma_flush
        devi_bus_dma_win

dev_ops
        devo_bus_ops            DEVI_BUS_OPS

bus_ops
        bus_ctl                 OPS_CTL
        bus_dma_map             OPS_MAP
        bus_dma_ctl             OPS_MCTL
```

```
256         bus_dma_allochdl        OPS_ALLOCHDL
257         bus_dma_freehdl         OPS_FREEHDL
258         bus_dma_bindhdl         OPS_BINDHDL
259         bus_dma_unbindhdl       OPS_UNBINDHDL
260         bus_dma_flush           OPS_FLUSH
261         bus_dma_win             OPS_WIN

263 sysent  SYSENT_SIZE     SYSENT_SIZE_SHIFT
264         sy_callc
265         sy_flags
266         sy_narg

268 stdata
269         sd_lock

271 queue
272         q_flag
273         q_next
274         q_stream
275         q_syncq
276         q_qinfo

278 qinit
279         qi_putp

281 syncq
282         sq_flags
283         sq_count
284         sq_lock
285         sq_wait

287 rm_platter
288         rm_idt_lim              IDTROFF
289         rm_gdt_lim              GDTROFF
290         rm_pdbr                 CR3OFF
291         rm_cpu                  CPUNOFF
292         rm_cr4                  CR4OFF
293         rm_cpu_halt_code        CPUHALTCODEOFF
294         rm_cpu_halted           CPUHALTEDOFF

296 ddi_acc_impl
297         ahi_acc_attr    ACC_ATTR
298         ahi_get8        ACC_GETB
299         ahi_get16       ACC_GETW
300         ahi_get32       ACC_GETL
301         ahi_get64       ACC_GETLL
302         ahi_put8        ACC_PUTB
303         ahi_put16       ACC_PUTW
304         ahi_put32       ACC_PUTL
305         ahi_put64       ACC_PUTLL
306         ahi_rep_get8    ACC_REP_GETB
307         ahi_rep_get16   ACC_REP_GETW
308         ahi_rep_get32   ACC_REP_GETL
309         ahi_rep_get64   ACC_REP_GETLL
310         ahi_rep_put8    ACC_REP_PUTB
311         ahi_rep_put16   ACC_REP_PUTW
312         ahi_rep_put32   ACC_REP_PUTL
313         ahi_rep_put64   ACC_REP_PUTLL

315 on_trap_data
316         ot_prot
317         ot_trap
318         ot_trampoline
319         ot_jmpbuf
320         ot_prev
321         ot_handle
```



```
322         ot_pad1

324 trap_trace_ctl_t        __TRAPTR_SIZE TRAPTR_SIZE_SHIFT
325         ttc_next        TRAPTR_NEXT
326         ttc_first       TRAPTR_FIRST
327         ttc_limit       TRAPTR_LIMIT

329 trap_trace_rec_t        TRAP_ENT_SIZE
330         ttr_cr2
331         ttr_info.idt_entry.vector       TTR_VECTOR
332         ttr_info.idt_entry.ipl          TTR_IPL
333         ttr_info.idt_entry.spl          TTR_SPL
334         ttr_info.idt_entry.pri          TTR_PRI
335         ttr_info.gate_entry.sysnum      TTR_SYSNUM
336         ttr_marker
337         ttr_stamp
338         ttr_curthread
339         ttr_sdepth
340         ttr_stack

342 lgrp_ld
343         lpl_lgrpid

345 dtrace_id_t     DTRACE_IDSIZE

347 cpu_core        CPU_CORE_SIZE   CPU_CORE_SHIFT
348         cpuc_dtrace_flags
349         cpuc_dtrace_illval

351 timespec        TIMESPEC_SIZE

353 gate_desc       GATE_DESC_SIZE

355 desctbr_t       DESCTBR_SIZE
356         dtr_limit
357         dtr_base

359 mod_stub_info   MODS_SIZE
360         mods_func_adr   MODS_INSTFCN
361         mods_errfcn     MODS_RETFCN
362         mods_flag

364 \#define        TRAP_TSIZE              _MUL(TRAP_ENT_SIZE, TRAPTR_NENT)

366 copyops
367         cp_copyin
368         cp_xcopyin
369         cp_copyout
370         cp_xcopyout
371         cp_copyinstr
372         cp_copyoutstr
373         cp_fuword8
374         cp_fuword16
375         cp_fuword32
376         cp_fuword64
377         cp_suword8
378         cp_suword16
379         cp_suword32
380         cp_suword64
381         cp_physio

383 brand
384         b_machops

386 brand_proc_data_t
387         spd_handler
```

```
fastboot_file_t
        fb_va
        fb_pte_list_va
        fb_pte_list_pa
        fb_dest_pa
        fb_size
        fb_next_pa
        fb_sections
        fb_sectcnt

fastboot_section_t
        fb_sec_offset
        fb_sec_paddr
        fb_sec_size
        fb_sec_bss_size

fastboot_info_t
        fi_files
        fi_has_pae
        fi_pagetable_va
        fi_pagetable_pa
        fi_last_table_pa
        fi_new_mbi_pa
        fi_valid

zone
        zone_brand_data

wc_cpu  WC_CPU_SIZE
        wc_retaddr
        wc_virtaddr
        wc_cr0
        wc_cr3
        wc_cr4
        wc_cr8
        wc_fs
        wc_fsbase
        wc_gs
        wc_gsbase
        wc_kgsbase
        wc_r8
        wc_r9
        wc_r10
        wc_r11
        wc_r12
        wc_r13
        wc_r14
        wc_r15
        wc_rax
        wc_rbp
        wc_rbx
        wc_rcx
        wc_rdi
        wc_rdx
        wc_rsi
        wc_rsp
        wc_gdt_limit    WC_GDT
        wc_gdt_base
        wc_idt_limit    WC_IDT
        wc_idt_base
        wc_tr
        wc_ldt
        wc_eflags
        wc_ebx
        wc_edi
```



```
        wc_esi
        wc_ebp
        wc_esp
        wc_esp
        wc_ss
        wc_cs
        wc_ds
        wc_es
        wc_cpu_id
        wc_saved_stack

wc_wakecode
        wc_cpu
```

```
**********************************************************
   12687 Sun Mar  2 12:33:07 2014
new/usr/src/uts/i86pc/os/mlsetup.c
4664 CPU->cpu_pri_data hasn't been used for years
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2012 Gary Mills
 *
 * Copyright (c) 1993, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2011 by Delphix. All rights reserved.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/disp.h>
#include <sys/promif.h>
#include <sys/clock.h>
#include <sys/cpuvar.h>
#include <sys/stack.h>
#include <vm/as.h>
#include <vm/hat.h>
#include <sys/reboot.h>
#include <sys/avintr.h>
#include <sys/vtrace.h>
#include <sys/proc.h>
#include <sys/thread.h>
#include <sys/cpupart.h>
#include <sys/pset.h>
#include <sys/copyops.h>
#include <sys/pg.h>
#include <sys/disp.h>
#include <sys/debug.h>
#include <sys/sunddi.h>
#include <sys/x86_archext.h>
#include <sys/privregs.h>
#include <sys/machsystm.h>
#include <sys/ontrap.h>
#include <sys/bootconf.h>
#include <sys/boot_console.h>
#include <sys/kdi_machimpl.h>
#include <sys/archsystm.h>
#include <sys/promif.h>
```



```
#include <sys/pci_cfgspace.h>
#ifdef __xpv
#include <sys/hypervisor.h>
#else
#include <sys/xpv_support.h>
#endif

/*
 * some globals for patching the result of cpuid
 * to solve problems w/ creative cpu vendors
 */

extern uint32_t cpuid_feature_ecx_include;
extern uint32_t cpuid_feature_ecx_exclude;
extern uint32_t cpuid_feature_edx_include;
extern uint32_t cpuid_feature_edx_exclude;

/*
 * Dummy spl priority masks
 */
static unsigned char dummy_cpu_pri[MAXIPL + 1] = {
        0xf, 0xf, 0xf, 0xf, 0xf, 0xf, 0xf, 0xf,
        0xf, 0xf, 0xf, 0xf, 0xf, 0xf, 0xf, 0xf, 0xf
};

/*
 * Set console mode
 */
static void
set_console_mode(uint8_t val)
{
        struct bop_regs rp = {0};

        rp.eax.byte.ah = 0x0;
        rp.eax.byte.al = val;
        rp.ebx.word.bx = 0x0;

        BOP_DOINT(bootops, 0x10, &rp);
}


/*
 * Setup routine called right before main(). Interposing this function
 * before main() allows us to call it in a machine-independent fashion.
 */
void
mlsetup(struct regs *rp)
{
        u_longlong_t prop_value;
        extern struct classfuncs sys_classfuncs;
        extern disp_t cpu0_disp;
        extern char t0stack[];
        extern int post_fastreboot;
        extern uint64_t plat_dr_options;

        ASSERT_STACK_ALIGNED();

        /*
         * initialize cpu_self
         */
        cpu[0]->cpu_self = cpu[0];

#if defined(__xpv)
        /*
         * Point at the hypervisor's virtual cpu structure
         */
```

```
120         cpu[0]->cpu_m.mcpu_vcpu_info = &HYPERVISOR_shared_info->vcpu_info[0];
121 #endif

123         /*
132          * Set up dummy cpu_pri_data values till psm spl code is
133          * installed.  This allows splx() to work on amd64.
134          */

136         cpu[0]->cpu_pri_data = dummy_cpu_pri;

138         /*
124          * check if we've got special bits to clear or set
125          * when checking cpu features
126          */

128         if (bootprop_getval("cpuid_feature_ecx_include", &prop_value) != 0)
129                 cpuid_feature_ecx_include = 0;
130         else
131                 cpuid_feature_ecx_include = (uint32_t)prop_value;

133         if (bootprop_getval("cpuid_feature_ecx_exclude", &prop_value) != 0)
134                 cpuid_feature_ecx_exclude = 0;
135         else
136                 cpuid_feature_ecx_exclude = (uint32_t)prop_value;

138         if (bootprop_getval("cpuid_feature_edx_include", &prop_value) != 0)
139                 cpuid_feature_edx_include = 0;
140         else
141                 cpuid_feature_edx_include = (uint32_t)prop_value;

143         if (bootprop_getval("cpuid_feature_edx_exclude", &prop_value) != 0)
144                 cpuid_feature_edx_exclude = 0;
145         else
146                 cpuid_feature_edx_exclude = (uint32_t)prop_value;

148         /*
149          * Initialize idt0, gdt0, ldt0_default, ktss0 and dftss.
150          */
151         init_desctbls();

153         /*
154          * lgrp_init() and possibly cpuid_pass1() need PCI config
155          * space access
156          */
157 #if defined(__xpv)
158         if (DOMAIN_IS_INITDOMAIN(xen_info))
159                 pci_cfgspace_init();
160 #else
161         pci_cfgspace_init();
162         /*
163          * Initialize the platform type from CPU 0 to ensure that
164          * determine_platform() is only ever called once.
165          */
166         determine_platform();
167 #endif

169         /*
170          * The first lightweight pass (pass0) through the cpuid data
171          * was done in locore before mlsetup was called.  Do the next
172          * pass in C code.
173          *
174          * The x86_featureset is initialized here based on the capabilities
175          * of the boot CPU.  Note that if we choose to support CPUs that have
176          * different feature sets (at which point we would almost certainly
177          * want to set the feature bits to correspond to the feature
178          * minimum) this value may be altered.
```



```
179          */
180         cpuid_pass1(cpu[0], x86_featureset);

182 #if !defined(__xpv)
183         if ((get_hwenv() & HW_XEN_HVM) != 0)
184                 xen_hvm_init();

186         /*
187          * Patch the tsc_read routine with appropriate set of instructions,
188          * depending on the processor family and architecure, to read the
189          * time-stamp counter while ensuring no out-of-order execution.
190          * Patch it while the kernel text is still writable.
191          *
192          * Note: tsc_read is not patched for intel processors whose family
193          * is >6 and for amd whose family >f (in case they don't support rdtscp
194          * instruction, unlikely). By default tsc_read will use cpuid for
195          * serialization in such cases. The following code needs to be
196          * revisited if intel processors of family >= f retains the
197          * instruction serialization nature of mfence instruction.
198          * Note: tsc_read is not patched for x86 processors which do
199          * not support "mfence". By default tsc_read will use cpuid for
200          * serialization in such cases.
201          *
202          * The Xen hypervisor does not correctly report whether rdtscp is
203          * supported or not, so we must assume that it is not.
204          */
205         if ((get_hwenv() & HW_XEN_HVM) == 0 &&
206             is_x86_feature(x86_featureset, X86FSET_TSCP))
207                 patch_tsc_read(X86_HAVE_TSCP);
208         else if (cpuid_getvendor(CPU) == X86_VENDOR_AMD &&
209             cpuid_getfamily(CPU) <= 0xf &&
210             is_x86_feature(x86_featureset, X86FSET_SSE2))
211                 patch_tsc_read(X86_TSC_MFENCE);
212         else if (cpuid_getvendor(CPU) == X86_VENDOR_Intel &&
213             cpuid_getfamily(CPU) <= 6 &&
214             is_x86_feature(x86_featureset, X86FSET_SSE2))
215                 patch_tsc_read(X86_TSC_LFENCE);

217 #endif  /* !__xpv */

219 #if defined(__i386) && !defined(__xpv)
220         /*
221          * Some i386 processors do not implement the rdtsc instruction,
222          * or at least they do not implement it correctly. Patch them to
223          * return 0.
224          */
225         if (!is_x86_feature(x86_featureset, X86FSET_TSC))
226                 patch_tsc_read(X86_NO_TSC);
227 #endif  /* __i386 && !__xpv */

229 #if defined(__amd64) && !defined(__xpv)
230         patch_memops(cpuid_getvendor(CPU));
231 #endif  /* __amd64 && !__xpv */

233 #if !defined(__xpv)
234         /* XXPV what, if anything, should be dorked with here under xen? */

236         /*
237          * While we're thinking about the TSC, let's set up %cr4 so that
238          * userland can issue rdtsc, and initialize the TSC_AUX value
239          * (the cpuid) for the rdtscp instruction on appropriately
240          * capable hardware.
241          */
242         if (is_x86_feature(x86_featureset, X86FSET_TSC))
243                 setcr4(getcr4() & ~CR4_TSD);
```

```
245         if (is_x86_feature(x86_featureset, X86FSET_TSCP))
246                 (void) wrmsr(MSR_AMD_TSCAUX, 0);
247
248         if (is_x86_feature(x86_featureset, X86FSET_DE))
249                 setcr4(getcr4() | CR4_DE);
250 #endif /* __xpv */
251
252         /*
253          * initialize t0
254          */
255         t0.t_stk = (caddr_t)rp - MINFRAME;
256         t0.t_stkbase = t0stack;
257         t0.t_pri = maxclsyspri - 3;
258         t0.t_schedflag = TS_LOAD | TS_DONT_SWAP;
259         t0.t_procp = &p0;
260         t0.t_plockp = &p0lock.pl_lock;
261         t0.t_lwp = &lwp0;
262         t0.t_forw = &t0;
263         t0.t_back = &t0;
264         t0.t_next = &t0;
265         t0.t_prev = &t0;
266         t0.t_cpu = cpu[0];
267         t0.t_disp_queue = &cpu0_disp;
268         t0.t_bind_cpu = PBIND_NONE;
269         t0.t_bind_pset = PS_NONE;
270         t0.t_bindflag = (uchar_t)default_binding_mode;
271         t0.t_cpupart = &cp_default;
272         t0.t_clfuncs = &sys_classfuncs.thread;
273         t0.t_copyops = NULL;
274         THREAD_ONPROC(&t0, CPU);
275
276         lwp0.lwp_thread = &t0;
277         lwp0.lwp_regs = (void *)rp;
278         lwp0.lwp_procp = &p0;
279         t0.t_tid = p0.p_lwpcnt = p0.p_lwprcnt = p0.p_lwpid = 1;
280
281         p0.p_exec = NULL;
282         p0.p_stat = SRUN;
283         p0.p_flag = SSYS;
284         p0.p_tlist = &t0;
285         p0.p_stksize = 2*PAGESIZE;
286         p0.p_stkpageszc = 0;
287         p0.p_as = &kas;
288         p0.p_lockp = &p0lock;
289         p0.p_brkpageszc = 0;
290         p0.p_t1_lgrpid = LGRP_NONE;
291         p0.p_tr_lgrpid = LGRP_NONE;
292         sigorset(&p0.p_ignore, &ignoredefault);
293
294         CPU->cpu_thread = &t0;
295         bzero(&cpu0_disp, sizeof (disp_t));
296         CPU->cpu_disp = &cpu0_disp;
297         CPU->cpu_disp->disp_cpu = CPU;
298         CPU->cpu_dispthread = &t0;
299         CPU->cpu_idle_thread = &t0;
300         CPU->cpu_flags = CPU_READY | CPU_RUNNING | CPU_EXISTS | CPU_ENABLE;
301         CPU->cpu_dispatch_pri = t0.t_pri;
302
303         CPU->cpu_id = 0;
304
305         CPU->cpu_pri = 12;              /* initial PIL for the boot CPU */
306
307         /*
308          * The kernel doesn't use LDTs unless a process explicitly requests one.
309          */
310         p0.p_ldt_desc = null_sdesc;
```



```
312         /*
313          * Initialize thread/cpu microstate accounting
314          */
315         init_mstate(&t0, LMS_SYSTEM);
316         init_cpu_mstate(CPU, CMS_SYSTEM);
317
318         /*
319          * Initialize lists of available and active CPUs.
320          */
321         cpu_list_init(CPU);
322
323         pg_cpu_bootstrap(CPU);
324
325         /*
326          * Now that we have taken over the GDT, IDT and have initialized
327          * active CPU list it's time to inform kmdb if present.
328          */
329         if (boothowto & RB_DEBUG)
330                 kdi_idt_sync();
331
332         /*
333          * Explicitly set console to text mode (0x3) if this is a boot
334          * post Fast Reboot, and the console is set to CONS_SCREEN_TEXT.
335          */
336         if (post_fastreboot && boot_console_type(NULL) == CONS_SCREEN_TEXT)
337                 set_console_mode(0x3);
338
339         /*
340          * If requested (boot -d) drop into kmdb.
341          *
342          * This must be done after cpu_list_init() on the 64-bit kernel
343          * since taking a trap requires that we re-compute gsbase based
344          * on the cpu list.
345          */
346         if (boothowto & RB_DEBUGENTER)
347                 kmdb_enter();
348
349         cpu_vm_data_init(CPU);
350
351         rp->r_fp = 0;   /* terminate kernel stack traces! */
352
353         prom_init("kernel", (void *)NULL);
354
355         /* User-set option overrides firmware value. */
356         if (bootprop_getval(PLAT_DR_OPTIONS_NAME, &prop_value) == 0) {
357                 plat_dr_options = (uint64_t)prop_value;
358         }
359 #if defined(__xpv)
360         /* No support of DR operations on xpv */
361         plat_dr_options = 0;
362 #else   /* __xpv */
363         /* Flag PLAT_DR_FEATURE_ENABLED should only be set by DR driver. */
364         plat_dr_options &= ~PLAT_DR_FEATURE_ENABLED;
365 #ifndef __amd64
366         /* Only enable CPU/memory DR on 64 bits kernel. */
367         plat_dr_options &= ~PLAT_DR_FEATURE_MEMORY;
368         plat_dr_options &= ~PLAT_DR_FEATURE_CPU;
369 #endif  /* __amd64 */
370 #endif  /* __xpv */
371
372         /*
373          * Get value of "plat_dr_physmax" boot option.
374          * It overrides values calculated from MSCT or SRAT table.
375          */
376         if (bootprop_getval(PLAT_DR_PHYSMAX_NAME, &prop_value) == 0) {
```

```
377                 plat_dr_physmax = ((uint64_t)prop_value) >> PAGESHIFT;
378         }

380         /* Get value of boot_ncpus. */
381         if (bootprop_getval(BOOT_NCPUS_NAME, &prop_value) != 0) {
382                 boot_ncpus = NCPU;
383         } else {
384                 boot_ncpus = (int)prop_value;
385                 if (boot_ncpus <= 0 || boot_ncpus > NCPU)
386                         boot_ncpus = NCPU;
387         }

389         /*
390          * Set max_ncpus and boot_max_ncpus to boot_ncpus if platform doesn't
391          * support CPU DR operations.
392          */
393         if (plat_dr_support_cpu() == 0) {
394                 max_ncpus = boot_max_ncpus = boot_ncpus;
395         } else {
396                 if (bootprop_getval(PLAT_MAX_NCPUS_NAME, &prop_value) != 0) {
397                         max_ncpus = NCPU;
398                 } else {
399                         max_ncpus = (int)prop_value;
400                         if (max_ncpus <= 0 || max_ncpus > NCPU) {
401                                 max_ncpus = NCPU;
402                         }
403                         if (boot_ncpus > max_ncpus) {
404                                 boot_ncpus = max_ncpus;
405                         }
406                 }

408                 if (bootprop_getval(BOOT_MAX_NCPUS_NAME, &prop_value) != 0) {
409                         boot_max_ncpus = boot_ncpus;
410                 } else {
411                         boot_max_ncpus = (int)prop_value;
412                         if (boot_max_ncpus <= 0 || boot_max_ncpus > NCPU) {
413                                 boot_max_ncpus = boot_ncpus;
414                         } else if (boot_max_ncpus > max_ncpus) {
415                                 boot_max_ncpus = max_ncpus;
416                         }
417                 }
418         }

420         /*
421          * Initialize the lgrp framework
422          */
423         lgrp_init(LGRP_INIT_STAGE1);

425         if (boothowto & RB_HALT) {
426                 prom_printf("unix: kernel halted by -h flag\n");
427                 prom_enter_mon();
428         }

430         ASSERT_STACK_ALIGNED();

432         /*
433          * Fill out cpu_ucode_info.  Update microcode if necessary.
434          */
435         ucode_check(CPU);

437         if (workaround_errata(CPU) != 0)
438                 panic("critical workaround(s) missing for boot cpu");
439 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   50258 Sun Mar  2 12:33:07 2014
new/usr/src/uts/i86pc/os/mp_startup.c
4664 CPU->cpu_pri_data hasn't been used for years
**********************************************************
______unchanged_portion_omitted_

 246 /*
 247  * Multiprocessor initialization.
 248  *
 249  * Allocate and initialize the cpu structure, TRAPTRACE buffer, and the
 250  * startup and idle threads for the specified CPU.
 251  * Parameter boot is true for boot time operations and is false for CPU
 252  * DR operations.
 253  */
 254 static struct cpu *
 255 mp_cpu_configure_common(int cpun, boolean_t boot)
 256 {
 257         struct cpu *cp;
 258         kthread_id_t tp;
 259         caddr_t sp;
 260         proc_t *procp;
 261 #if !defined(__xpv)
 262         extern int idle_cpu_prefer_mwait;
 263         extern void cpu_idle_mwait();
 264 #endif
 265         extern void idle();
 266         extern void cpu_idle();

 268 #ifdef TRAPTRACE
 269         trap_trace_ctl_t *ttc = &trap_trace_ctl[cpun];
 270 #endif

 272         ASSERT(MUTEX_HELD(&cpu_lock));
 273         ASSERT(cpun < NCPU && cpu[cpun] == NULL);

 275         if (cpu_free_list == NULL) {
 276                 cp = kmem_zalloc(sizeof (*cp), KM_SLEEP);
 277         } else {
 278                 cp = cpu_free_list;
 279                 cpu_free_list = cp->cpu_next_free;
 280         }

 282         cp->cpu_m.mcpu_istamp = cpun << 16;

 284         /* Create per CPU specific threads in the process p0. */
 285         procp = &p0;

 287         /*
 288          * Initialize the dispatcher first.
 289          */
 290         disp_cpu_init(cp);

 292         cpu_vm_data_init(cp);

 294         /*
 295          * Allocate and initialize the startup thread for this CPU.
 296          * Interrupt and process switch stacks get allocated later
 297          * when the CPU starts running.
 298          */
 299         tp = thread_create(NULL, 0, NULL, NULL, 0, procp,
 300             TS_STOPPED, maxclsyspri);

 302         /*
 303          * Set state to TS_ONPROC since this thread will start running
 304          * as soon as the CPU comes online.
```



```
 305          *
 306          * All the other fields of the thread structure are setup by
 307          * thread_create().
 308          */
 309         THREAD_ONPROC(tp, cp);
 310         tp->t_preempt = 1;
 311         tp->t_bound_cpu = cp;
 312         tp->t_affinitycnt = 1;
 313         tp->t_cpu = cp;
 314         tp->t_disp_queue = cp->cpu_disp;

 316         /*
 317          * Setup thread to start in mp_startup_common.
 318          */
 319         sp = tp->t_stk;
 320         tp->t_sp = (uintptr_t)(sp - MINFRAME);
 321 #if defined(__amd64)
 322         tp->t_sp -= STACK_ENTRY_ALIGN;          /* fake a call */
 323 #endif
 324         /*
 325          * Setup thread start entry point for boot or hotplug.
 326          */
 327         if (boot) {
 328                 tp->t_pc = (uintptr_t)mp_startup_boot;
 329         } else {
 330                 tp->t_pc = (uintptr_t)mp_startup_hotplug;
 331         }

 333         cp->cpu_id = cpun;
 334         cp->cpu_self = cp;
 335         cp->cpu_thread = tp;
 336         cp->cpu_lwp = NULL;
 337         cp->cpu_dispthread = tp;
 338         cp->cpu_dispatch_pri = DISP_PRIO(tp);

 340         /*
 341          * cpu_base_spl must be set explicitly here to prevent any blocking
 342          * operations in mp_startup_common from causing the spl of the cpu
 343          * to drop to 0 (allowing device interrupts before we're ready) in
 344          * resume().
 345          * cpu_base_spl MUST remain at LOCK_LEVEL until the cpu is CPU_READY.
 346          * As an extra bit of security on DEBUG kernels, this is enforced with
 347          * an assertion in mp_startup_common() -- before cpu_base_spl is set
 348          * to its proper value.
 349          */
 350         cp->cpu_base_spl = ipltospl(LOCK_LEVEL);

 352         /*
 353          * Now, initialize per-CPU idle thread for this CPU.
 354          */
 355         tp = thread_create(NULL, PAGESIZE, idle, NULL, 0, procp, TS_ONPROC, -1);

 357         cp->cpu_idle_thread = tp;

 359         tp->t_preempt = 1;
 360         tp->t_bound_cpu = cp;
 361         tp->t_affinitycnt = 1;
 362         tp->t_cpu = cp;
 363         tp->t_disp_queue = cp->cpu_disp;

 365         /*
 366          * Bootstrap the CPU's PG data
 367          */
 368         pg_cpu_bootstrap(cp);

 370         /*
```

```
371          * Perform CPC initialization on the new CPU.
372          */
373         kcpc_hw_init(cp);

375         /*
376          * Allocate virtual addresses for cpu_caddr1 and cpu_caddr2
377          * for each CPU.
378          */
379         setup_vaddr_for_ppcopy(cp);

381         /*
382          * Allocate page for new GDT and initialize from current GDT.
383          */
384 #if !defined(__lint)
385         ASSERT((sizeof (*cp->cpu_gdt) * NGDT) <= PAGESIZE);
386 #endif
387         cp->cpu_gdt = kmem_zalloc(PAGESIZE, KM_SLEEP);
388         bcopy(CPU->cpu_gdt, cp->cpu_gdt, (sizeof (*cp->cpu_gdt) * NGDT));

390 #if defined(__i386)
391         /*
392          * setup kernel %gs.
393          */
394         set_usegd(&cp->cpu_gdt[GDT_GS], cp, sizeof (struct cpu) -1, SDT_MEMRWA,
395             SEL_KPL, 0, 1);
396 #endif

398         /*
399          * If we have more than one node, each cpu gets a copy of IDT
400          * local to its node. If this is a Pentium box, we use cpu 0's
401          * IDT. cpu 0's IDT has been made read-only to workaround the
402          * cmpxchgl register bug
403          */
404         if (system_hardware.hd_nodes && x86_type != X86_TYPE_P5) {
405 #if !defined(__lint)
406                 ASSERT((sizeof (*CPU->cpu_idt) * NIDT) <= PAGESIZE);
407 #endif
408                 cp->cpu_idt = kmem_zalloc(PAGESIZE, KM_SLEEP);
409                 bcopy(CPU->cpu_idt, cp->cpu_idt, PAGESIZE);
410         } else {
411                 cp->cpu_idt = CPU->cpu_idt;
412         }

414         /*
415          * Get interrupt priority data from cpu 0.
416          */
417         cp->cpu_pri_data = CPU->cpu_pri_data;

419         /*
415          * alloc space for cpuid info
416          */
417         cpuid_alloc_space(cp);
418 #if !defined(__xpv)
419         if (is_x86_feature(x86_featureset, X86FSET_MWAIT) &&
420             idle_cpu_prefer_mwait) {
421                 cp->cpu_m.mcpu_mwait = cpuid_mwait_alloc(cp);
422                 cp->cpu_m.mcpu_idle_cpu = cpu_idle_mwait;
423         } else
424 #endif
425                 cp->cpu_m.mcpu_idle_cpu = cpu_idle;

427         init_cpu_info(cp);

429         /*
430          * alloc space for ucode_info
431          */
```



```
432         ucode_alloc_space(cp);
433         xc_init_cpu(cp);
434         hat_cpu_online(cp);

436 #ifdef TRAPTRACE
437         /*
438          * If this is a TRAPTRACE kernel, allocate TRAPTRACE buffers
439          */
440         ttc->ttc_first = (uintptr_t)kmem_zalloc(trap_trace_bufsize, KM_SLEEP);
441         ttc->ttc_next = ttc->ttc_first;
442         ttc->ttc_limit = ttc->ttc_first + trap_trace_bufsize;
443 #endif

445         /*
446          * Record that we have another CPU.
447          */
448         /*
449          * Initialize the interrupt threads for this CPU
450          */
451         cpu_intr_alloc(cp, NINTR_THREADS);

453         cp->cpu_flags = CPU_OFFLINE | CPU_QUIESCED | CPU_POWEROFF;
454         cpu_set_state(cp);

456         /*
457          * Add CPU to list of available CPUs.  It'll be on the active list
458          * after mp_startup_common().
459          */
460         cpu_add_unit(cp);

462         return (cp);
463 }
```

_____unchanged_portion_omitted_

```
**********************************************************
    5341 Sun Mar  2 12:33:07 2014
new/usr/src/uts/i86pc/sys/machcpuvar.h
4664 CPU->cpu_pri_data hasn't been used for years
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2011 Joyent, Inc. All rights reserved.
 */

#ifndef _SYS_MACHCPUVAR_H
#define _SYS_MACHCPUVAR_H

#ifdef __cplusplus
extern "C" {
#endif

#include <sys/inttypes.h>
#include <sys/x_call.h>
#include <sys/tss.h>
#include <sys/segments.h>
#include <sys/rm_platter.h>
#include <sys/avintr.h>
#include <sys/pte.h>

#ifndef _ASM
/*
 * On a virtualized platform a virtual cpu may not be actually
 * on a physical cpu, especially in situations where a configuration has
 * more vcpus than pcpus.  This function tells us (if it's able) if the
 * specified vcpu is currently running on a pcpu.  Note if it is not
 * known or not able to determine, it will return the unknown state.
 */
#define VCPU_STATE_UNKNOWN      0
#define VCPU_ON_PCPU            1
#define VCPU_NOT_ON_PCPU        2

extern int vcpu_on_pcpu(processorid_t);

/*
 * Machine specific fields of the cpu struct
 * defined in common/sys/cpuvar.h.
 *
```



```
 * Note:  This is kinda kludgy but seems to be the best
 * of our alternatives.
 */
typedef void *cpu_pri_lev_t;

struct cpuid_info;
struct cpu_ucode_info;
struct cmi_hdl;

/*
 * A note about the hypervisor affinity bits: a one bit in the affinity mask
 * means the corresponding event channel is allowed to be serviced
 * by this cpu.
 */
struct xen_evt_data {
        ulong_t         pending_sel[PIL_MAX + 1]; /* event array selectors */
        ulong_t         pending_evts[PIL_MAX + 1][sizeof (ulong_t) * 8];
        ulong_t         evt_affinity[sizeof (ulong_t) * 8]; /* service on cpu */
};

struct  machcpu {
        /*
         * x_call fields - used for interprocessor cross calls
         */
        struct xc_msg   *xc_msgbox;
        struct xc_msg   *xc_free;
        xc_data_t       xc_data;
        uint32_t        xc_wait_cnt;
        volatile uint32_t xc_work_cnt;

        int             mcpu_nodeid;            /* node-id */
        int             mcpu_pri;               /* CPU priority */
        cpu_pri_lev_t   mcpu_pri_data;          /* ptr to machine dependent */
                                                /* data for setting priority */
                                                /* level */

        struct hat      *mcpu_current_hat; /* cpu's current hat */

        struct hat_cpu_info     *mcpu_hat_info;

        volatile ulong_t        mcpu_tlb_info;

        /* i86 hardware table addresses that cannot be shared */

        user_desc_t     *mcpu_gdt;      /* GDT */
        gate_desc_t     *mcpu_idt;      /* current IDT */

        tss_t           *mcpu_tss;      /* TSS */

        kmutex_t        mcpu_ppaddr_mutex;
        caddr_t         mcpu_caddr1;    /* per cpu CADDR1 */
        caddr_t         mcpu_caddr2;    /* per cpu CADDR2 */
        uint64_t        mcpu_caddr1pte;
        uint64_t        mcpu_caddr2pte;

        struct softint  mcpu_softinfo;
        uint64_t        pil_high_start[HIGH_LEVELS];
        uint64_t        intrstat[PIL_MAX + 1][2];

        struct cpuid_info       *mcpu_cpi;

#if defined(__amd64)
        greg_t  mcpu_rtmp_rsp;          /* syscall: temporary %rsp stash */
        greg_t  mcpu_rtmp_r15;          /* syscall: temporary %r15 stash */
#endif
```

```
124         struct vcpu_info *mcpu_vcpu_info;
125         uint64_t        mcpu_gdtpa;     /* hypervisor: GDT physical address */

127         uint16_t mcpu_intr_pending;     /* hypervisor: pending intrpt levels */
128         uint16_t mcpu_ec_mbox;          /* hypervisor: evtchn_dev mailbox */
129         struct xen_evt_data *mcpu_evt_pend; /* hypervisor: pending events */

131         volatile uint32_t *mcpu_mwait;  /* MONITOR/MWAIT buffer */
132         void (*mcpu_idle_cpu)(void);    /* idle function */
133         uint16_t mcpu_idle_type;        /* CPU next idle type */
134         uint16_t max_cstates;           /* supported max cstates */

136         struct cpu_ucode_info   *mcpu_ucode_info;

138         void                    *mcpu_pm_mach_state;
139         struct cmi_hdl          *mcpu_cmi_hdl;
140         void                    *mcpu_mach_ctx_ptr;

142         /*
143          * A stamp that is unique per processor and changes
144          * whenever an interrupt happens. Userful for detecting
145          * if a section of code gets interrupted.
146          * The high order 16 bits will hold the cpu->cpu_id.
147          * The low order bits will be incremented on every interrupt.
148          */
149         volatile uint32_t       mcpu_istamp;
150 };

152 #define NINTR_THREADS   (LOCK_LEVEL-1)  /* number of interrupt threads */
153 #define MWAIT_HALTED    (1)             /* mcpu_mwait set when halting */
154 #define MWAIT_RUNNING   (0)             /* mcpu_mwait set to wakeup */
155 #define MWAIT_WAKEUP_IPI        (2)     /* need IPI to wakeup */
156 #define MWAIT_WAKEUP(cpu)       (*((cpu)->cpu_m.mcpu_mwait) = MWAIT_RUNNING)

158 #endif  /* _ASM */

160 /* Please DON'T add any more of this namespace-poisoning sewage here */

162 #define cpu_nodeid cpu_m.mcpu_nodeid
163 #define cpu_pri cpu_m.mcpu_pri
168 #define cpu_pri_data cpu_m.mcpu_pri_data
164 #define cpu_current_hat cpu_m.mcpu_current_hat
165 #define cpu_hat_info cpu_m.mcpu_hat_info
166 #define cpu_ppaddr_mutex cpu_m.mcpu_ppaddr_mutex
167 #define cpu_gdt cpu_m.mcpu_gdt
168 #define cpu_idt cpu_m.mcpu_idt
169 #define cpu_tss cpu_m.mcpu_tss
170 #define cpu_ldt cpu_m.mcpu_ldt
171 #define cpu_caddr1 cpu_m.mcpu_caddr1
172 #define cpu_caddr2 cpu_m.mcpu_caddr2
173 #define cpu_softinfo cpu_m.mcpu_softinfo
174 #define cpu_caddr1pte cpu_m.mcpu_caddr1pte
175 #define cpu_caddr2pte cpu_m.mcpu_caddr2pte

177 #ifdef  __cplusplus
178 }
```

_____***unchanged_portion_omitted_***