

```

*****
120665 Mon Apr 30 11:13:42 2012
new/usr/src/uts/i86pc/os/cpuid.c
2650 AMD family 0x15 PG support
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
23 * Copyright (c) 2011 by Delphix. All rights reserved.
24 */
25 /*
26 * Copyright (c) 2010, Intel Corporation.
27 * All rights reserved.
28 */
29 /*
30 * Portions Copyright 2009 Advanced Micro Devices, Inc.
31 */
32 /*
33 * Copyright (c) 2012, Joyent, Inc. All rights reserved.
34 */
35 /*
36 * Various routines to handle identification
37 * and classification of x86 processors.
38 */

40 #include <sys/types.h>
41 #include <sys/archsystem.h>
42 #include <sys/x86_archext.h>
43 #include <sys/kmem.h>
44 #include <sys/system.h>
45 #include <sys/cmn_err.h>
46 #include <sys/sunddi.h>
47 #include <sys/sunndi.h>
48 #include <sys/cpuvar.h>
49 #include <sys/processor.h>
50 #include <sys/sysmacros.h>
51 #include <sys/pg.h>
52 #include <sys/fp.h>
53 #include <sys/controlregs.h>
54 #include <sys/bitmap.h>
55 #include <sys/auxv_386.h>
56 #include <sys/memnode.h>
57 #include <sys/pci_cfgspace.h>

59 #ifdef __xpv
60 #include <sys/hypervisor.h>
61 #else

```

```

62 #include <sys/ontrap.h>
63 #endif

65 /*
66 * Pass 0 of cpuid feature analysis happens in locore. It contains special code
67 * to recognize Cyrix processors that are not cpuid-compliant, and to deal with
68 * them accordingly. For most modern processors, feature detection occurs here
69 * in pass 1.
70 *
71 * Pass 1 of cpuid feature analysis happens just at the beginning of mlsetup()
72 * for the boot CPU and does the basic analysis that the early kernel needs.
73 * x86_featureset is set based on the return value of cpuid_pass1() of the boot
74 * CPU.
75 *
76 * Pass 1 includes:
77 *
78 *   o Determining vendor/model/family/stepping and setting x86_type and
79 *     x86_vendor accordingly.
80 *   o Processing the feature flags returned by the cpuid instruction while
81 *     applying any workarounds or tricks for the specific processor.
82 *   o Mapping the feature flags into Solaris feature bits (X86_*).
83 *   o Processing extended feature flags if supported by the processor,
84 *     again while applying specific processor knowledge.
85 *   o Determining the CMT characteristics of the system.
86 *
87 * Pass 1 is done on non-boot CPUs during their initialization and the results
88 * are used only as a meager attempt at ensuring that all processors within the
89 * system support the same features.
90 *
91 * Pass 2 of cpuid feature analysis happens just at the beginning
92 * of startup(). It just copies in and corrects the remainder
93 * of the cpuid data we depend on: standard cpuid functions that we didn't
94 * need for pass1 feature analysis, and extended cpuid functions beyond the
95 * simple feature processing done in pass1.
96 *
97 * Pass 3 of cpuid analysis is invoked after basic kernel services; in
98 * particular kernel memory allocation has been made available. It creates a
99 * readable brand string based on the data collected in the first two passes.
100 *
101 * Pass 4 of cpuid analysis is invoked after post_startup() when all
102 * the support infrastructure for various hardware features has been
103 * initialized. It determines which processor features will be reported
104 * to userland via the aux vector.
105 *
106 * All passes are executed on all CPUs, but only the boot CPU determines what
107 * features the kernel will use.
108 *
109 * Much of the worst junk in this file is for the support of processors
110 * that didn't really implement the cpuid instruction properly.
111 *
112 * NOTE: The accessor functions (cpuid_get*) are aware of, and ASSERT upon,
113 * the pass numbers. Accordingly, changes to the pass code may require changes
114 * to the accessor code.
115 */

117 uint_t x86_vendor = X86_VENDOR_IntelClone;
118 uint_t x86_type = X86_TYPE_OTHER;
119 uint_t x86_clflush_size = 0;

121 uint_t pentiumpro_bug4046376;
122 uint_t pentiumpro_bug4064495;

124 uchar_t x86_featureset[BT_SIZEOFMAP(NUM_X86_FEATURES)];

126 static char *x86_feature_names[NUM_X86_FEATURES] = {
127     "lgpg",

```

```

128     "tsc",
129     "msr",
130     "mtrr",
131     "pge",
132     "de",
133     "cmov",
134     "mmx",
135     "mca",
136     "pae",
137     "cv8",
138     "pat",
139     "sep",
140     "sse",
141     "sse2",
142     "htt",
143     "asysc",
144     "nx",
145     "sse3",
146     "cx16",
147     "cmp",
148     "tscp",
149     "mwait",
150     "sse4a",
151     "cpuid",
152     "sse3",
153     "sse4_1",
154     "sse4_2",
155     "lgpg",
156     "clfsh",
157     "64",
158     "aes",
159     "pclmulqdq",
160     "xsave",
161     "avx",
162     "vmx",
163     "svm",
164     "topoext",
163     "svm"
165 };

```

unchanged portion omitted

```

266 /*
267 * These constants determine how many of the elements of the
268 * cpuid we cache in the cpuid_info data structure; the
269 * remaining elements are accessible via the cpuid instruction.
270 */

```

```

272 #define NMAX_CPI_STD      6          /* eax = 0 .. 5 */
273 #define NMAX_CPI_EXTD    0x1f      /* eax = 0x80000000 .. 0x8000001e */
272 #define NMAX_CPI_EXTD    0x1c      /* eax = 0x80000000 .. 0x8000001b */

```

```

275 /*
276 * Some terminology needs to be explained:
277 * - Socket: Something that can be plugged into a motherboard.
278 * - Package: Same as socket
279 * - Chip: Same as socket. Note that AMD's documentation uses term "chip"
280 * differently: there, chip is the same as processor node (below)
281 * - Processor node: Some AMD processors have more than one
282 * "subprocessor" embedded in a package. These subprocessors (nodes)
283 * are fully-functional processors themselves with cores, caches,
284 * memory controllers, PCI configuration spaces. They are connected
285 * inside the package with Hypertransport links. On single-node
286 * processors, processor node is equivalent to chip/socket/package.
287 * - Compute Unit: Some AMD processors pair cores in "compute units" that
288 * share the FPU and the I$ and L2 caches.

```

```

289 */

291 struct cpuid_info {
292     uint_t cpi_pass;          /* last pass completed */
293     /*
294      * standard function information
295      */
296     uint_t cpi_maxeax;       /* fn 0: %eax */
297     char cpi_vendorstr[13];  /* fn 0: %ebx:%ecx:%edx */
298     uint_t cpi_vendor;      /* enum of cpi_vendorstr */

300     uint_t cpi_family;      /* fn 1: extended family */
301     uint_t cpi_model;      /* fn 1: extended model */
302     uint_t cpi_step;       /* fn 1: stepping */
303     chipid_t cpi_chipid;   /* fn 1: %ebx: Intel: chip # */
304     /* AMD: package/socket # */
305     uint_t cpi_brandid;    /* fn 1: %ebx: brand ID */
306     int cpi_clogid;        /* fn 1: %ebx: thread # */
307     uint_t cpi_ncpu_per_chip; /* fn 1: %ebx: logical cpu count */
308     uint8_t cpi_cacheinfo[16]; /* fn 2: intel-style cache desc */
309     uint_t cpi_ncache;     /* fn 2: number of elements */
310     uint_t cpi_ncpu_shr_last_cache; /* fn 4: %eax: ncpus sharing cache */
311     id_t cpi_last_lvl_cacheid; /* fn 4: %eax: derived cache id */
312     uint_t cpi_std_4_size; /* fn 4: number of fn 4 elements */
313     struct cpuid_regs **cpi_std_4; /* fn 4: %ecx == 0 .. fn4_size */
314     struct cpuid_regs cpi_std[NMAX_CPI_STD]; /* 0 .. 5 */
315     /*
316      * extended function information
317      */
318     uint_t cpi_xmaxeax;     /* fn 0x80000000: %eax */
319     char cpi_brandstr[49]; /* fn 0x80000000[234] */
320     uint8_t cpi_pabits;    /* fn 0x80000006: %eax */
321     uint8_t cpi_vabits;   /* fn 0x80000006: %eax */
322     struct cpuid_regs cpi_extd[NMAX_CPI_EXTD]; /* 0x800000XX */

324     id_t cpi_coreid;      /* same coreid => strands share core */
325     int cpi_pkgcoreid;    /* core number within single package */
326     uint_t cpi_ncore_per_chip; /* AMD: fn 0x80000008: %ecx[7-0] */
327     /* Intel: fn 4: %eax[31-26] */

328     /*
329      * supported feature information
330      */
331     uint32_t cpi_support[5];
332     #define STD_EDX_FEATURES      0
333     #define AMD_EDX_FEATURES     1
334     #define TM_EDX_FEATURES      2
335     #define STD_ECX_FEATURES     3
336     #define AMD_ECX_FEATURES     4
337     /*
338      * Synthesized information, where known.
339      */
340     uint32_t cpi_chiprev;   /* See X86_CHIPREV_* in x86_archext.h */
341     const char *cpi_chiprevstr; /* May be NULL if chiprev unknown */
342     uint32_t cpi_socket;    /* Chip package/socket type */

344     struct mwait_info cpi_mwait; /* fn 5: monitor/mwait info */
345     uint32_t cpi_apicid;
346     uint_t cpi_procnodeid; /* AMD: nodeID on HT, Intel: chipid */
347     uint_t cpi_procnodes_per_pkg; /* AMD: # of nodes in the package */
348     /* Intel: 1 */
349     uint_t cpi_compunitid; /* AMD: ComputeUnit ID, Intel: coreid */
350     uint_t cpi_cores_per_compunit; /* AMD: # of cores in the ComputeUnit */

352     struct xsave_info cpi_xsave; /* fn D: xsave/xrestor info */
353 };

```

unchanged portion omitted

```

675 #endif /* __xpv */

677 static void
678 cpuid_intel_getids(cpu_t *cpu, void *feature)
679 {
680     uint_t i;
681     uint_t chipid_shift = 0;
682     uint_t coreid_shift = 0;
683     struct cpuid_info *cpi = cpu->cpu_m.mcpu_cpi;

685     for (i = 1; i < cpi->cpi_ncpu_per_chip; i <= 1)
686         chipid_shift++;

688     cpi->cpi_chipid = cpi->cpi_apicid >> chipid_shift;
689     cpi->cpi_clogid = cpi->cpi_apicid & ((1 << chipid_shift) - 1);

691     if (is_x86_feature(feature, X86FSET_CMP)) {
692         /*
693          * Multi-core (and possibly multi-threaded)
694          * processors.
695          */
696         uint_t ncpu_per_core;
697         if (cpi->cpi_ncore_per_chip == 1)
698             ncpu_per_core = cpi->cpi_ncpu_per_chip;
699         else if (cpi->cpi_ncore_per_chip > 1)
700             ncpu_per_core = cpi->cpi_ncpu_per_chip /
701                 cpi->cpi_ncore_per_chip;
702         /*
703          * 8bit APIC IDs on dual core Pentiums
704          * look like this:
705          *
706          * +-----+-----+-----+
707          * | Physical Package ID | MC | HT |
708          * +-----+-----+-----+
709          * <----- chipid ----->
710          * <----- coreid ----->
711          *
712          * <--- clogid --->
713          * <----->
714          *
715          * pkgcoreid
716          *
717          * Where the number of bits necessary to
718          * represent MC and HT fields together equals
719          * to the minimum number of bits necessary to
720          * store the value of cpi->cpi_ncpu_per_chip.
721          * Of those bits, the MC part uses the number
722          * of bits necessary to store the value of
723          * cpi->cpi_ncore_per_chip.
724          */
725         for (i = 1; i < ncpu_per_core; i <= 1)
726             coreid_shift++;
727         cpi->cpi_coreid = cpi->cpi_apicid >> coreid_shift;
728         cpi->cpi_pkgcoreid = cpi->cpi_clogid >> coreid_shift;
729     } else if (is_x86_feature(feature, X86FSET_HTTP)) {
730         /*
731          * Single-core multi-threaded processors.
732          */
733         cpi->cpi_coreid = cpi->cpi_chipid;
734         cpi->cpi_pkgcoreid = 0;
735     }
736     cpi->cpi_procnodeid = cpi->cpi_chipid;
737     cpi->cpi_compunitid = cpi->cpi_coreid;
738 }

738 static void
739 cpuid_amd_getids(cpu_t *cpu)

```

```

740 {
741     int i, first_half, coreidsz;
742     uint32_t nb_caps_reg;
743     uint_t node2_l1;
744     struct cpuid_info *cpi = cpu->cpu_m.mcpu_cpi;
745     struct cpuid_regs *cp;

747     /*
748      * AMD CMP chips currently have a single thread per core.
749      *
750      * Since no two cpus share a core we must assign a distinct coreid
751      * per cpu, and we do this by using the cpu_id. This scheme does not,
752      * however, guarantee that sibling cores of a chip will have sequential
753      * coreids starting at a multiple of the number of cores per chip -
754      * that is usually the case, but if the ACPI MADT table is presented
755      * in a different order then we need to perform a few more gymnastics
756      * for the pkgcoreid.
757      *
758      * All processors in the system have the same number of enabled
759      * cores. Cores within a processor are always numbered sequentially
760      * from 0 regardless of how many or which are disabled, and there
761      * is no way for operating system to discover the real core id when some
762      * are disabled.
763      *
764      * In family 0x15, the cores come in pairs called compute units. They
765      * share I$ and L2 caches and the FPU. Enumeration of this feature is
766      * simplified by the new topology extensions CPUID leaf, indicated by
767      * the X86 feature X86FSET_TOPOEXT.
768      */

770     cpi->cpi_coreid = cpu->cpu_id;
771     cpi->cpi_compunitid = cpu->cpu_id;

773     if (cpi->cpi_xmaxeax >= 0x80000008) {

775         coreidsz = BITX((cpi->cpi_extd[8].cp_ecx, 15, 12);

777         /*
778          * In AMD parlance chip is really a node while Solaris
779          * sees chip as equivalent to socket/package.
780          */
781         cpi->cpi_ncore_per_chip =
782             BITX((cpi->cpi_extd[8].cp_ecx, 7, 0) + 1;
783         if (coreidsz == 0) {
784             /* Use legacy method */
785             for (i = 1; i < cpi->cpi_ncore_per_chip; i <= 1)
786                 coreidsz++;
787             if (coreidsz == 0)
788                 coreidsz = 1;
789         }
790     } else {
791         /* Assume single-core part */
792         cpi->cpi_ncore_per_chip = 1;
793         coreidsz = 1;
794     }

796     cpi->cpi_clogid = cpi->cpi_pkgcoreid =
797         cpi->cpi_apicid & ((1 << coreidsz) - 1);
798     cpi->cpi_ncpu_per_chip = cpi->cpi_ncore_per_chip;

800     /* Get node ID, compute unit ID */
801     if (is_x86_feature(x86_featureset, X86FSET_TOPOEXT) &&
802         cpi->cpi_xmaxeax >= 0x8000001e) {
803         cp = &cpi->cpi_extd[0x1e];
804         cp->cp_eax = 0x8000001e;
805         (void) __cpuid_insn(cp);

```

```

807     cpi->cpi_procnodes_per_pkg = BITX(cp->cp_ecx, 10, 8) + 1;
808     cpi->cpi_procnodeid = BITX(cp->cp_ecx, 7, 0);
809     cpi->cpi_cores_per_compunit = BITX(cp->cp_ebx, 15, 8) + 1;
810     cpi->cpi_compunitid = BITX(cp->cp_ebx, 7, 0)
811         + (cpi->cpi_ncore_per_chip / cpi->cpi_cores_per_compunit)
812         * (cpi->cpi_procnodeid / cpi->cpi_procnodes_per_pkg);
813     } else if (cpi->cpi_family == 0xf || cpi->cpi_family >= 0x11) {
814         /* Get nodeID */
815         if (cpi->cpi_family == 0xf) {
816             cpi->cpi_procnodeid = (cpi->cpi_apicid >> coreidsz) & 7;
817             cpi->cpi_chipid = cpi->cpi_procnodeid;
818         } else if (cpi->cpi_family == 0x10) {
819             /*
820              * See if we are a multi-node processor.
821              * All processors in the system have the same number of nodes
822              */
823             nb_caps_reg = pci_getl_func(0, 24, 3, 0xe8);
824             if ((cpi->cpi_model < 8) || BITX(nb_caps_reg, 29, 29) == 0) {
825                 /* Single-node */
826                 cpi->cpi_procnodeid = BITX(cpi->cpi_apicid, 5,
827                     coreidsz);
828                 cpi->cpi_chipid = cpi->cpi_procnodeid;
829             } else {
830                 /*
831                  * Multi-node revision D (2 nodes per package
832                  * are supported)
833                  */
834                 cpi->cpi_procnodes_per_pkg = 2;
835
836                 first_half = (cpi->cpi_pkgcoreid <=
837                     (cpi->cpi_ncore_per_chip/2 - 1));
838
839                 if (cpi->cpi_apicid == cpi->cpi_pkgcoreid) {
840                     /* We are BSP */
841                     cpi->cpi_procnodeid = (first_half ? 0 : 1);
842                     cpi->cpi_chipid = cpi->cpi_procnodeid >> 1;
843                 } else {
844                     /* We are AP */
845                     /* Nodeid[2:1] bits to use for reading F3xe8 */
846                     node2_1 = BITX(cpi->cpi_apicid, 5, 4) << 1;
847
848                     nb_caps_reg =
849                         pci_getl_func(0, 24 + node2_1, 3, 0xe8);
850
851                     /*
852                      * Check IntNodeNum bit (31:30, but bit 31 is
853                      * always 0 on dual-node processors)
854                      */
855                     if (BITX(nb_caps_reg, 30, 30) == 0)
856                         cpi->cpi_procnodeid = node2_1 +
857                             !first_half;
858                     else
859                         cpi->cpi_procnodeid = node2_1 +
860                             first_half;
861
862                     cpi->cpi_chipid = cpi->cpi_procnodeid >> 1;
863                 }
864             } else if (cpi->cpi_family >= 0x11) {
865                 cpi->cpi_procnodeid = (cpi->cpi_apicid >> coreidsz) & 7;
866                 cpi->cpi_chipid = cpi->cpi_procnodeid;
867             } else {
868                 cpi->cpi_procnodeid = 0;

```

```

845     cpi->cpi_chipid = cpi->cpi_procnodeid;
862     }
864     cpi->cpi_chipid =
865         cpi->cpi_procnodeid / cpi->cpi_procnodes_per_pkg;
866     }
867     }
868     }
869     }
870     }
871     }
872     }
873     }
874     }
875     }
876     }
877     }
878     }
879     }
880     }
881     }
882     }
883     }
884     }
885     }
886     }
887     }
888     }
889     }
890     }
891     }
892     }
893     }
894     }
895     }
896     }
897     }
898     }
899     }
900     }
901     }
902     }
903     }
904     }
905     }
906     }
907     }
908     }
909     }
910     }
911     }
912     }
913     }
914     }
915     }
916     }
917     }
918     }
919     }
920     }
921     }
922     }
923     }
924     }
925     }
926     }
927     }
928     }
929     }
930     }
931     }
932     }
933     }
934     }
935     }
936     }
937     }
938     }
939     }
940     }
941     }
942     }
943     }
944     }
945     }

```

```

947  /*
948  * Beware: AMD uses "extended model" iff base *FAMILY* == 0xf.
949  * Intel, and presumably everyone else, uses model == 0xf, as
950  * one would expect (max value means possible overflow). Sigh.
951  */

953  switch (cpi->cpi_vendor) {
954  case X86_VENDOR_Intel:
955      if (IS_EXTENDED_MODEL_INTEL(cpi))
956          cpi->cpi_model += CPI_MODEL_XTD(cpi) << 4;
957      break;
958  case X86_VENDOR_AMD:
959      if (CPI_FAMILY(cpi) == 0xf)
960          cpi->cpi_model += CPI_MODEL_XTD(cpi) << 4;
961      break;
962  default:
963      if (cpi->cpi_model == 0xf)
964          cpi->cpi_model += CPI_MODEL_XTD(cpi) << 4;
965      break;
966  }

968  cpi->cpi_step = CPI_STEP(cpi);
969  cpi->cpi_brandid = CPI_BRANDID(cpi);

971  /*
972  * *default* assumptions:
973  * - believe %edx feature word
974  * - ignore %ecx feature word
975  * - 32-bit virtual and physical addressing
976  */
977  mask_edx = 0xffffffff;
978  mask_ecx = 0;

980  cpi->cpi_pabits = cpi->cpi_vabits = 32;

982  switch (cpi->cpi_vendor) {
983  case X86_VENDOR_Intel:
984      if (cpi->cpi_family == 5)
985          x86_type = X86_TYPE_P5;
986      else if (IS_LEGACY_P6(cpi)) {
987          x86_type = X86_TYPE_P6;
988          pentiumpro_bug4046376 = 1;
989          pentiumpro_bug4064495 = 1;
990          /*
991          * Clear the SEP bit when it was set erroneously
992          */
993          if (cpi->cpi_model < 3 && cpi->cpi_step < 3)
994              cp->cp_edx &= ~CPIID_INTC_EDX_SEP;
995      } else if (IS_NEW_F6(cpi) || cpi->cpi_family == 0xf) {
996          x86_type = X86_TYPE_P4;
997          /*
998          * We don't currently depend on any of the %ecx
999          * features until Prescott, so we'll only check
1000         * this from P4 onwards. We might want to revisit
1001         * that idea later.
1002         */
1003         mask_ecx = 0xffffffff;
1004     } else if (cpi->cpi_family > 0xf) {
1005         mask_ecx = 0xffffffff;
1006     }
1007     /*
1008     * We don't support MONITOR/MWAIT if leaf 5 is not available
1009     * to obtain the monitor linesize.
1010     */
1011     if (cpi->cpi_maxeax < 5)
1012         mask_ecx &= ~CPIID_INTC_ECX_MON;
1013     break;

```

```

1013     case X86_VENDOR_IntelClone:
1014     default:
1015         break;
1016     case X86_VENDOR_AMD:
1017     #if defined(OPTERON_ERRATUM_108)
1018         if (cpi->cpi_family == 0xf && cpi->cpi_model == 0xe) {
1019             cp->cp_edx = (0xf0f & cp->cp_edx) | 0xc0;
1020             cpi->cpi_model = 0xc;
1021         } else
1022     #endif
1023         if (cpi->cpi_family == 5) {
1024             /*
1025             * AMD K5 and K6
1026             *
1027             * These CPUs have an incomplete implementation
1028             * of MCA/MCE which we mask away.
1029             */
1030             mask_edx &= ~(CPIID_INTC_EDX_MCE | CPIID_INTC_EDX_MCA);

1032             /*
1033             * Model 0 uses the wrong (APIC) bit
1034             * to indicate PGE. Fix it here.
1035             */
1036             if (cpi->cpi_model == 0) {
1037                 if (cp->cp_edx & 0x200) {
1038                     cp->cp_edx &= ~0x200;
1039                     cp->cp_edx |= CPIID_INTC_EDX_PGE;
1040                 }
1041             }

1043             /*
1044             * Early models had problems w/ MMX; disable.
1045             */
1046             if (cpi->cpi_model < 6)
1047                 mask_edx &= ~CPIID_INTC_EDX_MMX;
1048         }

1050         /*
1051         * For newer families, SSE3 and CX16, at least, are valid;
1052         * enable all
1053         */
1054         if (cpi->cpi_family >= 0xf)
1055             mask_ecx = 0xffffffff;
1056         /*
1057         * We don't support MONITOR/MWAIT if leaf 5 is not available
1058         * to obtain the monitor linesize.
1059         */
1060         if (cpi->cpi_maxeax < 5)
1061             mask_ecx &= ~CPIID_INTC_ECX_MON;

1063     #if !defined(__xpv)
1064         /*
1065         * Do not use MONITOR/MWAIT to halt in the idle loop on any AMD
1066         * processors. AMD does not intend MWAIT to be used in the cpu
1067         * idle loop on current and future processors. 10h and future
1068         * AMD processors use more power in MWAIT than HLT.
1069         * Pre-family-10h Opterons do not have the MWAIT instruction.
1070         */
1071         idle_cpu_prefer_mwait = 0;
1072     #endif

1074     break;
1075     case X86_VENDOR_TM:
1076         /*
1077         * workaround the NT workaround in CMS 4.1
1078         */

```

```

1079     if (cpi->cpi_family == 5 && cpi->cpi_model == 4 &&
1080         (cpi->cpi_step == 2 || cpi->cpi_step == 3))
1081         cp->cp_edx |= CPUID_INTC_EDX_CX8;
1082     break;
1083 case X86_VENDOR_Centaur:
1084     /*
1085      * workaround the NT workarounds again
1086      */
1087     if (cpi->cpi_family == 6)
1088         cp->cp_edx |= CPUID_INTC_EDX_CX8;
1089     break;
1090 case X86_VENDOR_Cyrix:
1091     /*
1092      * We rely heavily on the probing in locore
1093      * to actually figure out what parts, if any,
1094      * of the Cyrix cpuid instruction to believe.
1095      */
1096     switch (x86_type) {
1097     case X86_TYPE_CYRIX_486:
1098         mask_edx = 0;
1099         break;
1100     case X86_TYPE_CYRIX_6x86:
1101         mask_edx = 0;
1102         break;
1103     case X86_TYPE_CYRIX_6x86L:
1104         mask_edx =
1105             CPUID_INTC_EDX_DE |
1106             CPUID_INTC_EDX_CX8;
1107         break;
1108     case X86_TYPE_CYRIX_6x86MX:
1109         mask_edx =
1110             CPUID_INTC_EDX_DE |
1111             CPUID_INTC_EDX_MSR |
1112             CPUID_INTC_EDX_CX8 |
1113             CPUID_INTC_EDX_PGE |
1114             CPUID_INTC_EDX_CMOV |
1115             CPUID_INTC_EDX_MMX;
1116         break;
1117     case X86_TYPE_CYRIX_GXm:
1118         mask_edx =
1119             CPUID_INTC_EDX_MSR |
1120             CPUID_INTC_EDX_CX8 |
1121             CPUID_INTC_EDX_CMOV |
1122             CPUID_INTC_EDX_MMX;
1123         break;
1124     case X86_TYPE_CYRIX_MediaGX:
1125         break;
1126     case X86_TYPE_CYRIX_MII:
1127     case X86_TYPE_VIA_CYRIX_III:
1128         mask_edx =
1129             CPUID_INTC_EDX_DE |
1130             CPUID_INTC_EDX_TSC |
1131             CPUID_INTC_EDX_MSR |
1132             CPUID_INTC_EDX_CX8 |
1133             CPUID_INTC_EDX_PGE |
1134             CPUID_INTC_EDX_CMOV |
1135             CPUID_INTC_EDX_MMX;
1136         break;
1137     default:
1138         break;
1139     }
1140     break;
1141 }
1143 #if defined(__xpv)
1144     /*

```

```

1145     * Do not support MONITOR/MWAIT under a hypervisor
1146     */
1147     mask_ecx &= ~CPUID_INTC_ECX_MON;
1148     /*
1149     * Do not support XSAVE under a hypervisor for now
1150     */
1151     xsave_force_disable = B_TRUE;
1153 #endif /* __xpv */
1155     if (xsave_force_disable) {
1156         mask_ecx &= ~CPUID_INTC_ECX_XSAVE;
1157         mask_ecx &= ~CPUID_INTC_ECX_AVX;
1158     }
1160     /*
1161     * Now we've figured out the masks that determine
1162     * which bits we choose to believe, apply the masks
1163     * to the feature words, then map the kernel's view
1164     * of these feature words into its feature word.
1165     */
1166     cp->cp_edx &= mask_edx;
1167     cp->cp_ecx &= mask_ecx;
1169     /*
1170     * apply any platform restrictions (we don't call this
1171     * immediately after __cpuid_insn here, because we need the
1172     * workarounds applied above first)
1173     */
1174     platform_cpuid_mangle(cpi->cpi_vendor, 1, cp);
1176     /*
1177     * fold in overrides from the "eeprom" mechanism
1178     */
1179     cp->cp_edx |= cpuid_feature_edx_include;
1180     cp->cp_edx &= ~cpuid_feature_edx_exclude;
1182     cp->cp_ecx |= cpuid_feature_ecx_include;
1183     cp->cp_ecx &= ~cpuid_feature_ecx_exclude;
1185     if (cp->cp_edx & CPUID_INTC_EDX_PSE) {
1186         add_x86_feature(featureset, X86FSET_LARGEPAGE);
1187     }
1188     if (cp->cp_edx & CPUID_INTC_EDX_TSC) {
1189         add_x86_feature(featureset, X86FSET_TSC);
1190     }
1191     if (cp->cp_edx & CPUID_INTC_EDX_MSR) {
1192         add_x86_feature(featureset, X86FSET_MSR);
1193     }
1194     if (cp->cp_edx & CPUID_INTC_EDX_MTRR) {
1195         add_x86_feature(featureset, X86FSET_MTRR);
1196     }
1197     if (cp->cp_edx & CPUID_INTC_EDX_PGE) {
1198         add_x86_feature(featureset, X86FSET_PGE);
1199     }
1200     if (cp->cp_edx & CPUID_INTC_EDX_CMOV) {
1201         add_x86_feature(featureset, X86FSET_CMOV);
1202     }
1203     if (cp->cp_edx & CPUID_INTC_EDX_MMX) {
1204         add_x86_feature(featureset, X86FSET_MMX);
1205     }
1206     if ((cp->cp_edx & CPUID_INTC_EDX_MCE) != 0 &&
1207         (cp->cp_edx & CPUID_INTC_EDX_MCA) != 0) {
1208         add_x86_feature(featureset, X86FSET_MCA);
1209     }
1210     if (cp->cp_edx & CPUID_INTC_EDX_PAE) {

```

```

1211         add_x86_feature(featureset, X86FSET_PAE);
1212     }
1213     if (cp->cp_edx & CPUID_INTC_EDX_CX8) {
1214         add_x86_feature(featureset, X86FSET_CX8);
1215     }
1216     if (cp->cp_ecx & CPUID_INTC_ECX_CX16) {
1217         add_x86_feature(featureset, X86FSET_CX16);
1218     }
1219     if (cp->cp_edx & CPUID_INTC_EDX_PAT) {
1220         add_x86_feature(featureset, X86FSET_PAT);
1221     }
1222     if (cp->cp_edx & CPUID_INTC_EDX_SEP) {
1223         add_x86_feature(featureset, X86FSET_SEP);
1224     }
1225     if (cp->cp_edx & CPUID_INTC_EDX_FXSR) {
1226         /*
1227          * In our implementation, fxsave/fxrstor
1228          * are prerequisites before we'll even
1229          * try and do SSE things.
1230          */
1231         if (cp->cp_edx & CPUID_INTC_EDX_SSE) {
1232             add_x86_feature(featureset, X86FSET_SSE);
1233         }
1234         if (cp->cp_edx & CPUID_INTC_EDX_SSE2) {
1235             add_x86_feature(featureset, X86FSET_SSE2);
1236         }
1237         if (cp->cp_ecx & CPUID_INTC_ECX_SSE3) {
1238             add_x86_feature(featureset, X86FSET_SSE3);
1239         }
1240         if (cp->cp_ecx & CPUID_INTC_ECX_SSSE3) {
1241             add_x86_feature(featureset, X86FSET_SSSE3);
1242         }
1243         if (cp->cp_ecx & CPUID_INTC_ECX_SSE4_1) {
1244             add_x86_feature(featureset, X86FSET_SSE4_1);
1245         }
1246         if (cp->cp_ecx & CPUID_INTC_ECX_SSE4_2) {
1247             add_x86_feature(featureset, X86FSET_SSE4_2);
1248         }
1249         if (cp->cp_ecx & CPUID_INTC_ECX_AES) {
1250             add_x86_feature(featureset, X86FSET_AES);
1251         }
1252         if (cp->cp_ecx & CPUID_INTC_ECX_PCLMULQDQ) {
1253             add_x86_feature(featureset, X86FSET_PCLMULQDQ);
1254         }
1255     }
1256     if (cp->cp_ecx & CPUID_INTC_ECX_XSAVE) {
1257         add_x86_feature(featureset, X86FSET_XSAVE);
1258         /* We only test AVX when there is XSAVE */
1259         if (cp->cp_ecx & CPUID_INTC_ECX_AVX) {
1260             add_x86_feature(featureset,
1261                 X86FSET_AVX);
1262         }
1263     }
1264 }
1265 if (cp->cp_edx & CPUID_INTC_EDX_DE) {
1266     add_x86_feature(featureset, X86FSET_DE);
1267 }
1268 #if !defined(__xpv)
1269     if (cp->cp_ecx & CPUID_INTC_ECX_MON) {
1270         /*
1271          * We require the CLFLUSH instruction for erratum workaround
1272          * to use MONITOR/MWAIT.
1273          */
1274         if (cp->cp_edx & CPUID_INTC_EDX_CLFSH) {
1275             cpi->cpi_mwait.support |= MWAIT_SUPPORT;

```

```

1277         add_x86_feature(featureset, X86FSET_MWAIT);
1278     } else {
1279         extern int idle_cpu_assert_cflush_monitor;
1280
1281         /*
1282          * All processors we are aware of which have
1283          * MONITOR/MWAIT also have CLFLUSH.
1284          */
1285         if (idle_cpu_assert_cflush_monitor) {
1286             ASSERT((cp->cp_ecx & CPUID_INTC_ECX_MON) &&
1287                 (cp->cp_edx & CPUID_INTC_EDX_CLFSH));
1288         }
1289     }
1290 }
1291 #endif /* __xpv */
1292
1293 if (cp->cp_ecx & CPUID_INTC_ECX_VMX) {
1294     add_x86_feature(featureset, X86FSET_VMX);
1295 }
1296
1297 /*
1298 * Only need it first time, rest of the cpus would follow suit.
1299 * we only capture this for the bootcpu.
1300 */
1301 if (cp->cp_edx & CPUID_INTC_EDX_CLFSH) {
1302     add_x86_feature(featureset, X86FSET_CLFSH);
1303     x86_clflush_size = (BITX(cp->cp_ebx, 15, 8) * 8);
1304 }
1305 if (is_x86_feature(featureset, X86FSET_PAE))
1306     cpi->cpi_pabits = 36;
1307
1308 /*
1309 * Hyperthreading configuration is slightly tricky on Intel
1310 * and pure clones, and even trickier on AMD.
1311 *
1312 * (AMD chose to set the HTT bit on their CMP processors,
1313 * even though they're not actually hyperthreaded. Thus it
1314 * takes a bit more work to figure out what's really going
1315 * on ... see the handling of the CMP_LGCY bit below)
1316 */
1317 if (cp->cp_edx & CPUID_INTC_EDX_HTT) {
1318     cpi->cpi_ncpu_per_chip = CPI_CPU_COUNT(cpi);
1319     if (cpi->cpi_ncpu_per_chip > 1)
1320         add_x86_feature(featureset, X86FSET_HTT);
1321 } else {
1322     cpi->cpi_ncpu_per_chip = 1;
1323 }
1324
1325 /*
1326 * Work on the "extended" feature information, doing
1327 * some basic initialization for cpuid_pass2()
1328 */
1329 xcpuid = 0;
1330 switch (cpi->cpi_vendor) {
1331 case X86_VENDOR_Intel:
1332     if (IS_NEW_F6(cpi) || cpi->cpi_family >= 0xf)
1333         xcpuid++;
1334     break;
1335 case X86_VENDOR_AMD:
1336     if (cpi->cpi_family > 5 ||
1337         (cpi->cpi_family == 5 && cpi->cpi_model >= 1))
1338         xcpuid++;
1339     break;
1340 case X86_VENDOR_Cyrix:
1341     /*
1342      * Only these Cyrix CPUs are -known- to support

```

```

1343     * extended cpuid operations.
1344     */
1345     if (x86_type == X86_TYPE_VIA_CYRIX_III ||
1346         x86_type == X86_TYPE_CYRIX_GXm)
1347         xcpuid++;
1348     break;
1349 case X86_VENDOR_Centaur:
1350 case X86_VENDOR_TM:
1351 default:
1352     xcpuid++;
1353     break;
1354 }
1355
1356 if (xcpuid) {
1357     cp = &cpi->cpi_extd[0];
1358     cp->cp_eax = 0x80000000;
1359     cpi->cpi_xmaxeax = __cpuid_insn(cp);
1360 }
1361
1362 if (cpi->cpi_xmaxeax & 0x80000000) {
1363
1364     if (cpi->cpi_xmaxeax > CPI_XMAXEAX_MAX)
1365         cpi->cpi_xmaxeax = CPI_XMAXEAX_MAX;
1366
1367     switch (cpi->cpi_vendor) {
1368     case X86_VENDOR_Intel:
1369     case X86_VENDOR_AMD:
1370         if (cpi->cpi_xmaxeax < 0x80000001)
1371             break;
1372         cp = &cpi->cpi_extd[1];
1373         cp->cp_eax = 0x80000001;
1374         (void) __cpuid_insn(cp);
1375
1376         if (cpi->cpi_vendor == X86_VENDOR_AMD &&
1377             cpi->cpi_family == 5 &&
1378             cpi->cpi_model == 6 &&
1379             cpi->cpi_step == 6) {
1380             /*
1381              * K6 model 6 uses bit 10 to indicate SYSC
1382              * Later models use bit 11. Fix it here.
1383              */
1384             if (cp->cp_edx & 0x400) {
1385                 cp->cp_edx &= ~0x400;
1386                 cp->cp_edx |= CPUID_AMD_EDX_SYSC;
1387             }
1388         }
1389
1390         platform_cpuid_mangle(cpi->cpi_vendor, 0x80000001, cp);
1391
1392         /*
1393          * Compute the additions to the kernel's feature word.
1394          */
1395         if (cp->cp_edx & CPUID_AMD_EDX_NX) {
1396             add_x86_feature(featureset, X86FSET_NX);
1397         }
1398
1399         /*
1400          * Regardless whether or not we boot 64-bit,
1401          * we should have a way to identify whether
1402          * the CPU is capable of running 64-bit.
1403          */
1404         if (cp->cp_edx & CPUID_AMD_EDX_LM) {
1405             add_x86_feature(featureset, X86FSET_64);
1406         }
1407
1408 #if defined(__amd64)

```

```

1409     /* 1 GB large page - enable only for 64 bit kernel */
1410     if (cp->cp_edx & CPUID_AMD_EDX_1GPG) {
1411         add_x86_feature(featureset, X86FSET_1GPG);
1412     }
1413 #endif
1414
1415     if ((cpi->cpi_vendor == X86_VENDOR_AMD) &&
1416         (cpi->cpi_std[1].cp_edx & CPUID_INTC_EDX_FXSR) &&
1417         (cp->cp_ecx & CPUID_AMD_ECX_SSE4A)) {
1418         add_x86_feature(featureset, X86FSET_SSE4A);
1419     }
1420
1421     /*
1422     * If both the HTT and CMP_LGCY bits are set,
1423     * then we're not actually HyperThreaded. Read
1424     * "AMD CPUID Specification" for more details.
1425     */
1426     if (cpi->cpi_vendor == X86_VENDOR_AMD &&
1427         is_x86_feature(featureset, X86FSET_HTT) &&
1428         (cp->cp_ecx & CPUID_AMD_ECX_CMP_LGCY)) {
1429         remove_x86_feature(featureset, X86FSET_HTT);
1430         add_x86_feature(featureset, X86FSET_CMP);
1431     }
1432 #if defined(__amd64)
1433     /*
1434     * It's really tricky to support syscall/sysret in
1435     * the i386 kernel; we rely on sysenter/sysexit
1436     * instead. In the amd64 kernel, things are -way-
1437     * better.
1438     */
1439     if (cp->cp_edx & CPUID_AMD_EDX_SYSC) {
1440         add_x86_feature(featureset, X86FSET_ASYSYC);
1441     }
1442
1443     /*
1444     * While we're thinking about system calls, note
1445     * that AMD processors don't support sysenter
1446     * in long mode at all, so don't try to program them.
1447     */
1448     if (x86_vendor == X86_VENDOR_AMD) {
1449         remove_x86_feature(featureset, X86FSET_SEP);
1450     }
1451 #endif
1452     if (cp->cp_edx & CPUID_AMD_EDX_TSACP) {
1453         add_x86_feature(featureset, X86FSET_TSACP);
1454     }
1455
1456     if (cp->cp_ecx & CPUID_AMD_ECX_SVM) {
1457         add_x86_feature(featureset, X86FSET_SVM);
1458     }
1459
1460     if (cp->cp_ecx & CPUID_AMD_ECX_TOPOEXT) {
1461         add_x86_feature(featureset, X86FSET_TOPOEXT);
1462     }
1463     break;
1464 default:
1465     break;
1466 }
1467
1468 /*
1469 * Get CPUID data about processor cores and hyperthreads.
1470 */
1471 switch (cpi->cpi_vendor) {
1472 case X86_VENDOR_Intel:
1473     if (cpi->cpi_maxeax >= 4) {
1474         cp = &cpi->cpi_std[4];

```



```

1475         cp->cp_eax = 4;
1476         cp->cp_ecx = 0;
1477         (void) __cpuid_insn(cp);
1478         platform_cpuid_mangle(cpi->cpi_vendor, 4, cp);
1479     }
1480     /*FALLTHROUGH*/
1481     case X86_VENDOR_AMD:
1482         if (cpi->cpi_xmaxeax < 0x80000008)
1483             break;
1484         cp = &cpi->cpi_extd[8];
1485         cp->cp_eax = 0x80000008;
1486         (void) __cpuid_insn(cp);
1487         platform_cpuid_mangle(cpi->cpi_vendor, 0x80000008, cp);
1488
1489         /*
1490          * Virtual and physical address limits from
1491          * cpuid override previously guessed values.
1492          */
1493         cpi->cpi_pabits = BITX(cp->cp_eax, 7, 0);
1494         cpi->cpi_vabits = BITX(cp->cp_eax, 15, 8);
1495         break;
1496     default:
1497         break;
1498 }
1499
1500 /*
1501  * Derive the number of cores per chip
1502  */
1503 switch (cpi->cpi_vendor) {
1504     case X86_VENDOR_Intel:
1505         if (cpi->cpi_maxeax < 4) {
1506             cpi->cpi_ncore_per_chip = 1;
1507             break;
1508         } else {
1509             cpi->cpi_ncore_per_chip =
1510                 BITX((cpi->cpi_std[4].cp_eax, 31, 26) + 1;
1511         }
1512         break;
1513     case X86_VENDOR_AMD:
1514         if (cpi->cpi_xmaxeax < 0x80000008) {
1515             cpi->cpi_ncore_per_chip = 1;
1516             break;
1517         } else {
1518             /*
1519              * On family 0xf cpuid fn 2 ECX[7:0] "NC" is
1520              * 1 less than the number of physical cores on
1521              * the chip. In family 0x10 this value can
1522              * be affected by "downcoring" - it reflects
1523              * 1 less than the number of cores actually
1524              * enabled on this node.
1525              */
1526             cpi->cpi_ncore_per_chip =
1527                 BITX((cpi->cpi_extd[8].cp_ecx, 7, 0) + 1;
1528         }
1529         break;
1530     default:
1531         cpi->cpi_ncore_per_chip = 1;
1532         break;
1533 }
1534
1535 /*
1536  * Get CPUID data about TSC Invariance in Deep C-State.
1537  */
1538 switch (cpi->cpi_vendor) {
1539     case X86_VENDOR_Intel:
1540         if (cpi->cpi_maxeax >= 7) {

```

```

1541         cp = &cpi->cpi_extd[7];
1542         cp->cp_eax = 0x80000007;
1543         cp->cp_ecx = 0;
1544         (void) __cpuid_insn(cp);
1545     }
1546     }
1547     default:
1548         break;
1549 }
1550 } else {
1551     cpi->cpi_ncore_per_chip = 1;
1552 }
1553
1554 /*
1555  * If more than one core, then this processor is CMP.
1556  */
1557 if (cpi->cpi_ncore_per_chip > 1) {
1558     add_x86_feature(featureset, X86FSET_CMP);
1559 }
1560
1561 /*
1562  * If the number of cores is the same as the number
1563  * of CPUs, then we cannot have HyperThreading.
1564  */
1565 if (cpi->cpi_ncpu_per_chip == cpi->cpi_ncore_per_chip) {
1566     remove_x86_feature(featureset, X86FSET_HTT);
1567 }
1568
1569 cpi->cpi_apicid = CPI_APIC_ID(cpi);
1570 cpi->cpi_procnodes_per_pkg = 1;
1571 cpi->cpi_cores_per_compunit = 1;
1572 if (is_x86_feature(featureset, X86FSET_HTT) == B_FALSE &&
1573     is_x86_feature(featureset, X86FSET_CMP) == B_FALSE) {
1574     /*
1575      * Single-core single-threaded processors.
1576      */
1577     cpi->cpi_chipid = -1;
1578     cpi->cpi_clogid = 0;
1579     cpi->cpi_coreid = cpu->cpu_id;
1580     cpi->cpi_pkgcoreid = 0;
1581     if (cpi->cpi_vendor == X86_VENDOR_AMD)
1582         cpi->cpi_procnodeid = BITX(cpi->cpi_apicid, 3, 0);
1583     else
1584         cpi->cpi_procnodeid = cpi->cpi_chipid;
1585 } else if (cpi->cpi_ncpu_per_chip > 1) {
1586     if (cpi->cpi_vendor == X86_VENDOR_Intel)
1587         cpuid_intel_getids(cpu, featureset);
1588     else if (cpi->cpi_vendor == X86_VENDOR_AMD)
1589         cpuid_amd_getids(cpu);
1590     else {
1591         /*
1592          * All other processors are currently
1593          * assumed to have single cores.
1594          */
1595         cpi->cpi_coreid = cpi->cpi_chipid;
1596         cpi->cpi_pkgcoreid = 0;
1597         cpi->cpi_procnodeid = cpi->cpi_chipid;
1598         cpi->cpi_compunitid = cpi->cpi_chipid;
1599     }
1600 }
1601
1602 /*
1603  * Synthesize chip "revision" and socket type
1604  */
1605 cpi->cpi_chiprev = __cpuid_chiprev(cpi->cpi_vendor, cpi->cpi_family,
1606     cpi->cpi_model, cpi->cpi_step);

```

```
1607     cpi->cpi_chiprevstr = _cpuid_chiprevstr(cpi->cpi_vendor,
1608     cpi->cpi_family, cpi->cpi_model, cpi->cpi_step);
1609     cpi->cpi_socket = _cpuid_skt(cpi->cpi_vendor, cpi->cpi_family,
1610     cpi->cpi_model, cpi->cpi_step);

1612 pass1_done:
1613     cpi->cpi_pass = 1;
1614 }
    unchanged_portion_omitted

3032 uint_t
3033 cpuid_get_compunitid(cpu_t *cpu)
3034 {
3035     ASSERT(cpuid_checkpass(cpu, 1));
3036     return (cpu->cpu_m.mcpu_cpi->cpi_compunitid);
3037 }

3039 uint_t
3040 cpuid_get_cores_per_compunit(cpu_t *cpu)
3041 {
3042     ASSERT(cpuid_checkpass(cpu, 1));
3043     return (cpu->cpu_m.mcpu_cpi->cpi_cores_per_compunit);
3044 }

3046 /*ARGSUSED*/
3047 int
3048 cpuid_have_cr8access(cpu_t *cpu)
3049 {
3050     #if defined(__amd64)
3051         return (1);
3052     #else
3053         struct cpuid_info *cpi;

3055         ASSERT(cpu != NULL);
3056         cpi = cpu->cpu_m.mcpu_cpi;
3057         if (cpi->cpi_vendor == X86_VENDOR_AMD && cpi->cpi_maxeax >= 1 &&
3058             (CPI_FEATURES_XTD_ECX(cpi) & CPUID_AMD_ECX_CR8D) != 0)
3059             return (1);
3060         return (0);
3061     #endif
3062 }
    unchanged_portion_omitted
```

```

*****
48130 Mon Apr 30 11:13:45 2012
new/usr/src/uts/i86pc/os/mp_machdep.c
2650 AMD family 0x15 PG support
*****

2 /*
3  * CDDL HEADER START
4  *
5  * The contents of this file are subject to the terms of the
6  * Common Development and Distribution License (the "License").
7  * You may not use this file except in compliance with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23  * Copyright (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
24 */
25 /*
26  * Copyright (c) 2009-2010, Intel Corporation.
27  * All rights reserved.
28 */

30 #define PSMI_1_7
31 #include <sys/smp_impldefs.h>
32 #include <sys/psm.h>
33 #include <sys/psm_modctl.h>
34 #include <sys/pit.h>
35 #include <sys/cmn_err.h>
36 #include <sys/strlog.h>
37 #include <sys/clock.h>
38 #include <sys/debug.h>
39 #include <sys/rtc.h>
40 #include <sys/x86_archext.h>
41 #include <sys/cpupart.h>
42 #include <sys/cpuvar.h>
43 #include <sys/cpu_event.h>
44 #include <sys/cmt.h>
45 #include <sys/cpu.h>
46 #include <sys/disp.h>
47 #include <sys/archsystem.h>
48 #include <sys/machsystem.h>
49 #include <sys/sysmacros.h>
50 #include <sys/memlist.h>
51 #include <sys/param.h>
52 #include <sys/promif.h>
53 #include <sys/cpu_pm.h>
54 #if defined(__xpv)
55 #include <sys/hypervisor.h>
56 #endif
57 #include <sys/mach_intr.h>
58 #include <vm/hat_i86.h>
59 #include <sys/kdi_machimpl.h>
60 #include <sys/sdt.h>
61 #include <sys/hpet.h>

```

```

62 #include <sys/sunddi.h>
63 #include <sys/sunndi.h>
64 #include <sys/cpc_pcbe.h>

66 #define OFFSETOF(s, m)      (size_t)&(((s *)0)->m)

68 /*
69  *      Local function prototypes
70 */
71 static int mp_disable_intr(processorid_t cpun);
72 static void mp_enable_intr(processorid_t cpun);
73 static void mach_init();
74 static void mach_picinit();
75 static int machhztomhz(uint64_t cpu_freq_hz);
76 static uint64_t mach_getcpufreq(void);
77 static void mach_fixcpufreq(void);
78 static int mach_clkinit(int, int *);
79 static void mach_smpinit(void);
80 static int mach_softlvl_to_vect(int ipl);
81 static void mach_get_platform(int owner);
82 static void mach_construct_info();
83 static int mach_translate_irq(dev_info_t *dip, int irqno);
84 static int mach_intr_ops(dev_info_t *, ddi_intr_handle_impl_t *,
85     psm_intr_op_t, int *);
86 static void mach_notify_error(int level, char *errmsg);
87 static void dummy_hrttime(void);
88 static void dummy_scalehrttime(hrttime_t *);
89 static uint64_t dummy_unscalehrttime(hrttime_t);
90 void cpu_idle(void);
91 static void cpu_wakeup(cpu_t *, int);
92 #ifndef __xpv
93 void cpu_idle_mwait(void);
94 static void cpu_wakeup_mwait(cpu_t *, int);
95 #endif
96 static int mach_cpu_create_devinfo(cpu_t *cp, dev_info_t **dipp);

98 /*
99  *      External reference functions
100 */
101 extern void return_instr();
102 extern uint64_t freq_tsc(uint32_t *);
103 #if defined(__i386)
104 extern uint64_t freq_notsc(uint32_t *);
105 #endif
106 extern void pc_gethrestime(timestruc_t *);
107 extern int cpuid_get_coreid(cpu_t *);
108 extern int cpuid_get_chipid(cpu_t *);

110 /*
111  *      PSM functions initialization
112 */
113 void (*psm_shutdownf)(int, int) = (void (*)(int, int))return_instr;
114 void (*psm_preshutdownf)(int, int) = (void (*)(int, int))return_instr;
115 void (*psm_notifyf)(int) = (void (*)(int))return_instr;
116 void (*psm_set_idle_cpuf)(int) = (void (*)(int))return_instr;
117 void (*psm_unset_idle_cpuf)(int) = (void (*)(int))return_instr;
118 void (*psminitf)() = mach_init;
119 void (*picinitf)() = return_instr;
120 int (*clkinitf)(int, int *) = (int (*)(int, int*))return_instr;
121 int (*ap_mlsetup)() = (int (*)(void))return_instr;
122 void (*send_dirintf)() = return_instr;
123 void (*setspl)(int) = (void (*)(int))return_instr;
124 int (*addspl)(int, int, int, int) = (int (*)(int, int, int, int))return_instr;
125 int (*delspl)(int, int, int, int) = (int (*)(int, int, int, int))return_instr;
126 int (*get_pending_spl)(void) = (int (*)(void))return_instr;
127 int (*addintr)(void *, int, avfunc, char *, int, caddr_t, caddr_t,

```

```

128     uint64_t *, dev_info_t *) = NULL;
129 void (*remintr)(void *, int, avfunc, int) = NULL;
130 void (*kdisetsoftint)(int, struct av_software *) =
131     (void (*)(int, struct av_software*))return_instr;
132 void (*setsoftint)(int, struct av_software *) =
133     (void (*)(int, struct av_software*))return_instr;
134 int (*slvltovect)(int) = (int (*)(int))return_instr;
135 int (*setlvl)(int, int *) = (int (*)(int, int*))return_instr;
136 void (*setlvlx)(int, int) = (void (*)(int, int))return_instr;
137 int (*psm_disable_intr)(int) = mp_disable_intr;
138 void (*psm_enable_intr)(int) = mp_enable_intr;
139 hrtime_t (*gethrtimef)(void) = dummy_hrtime;
140 hrtime_t (*gethrtimeunscaledf)(void) = dummy_hrtime;
141 void (*scalehrtimef)(hrtime_t *) = dummy_scalehrtime;
142 uint64_t (*unscalehrtimef)(hrtime_t) = dummy_unscalehrtime;
143 int (*psm_translate_irq)(dev_info_t *, int) = mach_translate_irq;
144 void (*gethrestimef)(timestruc_t *) = pc_gethrestime;
145 void (*psm_notify_error)(int, char *) = (void (*)(int, char*))NULL;
146 int (*psm_get_clockirq)(int) = NULL;
147 int (*psm_get_ipivect)(int, int) = NULL;
148 uchar_t (*psm_get_ioapicid)(uchar_t) = NULL;
149 uint32_t (*psm_get_localapicid)(uint32_t) = NULL;
150 uchar_t (*psm_xlate_vector_by_irq)(uchar_t) = NULL;

152 int (*psm_clkinit)(int) = NULL;
153 void (*psm_timer_reprogram)(hrtime_t) = NULL;
154 void (*psm_timer_enable)(void) = NULL;
155 void (*psm_timer_disable)(void) = NULL;
156 void (*psm_post_cyclic_setup)(void *arg) = NULL;
157 int (*psm_intr_ops)(dev_info_t *, ddi_intr_handle_impl_t *, psm_intr_op_t,
158     int *) = mach_intr_ops;
159 int (*psm_state)(psm_state_request_t *) = (int (*)(psm_state_request_t *))
160     return_instr;

162 void (*notify_error)(int, char *) = (void (*)(int, char*))return_instr;
163 void (*hrtime_tick)(void) = return_instr;

165 int (*psm_cpu_create_devinfo)(cpu_t *, dev_info_t **) = mach_cpu_create_devinfo;
166 int (*psm_cpu_get_devinfo)(cpu_t *, dev_info_t **) = NULL;

168 /* global IRM pool for APIX (PSM) module */
169 ddi_irm_pool_t *apix_irm_pool_p = NULL;

171 /*
172  * True if the generic TSC code is our source of hrtime, rather than whatever
173  * the PSM can provide.
174  */
175 #ifdef __xpv
176 int tsc_gethrtime_enable = 0;
177 #else
178 int tsc_gethrtime_enable = 1;
179 #endif
180 int tsc_gethrtime_inittd = 0;

182 /*
183  * True if the hrtime implementation is "hires"; namely, better than microdata.
184  */
185 int gethrtime_hires = 0;

187 /*
188  * Local Static Data
189  */
190 static struct psm_ops mach_ops;
191 static struct psm_ops *mach_set[4] = {&mach_ops, NULL, NULL, NULL};
192 static ushort_t mach_ver[4] = {0, 0, 0, 0};

```

```

194 /*
195  * virtualization support for psm
196  */
197 void *psm_vt_ops = NULL;
198 /*
199  * If non-zero, idle cpus will become "halted" when there's
200  * no work to do.
201  */
202 int     idle_cpu_use_hlt = 1;

204 #ifndef __xpv
205 /*
206  * If non-zero, idle cpus will use mwait if available to halt instead of hlt.
207  */
208 int     idle_cpu_prefer_mwait = 1;
209 /*
210  * Set to 0 to avoid MONITOR+CLFLUSH assertion.
211  */
212 int     idle_cpu_assert_cflush_monitor = 1;

214 /*
215  * If non-zero, idle cpus will not use power saving Deep C-States idle loop.
216  */
217 int     idle_cpu_no_deep_c = 0;
218 /*
219  * Non-power saving idle loop and wakeup pointers.
220  * Allows user to toggle Deep Idle power saving feature on/off.
221  */
222 void     (*non_deep_idle_cpu)() = cpu_idle;
223 void     (*non_deep_idle_disp_enq_thread)(cpu_t *, int);

225 /*
226  * Object for the kernel to access the HPET.
227  */
228 hpet_t hpet;

230 #endif /* ifndef __xpv */

232 uint_t cp_haltset_fanout = 0;

234 /*ARGSUSED*/
235 int
236 pg_plat_hw_shared(cpu_t *cp, pghw_type_t hw)
237 {
238     switch (hw) {
239     case PGHW_IPIPE:
240         if (is_x86_feature(x86_featureset, X86FSET_HTT)) {
241             /*
242              * Hyper-threading is SMT
243              */
244             return (1);
245         } else {
246             return (0);
247         }
248     case PGHW_FPU:
249         if (cpuid_get_cores_per_compunit(cp) > 1)
250             return (1);
251         else
252             return (0);
253     case PGHW_PROCNODE:
254         if (cpuid_get_procnodes_per_pkg(cp) > 1)
255             return (1);
256         else
257             return (0);
258     case PGHW_CHIP:
259         if (is_x86_feature(x86_featureset, X86FSET_CMP) ||

```

```

260         is_x86_feature(x86_featureset, X86FSET_HTT)
261         return (1);
262     else
263         return (0);
264 case PGHW_CACHE:
265     if (cpuid_get_ncpu_sharing_last_cache(cp) > 1)
266         return (1);
267     else
268         return (0);
269 case PGHW_POW_ACTIVE:
270     if (cpupm_domain_id(cp, CPUPM_DTYPE_ACTIVE) != (id_t)-1)
271         return (1);
272     else
273         return (0);
274 case PGHW_POW_IDLE:
275     if (cpupm_domain_id(cp, CPUPM_DTYPE_IDLE) != (id_t)-1)
276         return (1);
277     else
278         return (0);
279 default:
280     return (0);
281 }
282 }

```

unchanged portion omitted

```

302 /*
303  * Return a physical instance identifier for known hardware sharing
304  * relationships
305  */
306 id_t
307 pg_plat_hw_instance_id(cpu_t *cpu, pghw_type_t hw)
308 {
309     switch (hw) {
310     case PGHW_IPIPE:
311         return (cpuid_get_coreid(cpu));
312     case PGHW_CACHE:
313         return (cpuid_get_last_lvl_cacheid(cpu));
314     case PGHW_FPU:
315         return (cpuid_get_compunitid(cpu));
316     case PGHW_PROCNODE:
317         return (cpuid_get_procnoid(cpu));
318     case PGHW_CHIP:
319         return (cpuid_get_chipid(cpu));
320     case PGHW_POW_ACTIVE:
321         return (cpupm_domain_id(cpu, CPUPM_DTYPE_ACTIVE));
322     case PGHW_POW_IDLE:
323         return (cpupm_domain_id(cpu, CPUPM_DTYPE_IDLE));
324     default:
325         return (-1);
326     }
327 }

```

```

329 /*
330  * Express preference for optimizing for sharing relationship
331  * hw1 vs hw2
332  */
333 pghw_type_t
334 pg_plat_hw_rank(pghw_type_t hw1, pghw_type_t hw2)
335 {
336     int i, rank1, rank2;
337
338     static pghw_type_t hw_hier[] = {
339         PGHW_IPIPE,
340         PGHW_CACHE,
341         PGHW_FPU,
342         PGHW_PROCNODE,

```

```

343         PGHW_CHIP,
344         PGHW_POW_IDLE,
345         PGHW_POW_ACTIVE,
346         PGHW_NUM_COMPONENTS
347     };
348
349     for (i = 0; hw_hier[i] != PGHW_NUM_COMPONENTS; i++) {
350         if (hw_hier[i] == hw1)
351             rank1 = i;
352         if (hw_hier[i] == hw2)
353             rank2 = i;
354     }
355
356     if (rank1 > rank2)
357         return (hw1);
358     else
359         return (hw2);
360 }
361
362 /*
363  * Override the default CMT dispatcher policy for the specified
364  * hardware sharing relationship
365  */
366 pg_cmt_policy_t
367 pg_plat_cmt_policy(pghw_type_t hw)
368 {
369     /*
370      * For shared caches, also load balance across them to
371      * maximize aggregate cache capacity
372      *
373      * On AMD family 0x15 CPUs, cores come in pairs called
374      * compute units, sharing the FPU and the I$ and L2
375      * caches. Use balancing and cache affinity.
376      */
377     switch (hw) {
378     case PGHW_FPU:
379     case PGHW_CACHE:
380         return (CMT_BALANCE|CMT_AFFINITY);
381     default:
382         return (CMT_NO_POLICY);
383     }
384 }

```

unchanged portion omitted

new/usr/src/uts/intel/sys/x86_archext.h

1

```
*****
26594 Mon Apr 30 11:13:46 2012
new/usr/src/uts/intel/sys/x86_archext.h
2650 AMD family 0x15 PG support
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright (c) 1995, 2010, Oracle and/or its affiliates. All rights reserved.
23 * Copyright (c) 2011 by Delphix. All rights reserved.
24 */
25 /*
26 * Copyright (c) 2010, Intel Corporation.
27 * All rights reserved.
28 */
29 /*
30 * Copyright (c) 2011, Joyent, Inc. All rights reserved.
31 */

33 #ifndef _SYS_X86_ARCHEXT_H
34 #define _SYS_X86_ARCHEXT_H

36 #if !defined(_ASM)
37 #include <sys/regset.h>
38 #include <sys/processor.h>
39 #include <vm/seg_enum.h>
40 #include <vm/page.h>
41 #endif /* _ASM */

43 #ifdef __cplusplus
44 extern "C" {
45 #endif

47 /*
48 * cpuid instruction feature flags in %edx (standard function 1)
49 */

51 #define CPUID_INTC_EDX_FPU 0x00000001 /* x87 fpu present */
52 #define CPUID_INTC_EDX_VME 0x00000002 /* virtual-8086 extension */
53 #define CPUID_INTC_EDX_DE 0x00000004 /* debugging extensions */
54 #define CPUID_INTC_EDX_PSE 0x00000008 /* page size extension */
55 #define CPUID_INTC_EDX_TSC 0x00000010 /* time stamp counter */
56 #define CPUID_INTC_EDX_MSR 0x00000020 /* rdmsr and wrmsr */
57 #define CPUID_INTC_EDX_PAE 0x00000040 /* physical addr extension */
58 #define CPUID_INTC_EDX_MCE 0x00000080 /* machine check exception */
59 #define CPUID_INTC_EDX_CX8 0x00000100 /* cmpxchg8b instruction */
60 #define CPUID_INTC_EDX_APIC 0x00000200 /* local APIC */
61 /* 0x400 - reserved */
```

new/usr/src/uts/intel/sys/x86_archext.h

2

```
62 #define CPUID_INTC_EDX_SEP 0x00000800 /* sysenter and sysexit */
63 #define CPUID_INTC_EDX_MTRR 0x00001000 /* memory type range reg */
64 #define CPUID_INTC_EDX_PGE 0x00002000 /* page global enable */
65 #define CPUID_INTC_EDX_MCA 0x00004000 /* machine check arch */
66 #define CPUID_INTC_EDX_CMOV 0x00008000 /* conditional move insns */
67 #define CPUID_INTC_EDX_PAT 0x00010000 /* page attribute table */
68 #define CPUID_INTC_EDX_PSE36 0x00020000 /* 36-bit pagesize extension */
69 #define CPUID_INTC_EDX_PSN 0x00040000 /* processor serial number */
70 #define CPUID_INTC_EDX_CLFSH 0x00080000 /* clflush instruction */
71 /* 0x100000 - reserved */
72 #define CPUID_INTC_EDX_DS 0x00200000 /* debug store exists */
73 #define CPUID_INTC_EDX ACPI 0x00400000 /* monitoring + clock ctrl */
74 #define CPUID_INTC_EDX_MMX 0x00800000 /* MMX instructions */
75 #define CPUID_INTC_EDX_FXSR 0x01000000 /* fxsave and fxrstor */
76 #define CPUID_INTC_EDX_SSE 0x02000000 /* streaming SIMD extensions */
77 #define CPUID_INTC_EDX_SSE2 0x04000000 /* SSE extensions */
78 #define CPUID_INTC_EDX_SS 0x08000000 /* self-snoop */
79 #define CPUID_INTC_EDX_HTT 0x10000000 /* Hyper Thread Technology */
80 #define CPUID_INTC_EDX_TM 0x20000000 /* thermal monitoring */
81 #define CPUID_INTC_EDX_IA64 0x40000000 /* Itanium emulating IA32 */
82 #define CPUID_INTC_EDX_PBE 0x80000000 /* Pending Break Enable */

84 #define FMT_CPUID_INTC_EDX \
85     "\20" \
86     "\40pbe\37ia64\36tm\35htt\34ss\33sse2\32sse\31fxsr" \
87     "\30mmx\27acpi\26ds\24clfsh\23psn\22pse36\21pat" \
88     "\20cmov\17mca\16pge\15mtrr\14sep\12apic\11cx8" \
89     "\10mce\7pae\6msr\5tsc\4pse\3de\2vme\1fpu"

91 /*
92 * cpuid instruction feature flags in %ecx (standard function 1)
93 */

95 #define CPUID_INTC_ECX_SSE3 0x00000001 /* Yet more SSE extensions */
96 #define CPUID_INTC_ECX_PCLMULQDQ 0x00000002 /* PCLMULQDQ insn */
97 /* 0x00000004 - reserved */
98 #define CPUID_INTC_ECX_MON 0x00000008 /* MONITOR/MWAIT */
99 #define CPUID_INTC_ECX_DSCPL 0x00000010 /* CPL-qualified debug store */
100 #define CPUID_INTC_ECX_VMX 0x00000020 /* Hardware VM extensions */
101 #define CPUID_INTC_ECX_SMX 0x00000040 /* Secure mode extensions */
102 #define CPUID_INTC_ECX_EST 0x00000080 /* enhanced SpeedStep */
103 #define CPUID_INTC_ECX_TM2 0x00000100 /* thermal monitoring */
104 #define CPUID_INTC_ECX_SSSE3 0x00000200 /* Supplemental SSE3 insns */
105 #define CPUID_INTC_ECX_CID 0x00000400 /* L1 context ID */
106 /* 0x00000800 - reserved */
107 /* 0x00001000 - reserved */
108 #define CPUID_INTC_ECX_CX16 0x00002000 /* cmpxchg16 */
109 #define CPUID_INTC_ECX_ETPRD 0x00004000 /* extended task pri messages */
110 /* 0x00008000 - reserved */
111 /* 0x00010000 - reserved */
112 /* 0x00020000 - reserved */
113 #define CPUID_INTC_ECX_DCA 0x00040000 /* direct cache access */
114 #define CPUID_INTC_ECX_SSE4_1 0x00080000 /* SSE4.1 insns */
115 #define CPUID_INTC_ECX_SSE4_2 0x00100000 /* SSE4.2 insns */
116 #define CPUID_INTC_ECX_MOVBE 0x00400000 /* MOVBE insn */
117 #define CPUID_INTC_ECX_POPCNT 0x00800000 /* POPCNT insn */
118 #define CPUID_INTC_ECX_AES 0x02000000 /* AES insns */
119 #define CPUID_INTC_ECX_XSAVE 0x04000000 /* XSAVE/XRESTOR insns */
120 #define CPUID_INTC_ECX_OSXSAVE 0x08000000 /* OS supports XSAVE insns */
121 #define CPUID_INTC_ECX_AVX 0x10000000 /* AVX supported */

123 #define FMT_CPUID_INTC_ECX \
124     "\20" \
125     "\35avx\34osxsave\33xsave" \
126     "\32aes" \
127     "\30popcnt\27movbe\25sse4.2\24sse4.1\23dca"
```

```

128     "\20\17etprd\16cx16\13cid\12s3se3\11tm2"      \
129     "\10est\7smx\6vmx\5dscpl\4mon\2pclmulqdq\1sse3"

131 /*
132 * cpuid instruction feature flags in %edx (extended function 0x80000001)
133 */

135 #define CPUID_AMD_EDX_FPU      0x00000001    /* x87 fpu present */
136 #define CPUID_AMD_EDX_VME     0x00000002    /* virtual-8086 extension */
137 #define CPUID_AMD_EDX_DE      0x00000004    /* debugging extensions */
138 #define CPUID_AMD_EDX_PSE     0x00000008    /* page size extensions */
139 #define CPUID_AMD_EDX_TSC     0x00000010    /* time stamp counter */
140 #define CPUID_AMD_EDX_MSR     0x00000020    /* rdmsr and wrmsr */
141 #define CPUID_AMD_EDX_PAE     0x00000040    /* physical addr extension */
142 #define CPUID_AMD_EDX_MCE     0x00000080    /* machine check exception */
143 #define CPUID_AMD_EDX_CX8     0x00000100    /* cmpxchg8b instruction */
144 #define CPUID_AMD_EDX_APIC    0x00000200    /* local APIC */
145                                     /* 0x00000400 - sysc on K6m6 */
146 #define CPUID_AMD_EDX_SYSC    0x00000800    /* AMD: syscall and sysret */
147 #define CPUID_AMD_EDX_MTRR    0x00001000    /* memory type and range reg */
148 #define CPUID_AMD_EDX_PGE     0x00002000    /* page global enable */
149 #define CPUID_AMD_EDX_MCA     0x00004000    /* machine check arch */
150 #define CPUID_AMD_EDX_CMOV    0x00008000    /* conditional move insns */
151 #define CPUID_AMD_EDX_PAT     0x00010000    /* K7: page attribute table */
152 #define CPUID_AMD_EDX_FCMOV   0x00010000    /* FCMOVcc etc. */
153 #define CPUID_AMD_EDX_PSE36   0x00020000    /* 36-bit pagesize extension */
154                                     /* 0x00040000 - reserved */
155                                     /* 0x00080000 - reserved */
156 #define CPUID_AMD_EDX_NX      0x00100000    /* AMD: no-execute page prot */
157                                     /* 0x00200000 - reserved */
158 #define CPUID_AMD_EDX_MMXamd  0x00400000    /* AMD: MMX extensions */
159 #define CPUID_AMD_EDX_MMX     0x00800000    /* MMX instructions */
160 #define CPUID_AMD_EDX_FXSR    0x01000000    /* fxsave and fxrstor */
161 #define CPUID_AMD_EDX_FFXSR   0x02000000    /* fast fxsave/fxrstor */
162 #define CPUID_AMD_EDX_LGPG    0x04000000    /* LGB page */
163 #define CPUID_AMD_EDX_TSCP    0x08000000    /* rdtscl instruction */
164                                     /* 0x10000000 - reserved */
165 #define CPUID_AMD_EDX_LM      0x20000000    /* AMD: long mode */
166 #define CPUID_AMD_EDX_3DNowx  0x40000000    /* AMD: extensions to 3DNow! */
167 #define CPUID_AMD_EDX_3DNow   0x80000000    /* AMD: 3DNow! instructions */

169 #define FMT_CPUID_AMD_EDX     \
170     "\20" \
171     "\40a3d\37a3d\36lm\34tscp\32ffxsr\31fxsr" \
172     "\30mmx\27mmxext\25nx\22pse\21pat" \
173     "\20cmov\17mca\16pge\15mtrr\14syscall\12apic\11cx8" \
174     "\10mce\7pae\6msr\5tsc\4pse\3de\2vme\1fpu"

176 #define CPUID_AMD_ECX_AHF64   0x00000001    /* LAHF and SAHF in long mode */
177 #define CPUID_AMD_ECX_CMP_LGCY 0x00000002    /* AMD: multicore chip */
178 #define CPUID_AMD_ECX_SVM     0x00000004    /* AMD: secure VM */
179 #define CPUID_AMD_ECX_EAS     0x00000008    /* extended apic space */
180 #define CPUID_AMD_ECX_CR8D    0x00000010    /* AMD: 32-bit mov %cr8 */
181 #define CPUID_AMD_ECX_LZCNT   0x00000020    /* AMD: LZCNT insn */
182 #define CPUID_AMD_ECX_SSE4A   0x00000040    /* AMD: SSE4A insns */
183 #define CPUID_AMD_ECX_MAS     0x00000080    /* AMD: MisAlignSse mmode */
184 #define CPUID_AMD_ECX_3DNP    0x00000100    /* AMD: 3DNowPrefetch */
185 #define CPUID_AMD_ECX_OSVW    0x00000200    /* AMD: OSVW */
186 #define CPUID_AMD_ECX_IBS     0x00000400    /* AMD: IBS */
187 #define CPUID_AMD_ECX_SSE5    0x00000800    /* AMD: SSE5 */
188 #define CPUID_AMD_ECX_SKINIT   0x00001000    /* AMD: SKINIT */
189 #define CPUID_AMD_ECX_WDT     0x00002000    /* AMD: WDT */
190 #define CPUID_AMD_ECX_TOPOEXT 0x00400000    /* AMD: Topology Extensions */

192 #define FMT_CPUID_AMD_ECX     \
193     "\20" \

```

```

194     "\22topoext" \
195     "\14wdt\13skinit\12sse5\11libs\10osvw\93dnp\8mas" \
196     "\7sse4a\6lzcnc\5scr8d\3svm\2lcmplgcy\lahf64"

198 /*
199 * Intel now seems to have claimed part of the "extended" function
200 * space that we previously for non-Intel implementors to use.
201 * More excitingly still, they've claimed bit 20 to mean LAHF/SAHF
202 * is available in long mode i.e. what AMD indicate using bit 0.
203 * On the other hand, everything else is labelled as reserved.
204 */
205 #define CPUID_INTC_ECX_AHF64   0x00100000    /* LAHF and SAHF in long mode */

208 #define P5_MCHADDR            0x0
209 #define P5_CESR               0x11
210 #define P5_CTR0               0x12
211 #define P5_CTR1               0x13

213 #define K5_MCHADDR            0x0
214 #define K5_MCHTYPE           0x01
215 #define K5_TSC               0x10
216 #define K5_TR12              0x12

218 #define REG_PAT               0x277

220 #define REG_MC0_CTL           0x400
221 #define REG_MC5_MISC         0x417
222 #define REG_PERFCTR0         0xc1
223 #define REG_PERFCTR1         0xc2

225 #define REG_PERFEVNT0        0x186
226 #define REG_PERFEVNT1        0x187

228 #define REG_TSC               0x10    /* timestamp counter */
229 #define REG_APIC_BASE_MSR    0x1b
230 #define REG_X2APIC_BASE_MSR 0x800    /* The MSR address offset of x2APIC */

232 #if !defined(__xpv)
233 /*
234 * AMD C1E
235 */
236 #define MSR_AMD_INT_PENDING_CMP_HALT 0xc0010055
237 #define AMD_ACTONCMPHALT_SHIFT 27
238 #define AMD_ACTONCMPHALT_MASK 3
239 #endif

241 #define MSR_DEBUGCTL          0x1d9

243 #define DEBUGCTL_LBR          0x01
244 #define DEBUGCTL_BTF         0x02

246 /* Intel P6, AMD */
247 #define MSR_LBR_FROM          0x1db
248 #define MSR_LBR_TO           0x1dc
249 #define MSR_LEX_FROM         0x1dd
250 #define MSR_LEX_TO           0x1de

252 /* Intel P4 (pre-Prescott, non P4 M) */
253 #define MSR_P4_LBSTK_TOS      0x1da
254 #define MSR_P4_LBSTK_0       0x1db
255 #define MSR_P4_LBSTK_1       0x1dc
256 #define MSR_P4_LBSTK_2       0x1dd
257 #define MSR_P4_LBSTK_3       0x1de

259 /* Intel Pentium M */

```

```

260 #define MSR_P6M_LBSTK_TOS      0x1c9
261 #define MSR_P6M_LBSTK_0        0x040
262 #define MSR_P6M_LBSTK_1        0x041
263 #define MSR_P6M_LBSTK_2        0x042
264 #define MSR_P6M_LBSTK_3        0x043
265 #define MSR_P6M_LBSTK_4        0x044
266 #define MSR_P6M_LBSTK_5        0x045
267 #define MSR_P6M_LBSTK_6        0x046
268 #define MSR_P6M_LBSTK_7        0x047

270 /* Intel P4 (Prescott) */
271 #define MSR_PRP4_LBSTK_TOS      0x1da
272 #define MSR_PRP4_LBSTK_FROM_0   0x680
273 #define MSR_PRP4_LBSTK_FROM_1   0x681
274 #define MSR_PRP4_LBSTK_FROM_2   0x682
275 #define MSR_PRP4_LBSTK_FROM_3   0x683
276 #define MSR_PRP4_LBSTK_FROM_4   0x684
277 #define MSR_PRP4_LBSTK_FROM_5   0x685
278 #define MSR_PRP4_LBSTK_FROM_6   0x686
279 #define MSR_PRP4_LBSTK_FROM_7   0x687
280 #define MSR_PRP4_LBSTK_FROM_8   0x688
281 #define MSR_PRP4_LBSTK_FROM_9   0x689
282 #define MSR_PRP4_LBSTK_FROM_10  0x68a
283 #define MSR_PRP4_LBSTK_FROM_11  0x68b
284 #define MSR_PRP4_LBSTK_FROM_12  0x68c
285 #define MSR_PRP4_LBSTK_FROM_13  0x68d
286 #define MSR_PRP4_LBSTK_FROM_14  0x68e
287 #define MSR_PRP4_LBSTK_FROM_15  0x68f
288 #define MSR_PRP4_LBSTK_TO_0     0x6c0
289 #define MSR_PRP4_LBSTK_TO_1     0x6c1
290 #define MSR_PRP4_LBSTK_TO_2     0x6c2
291 #define MSR_PRP4_LBSTK_TO_3     0x6c3
292 #define MSR_PRP4_LBSTK_TO_4     0x6c4
293 #define MSR_PRP4_LBSTK_TO_5     0x6c5
294 #define MSR_PRP4_LBSTK_TO_6     0x6c6
295 #define MSR_PRP4_LBSTK_TO_7     0x6c7
296 #define MSR_PRP4_LBSTK_TO_8     0x6c8
297 #define MSR_PRP4_LBSTK_TO_9     0x6c9
298 #define MSR_PRP4_LBSTK_TO_10    0x6ca
299 #define MSR_PRP4_LBSTK_TO_11    0x6cb
300 #define MSR_PRP4_LBSTK_TO_12    0x6cc
301 #define MSR_PRP4_LBSTK_TO_13    0x6cd
302 #define MSR_PRP4_LBSTK_TO_14    0x6ce
303 #define MSR_PRP4_LBSTK_TO_15    0x6cf

305 #define MCI_CTL_VALUE           0xffffffff

307 #define MTRR_TYPE_UC            0
308 #define MTRR_TYPE_WC            1
309 #define MTRR_TYPE_WT            4
310 #define MTRR_TYPE_WP            5
311 #define MTRR_TYPE_WB            6
312 #define MTRR_TYPE_UC_          7

314 /*
315  * For Solaris we set up the page attribute table in the following way:
316  * PAT0 Write-Back
317  * PAT1 Write-Through
318  * PAT2 Uncacheable-
319  * PAT3 Uncacheable
320  * PAT4 Write-Back
321  * PAT5 Write-Through
322  * PAT6 Write-Combine
323  * PAT7 Uncacheable
324  * The only difference from h/w default is entry 6.
325  */

```

```

326 #define PAT_DEFAULT_ATTRIBUTE  \
327     ((uint64_t)MTRR_TYPE_WB | \
328     ((uint64_t)MTRR_TYPE_WT << 8) | \
329     ((uint64_t)MTRR_TYPE_UC_ << 16) | \
330     ((uint64_t)MTRR_TYPE_UC << 24) | \
331     ((uint64_t)MTRR_TYPE_WB << 32) | \
332     ((uint64_t)MTRR_TYPE_WT << 40) | \
333     ((uint64_t)MTRR_TYPE_WC << 48) | \
334     ((uint64_t)MTRR_TYPE_UC << 56))

336 #define X86FSET_LARGE PAGE      0
337 #define X86FSET_TSC             1
338 #define X86FSET_MSR             2
339 #define X86FSET_MTRR            3
340 #define X86FSET_PGE             4
341 #define X86FSET_DE              5
342 #define X86FSET_CMOV            6
343 #define X86FSET_MMX             7
344 #define X86FSET_MCA             8
345 #define X86FSET_PAE             9
346 #define X86FSET_CX8            10
347 #define X86FSET_PAT            11
348 #define X86FSET_SEP            12
349 #define X86FSET_SSE            13
350 #define X86FSET_SSE2           14
351 #define X86FSET_HTT            15
352 #define X86FSET_ASYSC          16
353 #define X86FSET_NX             17
354 #define X86FSET_SSE3           18
355 #define X86FSET_CX16           19
356 #define X86FSET_CMP            20
357 #define X86FSET_TSCP           21
358 #define X86FSET_MWAIT          22
359 #define X86FSET_SSE4A          23
360 #define X86FSET_CPUID          24
361 #define X86FSET_SSSE3          25
362 #define X86FSET_SSE4_1         26
363 #define X86FSET_SSE4_2         27
364 #define X86FSET_LGPG           28
365 #define X86FSET_CLFSH          29
366 #define X86FSET_64             30
367 #define X86FSET_AES            31
368 #define X86FSET_PCLMULQDQ      32
369 #define X86FSET_XSAVE          33
370 #define X86FSET_AVX            34
371 #define X86FSET_VMX            35
372 #define X86FSET_SVM            36
373 #define X86FSET_TOPOEXT        37

375 /*
376  * flags to patch tsc_read routine.
377  */
378 #define X86_NO_TSC               0x0
379 #define X86_HAVE_TSCP            0x1
380 #define X86_TSC_MFENCE           0x2
381 #define X86_TSC_LFENCE           0x4

383 /*
384  * Intel Deep C-State invariant TSC in leaf 0x80000007.
385  */
386 #define CPUID_TSC_CSTATE_INVARIANCE (0x100)

388 /*
389  * Intel Deep C-state always-running local APIC timer
390  */
391 #define CPUID_CSTATE_ARAT        (0x4)

```



```

393 /*
394 * Intel ENERGY_PERF_BIAS MSR indicated by feature bit CPUID.6.ECX[3].
395 */
396 #define CPUID_EPB_SUPPORT      (1 << 3)

398 /*
399 * Intel TSC deadline timer
400 */
401 #define CPUID_DEADLINE_TSC    (1 << 24)

403 /*
404 * x86_type is a legacy concept; this is supplanted
405 * for most purposes by x86_featureset; modern CPUs
406 * should be X86_TYPE_OTHER
407 */
408 #define X86_TYPE_OTHER        0
409 #define X86_TYPE_486          1
410 #define X86_TYPE_P5           2
411 #define X86_TYPE_P6           3
412 #define X86_TYPE_CYRIX_486    4
413 #define X86_TYPE_CYRIX_6x86L  5
414 #define X86_TYPE_CYRIX_6x86   6
415 #define X86_TYPE_CYRIX_GXM    7
416 #define X86_TYPE_CYRIX_6x86MX 8
417 #define X86_TYPE_CYRIX_MediaGX 9
418 #define X86_TYPE_CYRIX_MII    10
419 #define X86_TYPE_VIA_CYRIX_III 11
420 #define X86_TYPE_P4           12

422 /*
423 * x86_vendor allows us to select between
424 * implementation features and helps guide
425 * the interpretation of the cpuid instruction.
426 */
427 #define X86_VENDOR_Intel      0
428 #define X86_VENDORSTR_Intel  "GenuineIntel"

430 #define X86_VENDOR_IntelClone 1

432 #define X86_VENDOR_AMD        2
433 #define X86_VENDORSTR_AMD     "AuthenticAMD"

435 #define X86_VENDOR_Cyrix      3
436 #define X86_VENDORSTR_CYRIX  "CyrixInstead"

438 #define X86_VENDOR_UMC        4
439 #define X86_VENDORSTR_UMC    "UMC UMC UMC "

441 #define X86_VENDOR_NexGen     5
442 #define X86_VENDORSTR_NexGen "NexGenDriven"

444 #define X86_VENDOR_Centaur    6
445 #define X86_VENDORSTR_Centaur "CentaurHauls"

447 #define X86_VENDOR_Rise       7
448 #define X86_VENDORSTR_Rise    "RiseRiseRise"

450 #define X86_VENDOR_SiS        8
451 #define X86_VENDORSTR_SiS     "SiS SiS SiS "

453 #define X86_VENDOR_TM         9
454 #define X86_VENDORSTR_TM      "GenuineTMx86"

456 #define X86_VENDOR_NSC        10
457 #define X86_VENDORSTR_NSC     "Geode by NSC"

```

```

459 /*
460 * Vendor string max len + \0
461 */
462 #define X86_VENDOR_STRLLEN    13

464 /*
465 * Some vendor/family/model/stepping ranges are commonly grouped under
466 * a single identifying banner by the vendor. The following encode
467 * that "revision" in a uint32_t with the 8 most significant bits
468 * identifying the vendor with X86_VENDOR_*, the next 8 identifying the
469 * family, and the remaining 16 typically forming a bitmask of revisions
470 * within that family with more significant bits indicating "later" revisions.
471 */

473 #define _X86_CHIPREV_VENDOR_MASK      0xff000000u
474 #define _X86_CHIPREV_VENDOR_SHIFT    24
475 #define _X86_CHIPREV_FAMILY_MASK     0x00ff0000u
476 #define _X86_CHIPREV_FAMILY_SHIFT    16
477 #define _X86_CHIPREV_REV_MASK        0x0000ffffu

479 #define _X86_CHIPREV_VENDOR(x) \
480 ((x) & _X86_CHIPREV_VENDOR_MASK) >> _X86_CHIPREV_VENDOR_SHIFT)
481 #define _X86_CHIPREV_FAMILY(x) \
482 (((x) & _X86_CHIPREV_FAMILY_MASK) >> _X86_CHIPREV_FAMILY_SHIFT)
483 #define _X86_CHIPREV_REV(x) \
484 ((x) & _X86_CHIPREV_REV_MASK)

486 /* True if x matches in vendor and family and if x matches the given rev mask */
487 #define X86_CHIPREV_MATCH(x, mask) \
488 (_X86_CHIPREV_VENDOR(x) == _X86_CHIPREV_VENDOR(mask) && \
489  _X86_CHIPREV_FAMILY(x) == _X86_CHIPREV_FAMILY(mask) && \
490  ((_X86_CHIPREV_REV(x) & _X86_CHIPREV_REV(mask)) != 0))

492 /* True if x matches in vendor and family, and rev is at least minx */
493 #define X86_CHIPREV_ATLEAST(x, minx) \
494 (_X86_CHIPREV_VENDOR(x) == _X86_CHIPREV_VENDOR(minx) && \
495  _X86_CHIPREV_FAMILY(x) == _X86_CHIPREV_FAMILY(minx) && \
496  _X86_CHIPREV_REV(x) >= _X86_CHIPREV_REV(minx))

498 #define _X86_CHIPREV_MKREV(vendor, family, rev) \
499 ((uint32_t)(vendor) << _X86_CHIPREV_VENDOR_SHIFT | \
500  (family) << _X86_CHIPREV_FAMILY_SHIFT | (rev))

502 /* True if x matches in vendor, and family is at least minx */
503 #define X86_CHIPFAM_ATLEAST(x, minx) \
504 (_X86_CHIPREV_VENDOR(x) == _X86_CHIPREV_VENDOR(minx) && \
505  _X86_CHIPREV_FAMILY(x) >= _X86_CHIPREV_FAMILY(minx))

507 /* Revision default */
508 #define X86_CHIPREV_UNKNOWN      0x0

510 /*
511 * Definitions for AMD Family 0xf. Minor revisions C0 and CG are
512 * sufficiently different that we will distinguish them; in all other
513 * case we will identify the major revision.
514 */
515 #define X86_CHIPREV_AMD_F_REV_B _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0001)
516 #define X86_CHIPREV_AMD_F_REV_C0 _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0002)
517 #define X86_CHIPREV_AMD_F_REV_CG _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0004)
518 #define X86_CHIPREV_AMD_F_REV_D _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0008)
519 #define X86_CHIPREV_AMD_F_REV_E _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0010)
520 #define X86_CHIPREV_AMD_F_REV_F _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0020)
521 #define X86_CHIPREV_AMD_F_REV_G _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0040)

523 /*

```

```

524 * Definitions for AMD Family 0x10. Rev A was Engineering Samples only.
525 */
526 #define X86_CHIPREV_AMD_10_REV_A \
527     _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0001)
528 #define X86_CHIPREV_AMD_10_REV_B \
529     _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0002)
530 #define X86_CHIPREV_AMD_10_REV_C \
531     _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0004)
532 #define X86_CHIPREV_AMD_10_REV_D \
533     _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0008)

535 /*
536 * Definitions for AMD Family 0x11.
537 */
538 #define X86_CHIPREV_AMD_11 \
539     _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x11, 0x0001)

542 /*
543 * Various socket/package types, extended as the need to distinguish
544 * a new type arises. The top 8 byte identifies the vendor and the
545 * remaining 24 bits describe 24 socket types.
546 */
548 #define _X86_SOCKET_VENDOR_SHIFT      24
549 #define _X86_SOCKET_VENDOR(x)        ((x) >> _X86_SOCKET_VENDOR_SHIFT)
550 #define _X86_SOCKET_TYPE_MASK        0x00ffffff
551 #define _X86_SOCKET_TYPE(x)          ((x) & _X86_SOCKET_TYPE_MASK)

553 #define _X86_SOCKET_MKVAL(vendor, bitval) \
554     ((uint32_t)(vendor) << _X86_SOCKET_VENDOR_SHIFT | (bitval))

556 #define X86_SOCKET_MATCH(s, mask) \
557     (_X86_SOCKET_VENDOR(s) == _X86_SOCKET_VENDOR(mask) && \
558     (_X86_SOCKET_TYPE(s) & _X86_SOCKET_TYPE(mask)) != 0)

560 #define X86_SOCKET_UNKNOWN 0x0
561 /*
562 * AMD socket types
563 */
564 #define X86_SOCKET_754          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000001)
565 #define X86_SOCKET_939          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000002)
566 #define X86_SOCKET_940          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000004)
567 #define X86_SOCKET_S1g1         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000008)
568 #define X86_SOCKET_AM2          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000010)
569 #define X86_SOCKET_F1207        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000020)
570 #define X86_SOCKET_S1g2         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000040)
571 #define X86_SOCKET_S1g3         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000080)
572 #define X86_SOCKET_AM           _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000100)
573 #define X86_SOCKET_AM2R2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000200)
574 #define X86_SOCKET_AM3          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000400)
575 #define X86_SOCKET_G34          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000800)
576 #define X86_SOCKET_AS2          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x001000)
577 #define X86_SOCKET_C32          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x002000)

579 /*
580 * xgetbv/xsetbv support
581 */

583 #define XFEATURE_ENABLED_MASK 0x0
584 /*
585 * XFEATURE_ENABLED_MASK values (eax)
586 */
587 #define XFEATURE_LEGACY_FP      0x1
588 #define XFEATURE_SSE            0x2
589 #define XFEATURE_AVX            0x4

```

```

590 #define XFEATURE_MAX              XFEATURE_AVX
591 #define XFEATURE_FP_ALL          (XFEATURE_LEGACY_FP|XFEATURE_SSE|XFEATURE_AVX)

593 #if !defined(_ASM)

595 #if defined(_KERNEL) || defined(_KMEMUSER)

597 #define NUM_X86_FEATURES        38
598 #define NUM_X86_FEATURES        37
598 extern uchar_t x86_featureset[];

600 extern void free_x86_featureset(void *featureset);
601 extern boolean_t is_x86_feature(void *featureset, uint_t feature);
602 extern void add_x86_feature(void *featureset, uint_t feature);
603 extern void remove_x86_feature(void *featureset, uint_t feature);
604 extern boolean_t compare_x86_featureset(void *setA, void *setB);
605 extern void print_x86_featureset(void *featureset);

608 extern uint_t x86_type;
609 extern uint_t x86_vendor;
610 extern uint_t x86_clflush_size;

612 extern uint_t pentiumpro_bug4046376;
613 extern uint_t pentiumpro_bug4064495;

615 extern uint_t enable486;

617 extern const char CyrixInstead[];

619 #endif

621 #if defined(_KERNEL)

623 /*
624 * This structure is used to pass arguments and get return values back
625 * from the CPUID instruction in __cpuid_insn() routine.
626 */
627 struct cpuid_regs {
628     uint32_t      cp_eax;
629     uint32_t      cp_ebx;
630     uint32_t      cp_ecx;
631     uint32_t      cp_edx;
632 };

634 /*
635 * Utility functions to get/set extended control registers (XCR)
636 * Initial use is to get/set the contents of the XFEATURE_ENABLED_MASK.
637 */
638 extern uint64_t get_xcr(uint_t);
639 extern void set_xcr(uint_t, uint64_t);

641 extern uint64_t rdmsr(uint_t);
642 extern void wrmsr(uint_t, const uint64_t);
643 extern uint64_t xrdmsr(uint_t);
644 extern void xwrmsr(uint_t, const uint64_t);
645 extern int checked_rdmsr(uint_t, uint64_t *);
646 extern int checked_wrmsr(uint_t, uint64_t);

648 extern void invalidate_cache(void);
649 extern ulong_t getcr4(void);
650 extern void setcr4(ulong_t);

652 extern void mtrr_sync(void);

654 extern void cpu_fast_syscall_enable(void *);

```

```

655 extern void cpu_fast_syscall_disable(void *);

657 struct cpu;

659 extern int cpuid_checkpass(struct cpu *, int);
660 extern uint32_t cpuid_insn(struct cpu *, struct cpuid_regs *);
661 extern uint32_t __cpuid_insn(struct cpuid_regs *);
662 extern int cpuid_getbrandstr(struct cpu *, char *, size_t);
663 extern int cpuid_getidstr(struct cpu *, char *, size_t);
664 extern const char *cpuid_getvendorstr(struct cpu *);
665 extern uint_t cpuid_getvendor(struct cpu *);
666 extern uint_t cpuid_getfamily(struct cpu *);
667 extern uint_t cpuid_getmodel(struct cpu *);
668 extern uint_t cpuid_getstep(struct cpu *);
669 extern uint_t cpuid_getsig(struct cpu *);
670 extern uint_t cpuid_get_ncpu_per_chip(struct cpu *);
671 extern uint_t cpuid_get_ncore_per_chip(struct cpu *);
672 extern uint_t cpuid_get_ncpu_sharing_last_cache(struct cpu *);
673 extern id_t cpuid_get_last_lvl_cacheid(struct cpu *);
674 extern int cpuid_get_chipid(struct cpu *);
675 extern id_t cpuid_get_coreid(struct cpu *);
676 extern int cpuid_get_pkgcoreid(struct cpu *);
677 extern int cpuid_get_clogid(struct cpu *);
678 extern int cpuid_get_cacheid(struct cpu *);
679 extern uint32_t cpuid_get_apicid(struct cpu *);
680 extern uint_t cpuid_get_procnodeid(struct cpu *cpu);
681 extern uint_t cpuid_get_procnodes_per_pkg(struct cpu *cpu);
682 extern uint_t cpuid_get_compunitid(struct cpu *cpu);
683 extern uint_t cpuid_get_cores_per_compunit(struct cpu *cpu);
684 extern int cpuid_is_cmt(struct cpu *);
685 extern int cpuid_syscall32_insn(struct cpu *);
686 extern int getl2cacheinfo(struct cpu *, int *, int *, int *);

688 extern uint32_t cpuid_getchipprev(struct cpu *);
689 extern const char *cpuid_getchipprevstr(struct cpu *);
690 extern uint32_t cpuid_getsockettype(struct cpu *);
691 extern const char *cpuid_getsocketstr(struct cpu *);

693 extern int cpuid_have_cr8access(struct cpu *);

695 extern int cpuid_opteron_erratum(struct cpu *, uint_t);

697 struct cpuid_info;

699 extern void setx86isalist(void);
700 extern void cpuid_alloc_space(struct cpu *);
701 extern void cpuid_free_space(struct cpu *);
702 extern void cpuid_pass1(struct cpu *, uchar_t *);
703 extern void cpuid_pass2(struct cpu *);
704 extern void cpuid_pass3(struct cpu *);
705 extern uint_t cpuid_pass4(struct cpu *);
706 extern void cpuid_set_cpu_properties(void *, processorid_t,
707     struct cpuid_info *);

709 extern void cpuid_get_addrsize(struct cpu *, uint_t *, uint_t *);
710 extern uint_t cpuid_get_dtlb_nent(struct cpu *, size_t);

712 #if !defined(__xpv)
713 extern uint32_t *cpuid_mwait_alloc(struct cpu *);
714 extern void cpuid_mwait_free(struct cpu *);
715 extern int cpuid_deep_cstates_supported(void);
716 extern int cpuid_arat_supported(void);
717 extern int cpuid_iepb_supported(struct cpu *);
718 extern int cpuid_deadline_tsc_supported(void);
719 extern int vmware_platform(void);
720 #endif

```

```

722 struct cpu_ucose_info;

724 extern void ucode_alloc_space(struct cpu *);
725 extern void ucode_free_space(struct cpu *);
726 extern void ucode_check(struct cpu *);
727 extern void ucode_cleanup();

729 #if !defined(__xpv)
730 extern char _tsc_mfence_start;
731 extern char _tsc_mfence_end;
732 extern char _tscp_start;
733 extern char _tscp_end;
734 extern char _no_rdtsc_start;
735 extern char _no_rdtsc_end;
736 extern char _tsc_lfence_start;
737 extern char _tsc_lfence_end;
738 #endif

740 #if !defined(__xpv)
741 extern char bcopy_patch_start;
742 extern char bcopy_patch_end;
743 extern char bcopy_ck_size;
744 #endif

746 extern void post_startup_cpu_fixups(void);

748 extern uint_t workaround_errata(struct cpu *);

750 #if defined(OPTERON_ERRATUM_93)
751 extern int opteron_erratum_93;
752 #endif

754 #if defined(OPTERON_ERRATUM_91)
755 extern int opteron_erratum_91;
756 #endif

758 #if defined(OPTERON_ERRATUM_100)
759 extern int opteron_erratum_100;
760 #endif

762 #if defined(OPTERON_ERRATUM_121)
763 extern int opteron_erratum_121;
764 #endif

766 #if defined(OPTERON_WORKAROUND_6323525)
767 extern int opteron_workaround_6323525;
768 extern void patch_workaround_6323525(void);
769 #endif

771 #if !defined(__xpv)
772 extern void determine_platform(void);
773 #endif
774 extern int get_hwenv(void);
775 extern int is_controlldom(void);

777 extern void xsave_setup_msr(struct cpu *);

779 /*
780  * Defined hardware environments
781  */
782 #define HW_NATIVE 0x00 /* Running on bare metal */
783 #define HW_XEN_PV 0x01 /* Running on Xen Hypervisor paravirtualized */
784 #define HW_XEN_HVM 0x02 /* Running on Xen hypervisor HVM */
785 #define HW_VMWARE 0x03 /* Running on VMware hypervisor */

```

new/usr/src/uts/intel/sys/x86_archext.h

13

```
787 #endif /* _KERNEL */
```

```
789 #endif
```

```
791 #ifdef __cplusplus
```

```
792 }  
unchanged_portion_omitted
```