```
**********************************************************
  122445 Tue Jan  7 20:34:25 2014
new/usr/src/uts/i86pc/os/cpuid.c
4444 remove unused cpuid-related globals
**********************************************************
```

```
   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  * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
  25  */
  26 /*
  27  * Copyright (c) 2010, Intel Corporation.
  28  * All rights reserved.
  29  */
  30 /*
  31  * Portions Copyright 2009 Advanced Micro Devices, Inc.
  32  */
  33 /*
  34  * Copyright (c) 2012, Joyent, Inc. All rights reserved.
  35  */
  36 /*
  37  * Various routines to handle identification
  38  * and classification of x86 processors.
  39  */

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

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



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

  66 /*
  67  * Pass 0 of cpuid feature analysis happens in locore. It contains special code
  68  * to recognize Cyrix processors that are not cpuid-compliant, and to deal with
  69  * them accordingly. For most modern processors, feature detection occurs here
  70  * in pass 1.
  71  *
  72  * Pass 1 of cpuid feature analysis happens just at the beginning of mlsetup()
  73  * for the boot CPU and does the basic analysis that the early kernel needs.
  74  * x86_featureset is set based on the return value of cpuid_pass1() of the boot
  75  * CPU.
  76  *
  77  * Pass 1 includes:
  78  *
  79  *      o Determining vendor/model/family/stepping and setting x86_type and
  80  *        x86_vendor accordingly.
  81  *      o Processing the feature flags returned by the cpuid instruction while
  82  *        applying any workarounds or tricks for the specific processor.
  83  *      o Mapping the feature flags into Solaris feature bits (X86_*).
  84  *      o Processing extended feature flags if supported by the processor,
  85  *        again while applying specific processor knowledge.
  86  *      o Determining the CMT characteristics of the system.
  87  *
  88  * Pass 1 is done on non-boot CPUs during their initialization and the results
  89  * are used only as a meager attempt at ensuring that all processors within the
  90  * system support the same features.
  91  *
  92  * Pass 2 of cpuid feature analysis happens just at the beginning
  93  * of startup().  It just copies in and corrects the remainder
  94  * of the cpuid data we depend on: standard cpuid functions that we didn't
  95  * need for pass1 feature analysis, and extended cpuid functions beyond the
  96  * simple feature processing done in pass1.
  97  *
  98  * Pass 3 of cpuid analysis is invoked after basic kernel services; in
  99  * particular kernel memory allocation has been made available. It creates a
 100  * readable brand string based on the data collected in the first two passes.
 101  *
 102  * Pass 4 of cpuid analysis is invoked after post_startup() when all
 103  * the support infrastructure for various hardware features has been
 104  * initialized. It determines which processor features will be reported
 105  * to userland via the aux vector.
 106  *
 107  * All passes are executed on all CPUs, but only the boot CPU determines what
 108  * features the kernel will use.
 109  *
 110  * Much of the worst junk in this file is for the support of processors
 111  * that didn't really implement the cpuid instruction properly.
 112  *
 113  * NOTE: The accessor functions (cpuid_get*) are aware of, and ASSERT upon,
 114  * the pass numbers.  Accordingly, changes to the pass code may require changes
 115  * to the accessor code.
 116  */

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

 122 uint_t pentiumpro_bug4046376;
 123 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         "ssse3",
 153         "sse4_1",
 154         "sse4_2",
 155         "1gpg",
 156         "clfsh",
 157         "64",
 158         "aes",
 159         "pclmulqdq",
 160         "xsave",
 161         "avx",
 162         "vmx",
 163         "svm",
 164         "topoext",
 165         "f16c",
 166         "rdrand"
 167 };
______unchanged_portion_omitted_

 217 uint_t enable486;

 216 static size_t xsave_state_size = 0;
 217 uint64_t xsave_bv_all = (XFEATURE_LEGACY_FP | XFEATURE_SSE);
 218 boolean_t xsave_force_disable = B_FALSE;

 220 /*
 221  * This is set to platform type we are running on.
 222  */
 223 static int platform_type = -1;

 225 #if !defined(__xpv)
 226 /*
 227  * Variable to patch if hypervisor platform detection needs to be
 228  * disabled (e.g. platform_type will always be HW_NATIVE if this is 0).
 229  */
 230 int enable_platform_detection = 1;
 231 #endif

 233 /*
 234  * monitor/mwait info.
 235  *
 236  * size_actual and buf_actual are the real address and size allocated to get
```



```
 237  * proper mwait_buf alignement.  buf_actual and size_actual should be passed
 238  * to kmem_free().  Currently kmem_alloc() and mwait happen to both use
 239  * processor cache-line alignment, but this is not guarantied in the furture.
 240  */
 241 struct mwait_info {
 242         size_t          mon_min;        /* min size to avoid missed wakeups */
 243         size_t          mon_max;        /* size to avoid false wakeups */
 244         size_t          size_actual;    /* size actually allocated */
 245         void            *buf_actual;    /* memory actually allocated */
 246         uint32_t        support;        /* processor support of monitor/mwait */
 247 };
______unchanged_portion_omitted_

 933 void
 934 cpuid_pass1(cpu_t *cpu, uchar_t *featureset)
 935 {
 936         uint32_t mask_ecx, mask_edx;
 937         struct cpuid_info *cpi;
 938         struct cpuid_regs *cp;
 939         int xcpuid;
 940 #if !defined(__xpv)
 941         extern int idle_cpu_prefer_mwait;
 942 #endif

 944         /*
 945          * Space statically allocated for BSP, ensure pointer is set
 946          */
 947         if (cpu->cpu_id == 0) {
 948                 if (cpu->cpu_m.mcpu_cpi == NULL)
 949                         cpu->cpu_m.mcpu_cpi = &cpuid_info0;
 950         }

 952         add_x86_feature(featureset, X86FSET_CPUID);

 954         cpi = cpu->cpu_m.mcpu_cpi;
 955         ASSERT(cpi != NULL);
 956         cp = &cpi->cpi_std[0];
 957         cp->cp_eax = 0;
 958         cpi->cpi_maxeax = __cpuid_insn(cp);
 959         {
 960                 uint32_t *iptr = (uint32_t *)cpi->cpi_vendorstr;
 961                 *iptr++ = cp->cp_ebx;
 962                 *iptr++ = cp->cp_edx;
 963                 *iptr++ = cp->cp_ecx;
 964                 *(char *)&cpi->cpi_vendorstr[12] = '\0';
 965         }

 967         cpi->cpi_vendor = _cpuid_vendorstr_to_vendorcode(cpi->cpi_vendorstr);
 968         x86_vendor = cpi->cpi_vendor; /* for compatibility */

 970         /*
 971          * Limit the range in case of weird hardware
 972          */
 973         if (cpi->cpi_maxeax > CPI_MAXEAX_MAX)
 974                 cpi->cpi_maxeax = CPI_MAXEAX_MAX;
 975         if (cpi->cpi_maxeax < 1)
 976                 goto pass1_done;

 978         cp = &cpi->cpi_std[1];
 979         cp->cp_eax = 1;
 980         (void) __cpuid_insn(cp);

 982         /*
 983          * Extract identifying constants for easy access.
 984          */
 985         cpi->cpi_model = CPI_MODEL(cpi);
```

```
 986         cpi->cpi_family = CPI_FAMILY(cpi);
 988         if (cpi->cpi_family == 0xf)
 989                 cpi->cpi_family += CPI_FAMILY_XTD(cpi);
 991         /*
 992          * Beware: AMD uses "extended model" iff base *FAMILY* == 0xf.
 993          * Intel, and presumably everyone else, uses model == 0xf, as
 994          * one would expect (max value means possible overflow).  Sigh.
 995          */
 997         switch (cpi->cpi_vendor) {
 998         case X86_VENDOR_Intel:
 999                 if (IS_EXTENDED_MODEL_INTEL(cpi))
1000                         cpi->cpi_model += CPI_MODEL_XTD(cpi) << 4;
1001                 break;
1002         case X86_VENDOR_AMD:
1003                 if (CPI_FAMILY(cpi) == 0xf)
1004                         cpi->cpi_model += CPI_MODEL_XTD(cpi) << 4;
1005                 break;
1006         default:
1007                 if (cpi->cpi_model == 0xf)
1008                         cpi->cpi_model += CPI_MODEL_XTD(cpi) << 4;
1009                 break;
1010         }
1012         cpi->cpi_step = CPI_STEP(cpi);
1013         cpi->cpi_brandid = CPI_BRANDID(cpi);
1015         /*
1016          * *default* assumptions:
1017          * - believe %edx feature word
1018          * - ignore %ecx feature word
1019          * - 32-bit virtual and physical addressing
1020          */
1021         mask_edx = 0xffffffff;
1022         mask_ecx = 0;
1024         cpi->cpi_pabits = cpi->cpi_vabits = 32;
1026         switch (cpi->cpi_vendor) {
1027         case X86_VENDOR_Intel:
1028                 if (cpi->cpi_family == 5)
1029                         x86_type = X86_TYPE_P5;
1030                 else if (IS_LEGACY_P6(cpi)) {
1031                         x86_type = X86_TYPE_P6;
1032                         pentiumpro_bug4046376 = 1;
1036                         pentiumpro_bug4064495 = 1;
1033                         /*
1034                          * Clear the SEP bit when it was set erroneously
1035                          */
1036                         if (cpi->cpi_model < 3 && cpi->cpi_step < 3)
1037                                 cp->cp_edx &= ~CPUID_INTC_EDX_SEP;
1038                 } else if (IS_NEW_F6(cpi) || cpi->cpi_family == 0xf) {
1039                         x86_type = X86_TYPE_P4;
1040                         /*
1041                          * We don't currently depend on any of the %ecx
1042                          * features until Prescott, so we'll only check
1043                          * this from P4 onwards.  We might want to revisit
1044                          * that idea later.
1045                          */
1046                         mask_ecx = 0xffffffff;
1047                 } else if (cpi->cpi_family > 0xf)
1048                         mask_ecx = 0xffffffff;
1049                 /*
1050                  * We don't support MONITOR/MWAIT if leaf 5 is not available
```



```
1051                  * to obtain the monitor linesize.
1052                  */
1053                 if (cpi->cpi_maxeax < 5)
1054                         mask_ecx &= ~CPUID_INTC_ECX_MON;
1055                 break;
1056         case X86_VENDOR_IntelClone:
1057         default:
1058                 break;
1059         case X86_VENDOR_AMD:
1060 #if defined(OPTERON_ERRATUM_108)
1061                 if (cpi->cpi_family == 0xf && cpi->cpi_model == 0xe) {
1062                         cp->cp_eax = (0xf0f & cp->cp_eax) | 0xc0;
1063                         cpi->cpi_model = 0xc;
1064                 } else
1065 #endif
1066                 if (cpi->cpi_family == 5) {
1067                         /*
1068                          * AMD K5 and K6
1069                          *
1070                          * These CPUs have an incomplete implementation
1071                          * of MCA/MCE which we mask away.
1072                          */
1073                         mask_edx &= ~(CPUID_INTC_EDX_MCE | CPUID_INTC_EDX_MCA);
1075                         /*
1076                          * Model 0 uses the wrong (APIC) bit
1077                          * to indicate PGE.  Fix it here.
1078                          */
1079                         if (cpi->cpi_model == 0) {
1080                                 if (cp->cp_edx & 0x200) {
1081                                         cp->cp_edx &= ~0x200;
1082                                         cp->cp_edx |= CPUID_INTC_EDX_PGE;
1083                                 }
1084                         }
1086                         /*
1087                          * Early models had problems w/ MMX; disable.
1088                          */
1089                         if (cpi->cpi_model < 6)
1090                                 mask_edx &= ~CPUID_INTC_EDX_MMX;
1091                 }
1093                 /*
1094                  * For newer families, SSE3 and CX16, at least, are valid;
1095                  * enable all
1096                  */
1097                 if (cpi->cpi_family >= 0xf)
1098                         mask_ecx = 0xffffffff;
1099                 /*
1100                  * We don't support MONITOR/MWAIT if leaf 5 is not available
1101                  * to obtain the monitor linesize.
1102                  */
1103                 if (cpi->cpi_maxeax < 5)
1104                         mask_ecx &= ~CPUID_INTC_ECX_MON;
1106 #if !defined(__xpv)
1107                 /*
1108                  * Do not use MONITOR/MWAIT to halt in the idle loop on any AMD
1109                  * processors.  AMD does not intend MWAIT to be used in the cpu
1110                  * idle loop on current and future processors.  10h and future
1111                  * AMD processors use more power in MWAIT than HLT.
1112                  * Pre-family-10h Opterons do not have the MWAIT instruction.
1113                  */
1114                 idle_cpu_prefer_mwait = 0;
1115 #endif
```

```
1117                break;
1118        case X86_VENDOR_TM:
1119                /*
1120                 * workaround the NT workaround in CMS 4.1
1121                 */
1122                if (cpi->cpi_family == 5 && cpi->cpi_model == 4 &&
1123                    (cpi->cpi_step == 2 || cpi->cpi_step == 3))
1124                        cp->cp_edx |= CPUID_INTC_EDX_CX8;
1125                break;
1126        case X86_VENDOR_Centaur:
1127                /*
1128                 * workaround the NT workarounds again
1129                 */
1130                if (cpi->cpi_family == 6)
1131                        cp->cp_edx |= CPUID_INTC_EDX_CX8;
1132                break;
1133        case X86_VENDOR_Cyrix:
1134                /*
1135                 * We rely heavily on the probing in locore
1136                 * to actually figure out what parts, if any,
1137                 * of the Cyrix cpuid instruction to believe.
1138                 */
1139                switch (x86_type) {
1140                case X86_TYPE_CYRIX_486:
1141                        mask_edx = 0;
1142                        break;
1143                case X86_TYPE_CYRIX_6x86:
1144                        mask_edx = 0;
1145                        break;
1146                case X86_TYPE_CYRIX_6x86L:
1147                        mask_edx =
1148                            CPUID_INTC_EDX_DE |
1149                            CPUID_INTC_EDX_CX8;
1150                        break;
1151                case X86_TYPE_CYRIX_6x86MX:
1152                        mask_edx =
1153                            CPUID_INTC_EDX_DE |
1154                            CPUID_INTC_EDX_MSR |
1155                            CPUID_INTC_EDX_CX8 |
1156                            CPUID_INTC_EDX_PGE |
1157                            CPUID_INTC_EDX_CMOV |
1158                            CPUID_INTC_EDX_MMX;
1159                        break;
1160                case X86_TYPE_CYRIX_GXm:
1161                        mask_edx =
1162                            CPUID_INTC_EDX_MSR |
1163                            CPUID_INTC_EDX_CX8 |
1164                            CPUID_INTC_EDX_CMOV |
1165                            CPUID_INTC_EDX_MMX;
1166                        break;
1167                case X86_TYPE_CYRIX_MediaGX:
1168                        break;
1169                case X86_TYPE_CYRIX_MII:
1170                case X86_TYPE_VIA_CYRIX_III:
1171                        mask_edx =
1172                            CPUID_INTC_EDX_DE |
1173                            CPUID_INTC_EDX_TSC |
1174                            CPUID_INTC_EDX_MSR |
1175                            CPUID_INTC_EDX_CX8 |
1176                            CPUID_INTC_EDX_PGE |
1177                            CPUID_INTC_EDX_CMOV |
1178                            CPUID_INTC_EDX_MMX;
1179                        break;
1180                default:
1181                        break;
1182                }
```



```
1183                break;
1184        }

1186 #if defined(__xpv)
1187        /*
1188         * Do not support MONITOR/MWAIT under a hypervisor
1189         */
1190        mask_ecx &= ~CPUID_INTC_ECX_MON;
1191        /*
1192         * Do not support XSAVE under a hypervisor for now
1193         */
1194        xsave_force_disable = B_TRUE;

1196 #endif  /* __xpv */

1198        if (xsave_force_disable) {
1199                mask_ecx &= ~CPUID_INTC_ECX_XSAVE;
1200                mask_ecx &= ~CPUID_INTC_ECX_AVX;
1201                mask_ecx &= ~CPUID_INTC_ECX_F16C;
1202        }

1204        /*
1205         * Now we've figured out the masks that determine
1206         * which bits we choose to believe, apply the masks
1207         * to the feature words, then map the kernel's view
1208         * of these feature words into its feature word.
1209         */
1210        cp->cp_edx &= mask_edx;
1211        cp->cp_ecx &= mask_ecx;

1213        /*
1214         * apply any platform restrictions (we don't call this
1215         * immediately after __cpuid_insn here, because we need the
1216         * workarounds applied above first)
1217         */
1218        platform_cpuid_mangle(cpi->cpi_vendor, 1, cp);

1220        /*
1221         * fold in overrides from the "eeprom" mechanism
1222         */
1223        cp->cp_edx |= cpuid_feature_edx_include;
1224        cp->cp_edx &= ~cpuid_feature_edx_exclude;

1226        cp->cp_ecx |= cpuid_feature_ecx_include;
1227        cp->cp_ecx &= ~cpuid_feature_ecx_exclude;

1229        if (cp->cp_edx & CPUID_INTC_EDX_PSE) {
1230                add_x86_feature(featureset, X86FSET_LARGEPAGE);
1231        }
1232        if (cp->cp_edx & CPUID_INTC_EDX_TSC) {
1233                add_x86_feature(featureset, X86FSET_TSC);
1234        }
1235        if (cp->cp_edx & CPUID_INTC_EDX_MSR) {
1236                add_x86_feature(featureset, X86FSET_MSR);
1237        }
1238        if (cp->cp_edx & CPUID_INTC_EDX_MTRR) {
1239                add_x86_feature(featureset, X86FSET_MTRR);
1240        }
1241        if (cp->cp_edx & CPUID_INTC_EDX_PGE) {
1242                add_x86_feature(featureset, X86FSET_PGE);
1243        }
1244        if (cp->cp_edx & CPUID_INTC_EDX_CMOV) {
1245                add_x86_feature(featureset, X86FSET_CMOV);
1246        }
1247        if (cp->cp_edx & CPUID_INTC_EDX_MMX) {
1248                add_x86_feature(featureset, X86FSET_MMX);
```

```
1249         }
1250         if ((cp->cp_edx & CPUID_INTC_EDX_MCE) != 0 &&
1251             (cp->cp_edx & CPUID_INTC_EDX_MCA) != 0) {
1252                 add_x86_feature(featureset, X86FSET_MCA);
1253         }
1254         if (cp->cp_edx & CPUID_INTC_EDX_PAE) {
1255                 add_x86_feature(featureset, X86FSET_PAE);
1256         }
1257         if (cp->cp_edx & CPUID_INTC_EDX_CX8) {
1258                 add_x86_feature(featureset, X86FSET_CX8);
1259         }
1260         if (cp->cp_ecx & CPUID_INTC_ECX_CX16) {
1261                 add_x86_feature(featureset, X86FSET_CX16);
1262         }
1263         if (cp->cp_edx & CPUID_INTC_EDX_PAT) {
1264                 add_x86_feature(featureset, X86FSET_PAT);
1265         }
1266         if (cp->cp_edx & CPUID_INTC_EDX_SEP) {
1267                 add_x86_feature(featureset, X86FSET_SEP);
1268         }
1269         if (cp->cp_edx & CPUID_INTC_EDX_FXSR) {
1270                 /*
1271                  * In our implementation, fxsave/fxrstor
1272                  * are prerequisites before we'll even
1273                  * try and do SSE things.
1274                  */
1275                 if (cp->cp_edx & CPUID_INTC_EDX_SSE) {
1276                         add_x86_feature(featureset, X86FSET_SSE);
1277                 }
1278                 if (cp->cp_edx & CPUID_INTC_EDX_SSE2) {
1279                         add_x86_feature(featureset, X86FSET_SSE2);
1280                 }
1281                 if (cp->cp_ecx & CPUID_INTC_ECX_SSE3) {
1282                         add_x86_feature(featureset, X86FSET_SSE3);
1283                 }
1284                 if (cp->cp_ecx & CPUID_INTC_ECX_SSSE3) {
1285                         add_x86_feature(featureset, X86FSET_SSSE3);
1286                 }
1287                 if (cp->cp_ecx & CPUID_INTC_ECX_SSE4_1) {
1288                         add_x86_feature(featureset, X86FSET_SSE4_1);
1289                 }
1290                 if (cp->cp_ecx & CPUID_INTC_ECX_SSE4_2) {
1291                         add_x86_feature(featureset, X86FSET_SSE4_2);
1292                 }
1293                 if (cp->cp_ecx & CPUID_INTC_ECX_AES) {
1294                         add_x86_feature(featureset, X86FSET_AES);
1295                 }
1296                 if (cp->cp_ecx & CPUID_INTC_ECX_PCLMULQDQ) {
1297                         add_x86_feature(featureset, X86FSET_PCLMULQDQ);
1298                 }

1300                 if (cp->cp_ecx & CPUID_INTC_ECX_XSAVE) {
1301                         add_x86_feature(featureset, X86FSET_XSAVE);

1303                         /* We only test AVX when there is XSAVE */
1304                         if (cp->cp_ecx & CPUID_INTC_ECX_AVX) {
1305                                 add_x86_feature(featureset,
1306                                     X86FSET_AVX);

1308                                 if (cp->cp_ecx & CPUID_INTC_ECX_F16C)
1309                                         add_x86_feature(featureset,
1310                                             X86FSET_F16C);
1311                         }
1312                 }
1313         }
1314         if (cp->cp_edx & CPUID_INTC_EDX_DE) {
```



```
1315                 add_x86_feature(featureset, X86FSET_DE);
1316         }
1317 #if !defined(__xpv)
1318         if (cp->cp_ecx & CPUID_INTC_ECX_MON) {

1320                 /*
1321                  * We require the CLFLUSH instruction for erratum workaround
1322                  * to use MONITOR/MWAIT.
1323                  */
1324                 if (cp->cp_edx & CPUID_INTC_EDX_CLFSH) {
1325                         cpi->cpi_mwait.support |= MWAIT_SUPPORT;
1326                         add_x86_feature(featureset, X86FSET_MWAIT);
1327                 } else {
1328                         extern int idle_cpu_assert_cflush_monitor;

1330                         /*
1331                          * All processors we are aware of which have
1332                          * MONITOR/MWAIT also have CLFLUSH.
1333                          */
1334                         if (idle_cpu_assert_cflush_monitor) {
1335                                 ASSERT((cp->cp_ecx & CPUID_INTC_ECX_MON) &&
1336                                     (cp->cp_edx & CPUID_INTC_EDX_CLFSH));
1337                         }
1338                 }
1339         }
1340 #endif  /* __xpv */

1342         if (cp->cp_ecx & CPUID_INTC_ECX_VMX) {
1343                 add_x86_feature(featureset, X86FSET_VMX);
1344         }

1346         if (cp->cp_ecx & CPUID_INTC_ECX_RDRAND)
1347                 add_x86_feature(featureset, X86FSET_RDRAND);

1349         /*
1350          * Only need it first time, rest of the cpus would follow suit.
1351          * we only capture this for the bootcpu.
1352          */
1353         if (cp->cp_edx & CPUID_INTC_EDX_CLFSH) {
1354                 add_x86_feature(featureset, X86FSET_CLFSH);
1355                 x86_clflush_size = (BITX(cp->cp_ebx, 15, 8) * 8);
1356         }
1357         if (is_x86_feature(featureset, X86FSET_PAE))
1358                 cpi->cpi_pabits = 36;

1360         /*
1361          * Hyperthreading configuration is slightly tricky on Intel
1362          * and pure clones, and even trickier on AMD.
1363          *
1364          * (AMD chose to set the HTT bit on their CMP processors,
1365          * even though they're not actually hyperthreaded.  Thus it
1366          * takes a bit more work to figure out what's really going
1367          * on ... see the handling of the CMP_LGCY bit below)
1368          */
1369         if (cp->cp_edx & CPUID_INTC_EDX_HTT) {
1370                 cpi->cpi_ncpu_per_chip = CPI_CPU_COUNT(cpi);
1371                 if (cpi->cpi_ncpu_per_chip > 1)
1372                         add_x86_feature(featureset, X86FSET_HTT);
1373         } else {
1374                 cpi->cpi_ncpu_per_chip = 1;
1375         }

1377         /*
1378          * Work on the "extended" feature information, doing
1379          * some basic initialization for cpuid_pass2()
1380          */
```

```
1381         xcpuid = 0;
1382         switch (cpi->cpi_vendor) {
1383         case X86_VENDOR_Intel:
1384                 if (IS_NEW_F6(cpi) || cpi->cpi_family >= 0xf)
1385                         xcpuid++;
1386                 break;
1387         case X86_VENDOR_AMD:
1388                 if (cpi->cpi_family > 5 ||
1389                     (cpi->cpi_family == 5 && cpi->cpi_model >= 1))
1390                         xcpuid++;
1391                 break;
1392         case X86_VENDOR_Cyrix:
1393                 /*
1394                  * Only these Cyrix CPUs are -known- to support
1395                  * extended cpuid operations.
1396                  */
1397                 if (x86_type == X86_TYPE_VIA_CYRIX_III ||
1398                     x86_type == X86_TYPE_CYRIX_GXm)
1399                         xcpuid++;
1400                 break;
1401         case X86_VENDOR_Centaur:
1402         case X86_VENDOR_TM:
1403         default:
1404                 xcpuid++;
1405                 break;
1406         }

1408         if (xcpuid) {
1409                 cp = &cpi->cpi_extd[0];
1410                 cp->cp_eax = 0x80000000;
1411                 cpi->cpi_xmaxeax = __cpuid_insn(cp);
1412         }

1414         if (cpi->cpi_xmaxeax & 0x80000000) {

1416                 if (cpi->cpi_xmaxeax > CPI_XMAXEAX_MAX)
1417                         cpi->cpi_xmaxeax = CPI_XMAXEAX_MAX;

1419                 switch (cpi->cpi_vendor) {
1420                 case X86_VENDOR_Intel:
1421                 case X86_VENDOR_AMD:
1422                         if (cpi->cpi_xmaxeax < 0x80000001)
1423                                 break;
1424                         cp = &cpi->cpi_extd[1];
1425                         cp->cp_eax = 0x80000001;
1426                         (void) __cpuid_insn(cp);

1428                         if (cpi->cpi_vendor == X86_VENDOR_AMD &&
1429                             cpi->cpi_family == 5 &&
1430                             cpi->cpi_model == 6 &&
1431                             cpi->cpi_step == 6) {
1432                                 /*
1433                                  * K6 model 6 uses bit 10 to indicate SYSC
1434                                  * Later models use bit 11. Fix it here.
1435                                  */
1436                                 if (cp->cp_edx & 0x400) {
1437                                         cp->cp_edx &= ~0x400;
1438                                         cp->cp_edx |= CPUID_AMD_EDX_SYSC;
1439                                 }
1440                         }

1442                         platform_cpuid_mangle(cpi->cpi_vendor, 0x80000001, cp);

1444                         /*
1445                          * Compute the additions to the kernel's feature word.
1446                          */
```



```
1447                         if (cp->cp_edx & CPUID_AMD_EDX_NX) {
1448                                 add_x86_feature(featureset, X86FSET_NX);
1449                         }

1451                         /*
1452                          * Regardless whether or not we boot 64-bit,
1453                          * we should have a way to identify whether
1454                          * the CPU is capable of running 64-bit.
1455                          */
1456                         if (cp->cp_edx & CPUID_AMD_EDX_LM) {
1457                                 add_x86_feature(featureset, X86FSET_64);
1458                         }

1460 #if defined(__amd64)
1461                         /* 1 GB large page - enable only for 64 bit kernel */
1462                         if (cp->cp_edx & CPUID_AMD_EDX_1GPG) {
1463                                 add_x86_feature(featureset, X86FSET_1GPG);
1464                         }
1465 #endif

1467                         if ((cpi->cpi_vendor == X86_VENDOR_AMD) &&
1468                             (cpi->cpi_std[1].cp_edx & CPUID_INTC_EDX_FXSR) &&
1469                             (cp->cp_ecx & CPUID_AMD_ECX_SSE4A)) {
1470                                 add_x86_feature(featureset, X86FSET_SSE4A);
1471                         }

1473                         /*
1474                          * If both the HTT and CMP_LGCY bits are set,
1475                          * then we're not actually HyperThreaded.  Read
1476                          * "AMD CPUID Specification" for more details.
1477                          */
1478                         if (cpi->cpi_vendor == X86_VENDOR_AMD &&
1479                             is_x86_feature(featureset, X86FSET_HTT) &&
1480                             (cp->cp_ecx & CPUID_AMD_ECX_CMP_LGCY)) {
1481                                 remove_x86_feature(featureset, X86FSET_HTT);
1482                                 add_x86_feature(featureset, X86FSET_CMP);
1483                         }
1484 #if defined(__amd64)
1485                         /*
1486                          * It's really tricky to support syscall/sysret in
1487                          * the i386 kernel; we rely on sysenter/sysexit
1488                          * instead.  In the amd64 kernel, things are -way-
1489                          * better.
1490                          */
1491                         if (cp->cp_edx & CPUID_AMD_EDX_SYSC) {
1492                                 add_x86_feature(featureset, X86FSET_ASYSC);
1493                         }

1495                         /*
1496                          * While we're thinking about system calls, note
1497                          * that AMD processors don't support sysenter
1498                          * in long mode at all, so don't try to program them.
1499                          */
1500                         if (x86_vendor == X86_VENDOR_AMD) {
1501                                 remove_x86_feature(featureset, X86FSET_SEP);
1502                         }
1503 #endif
1504                         if (cp->cp_edx & CPUID_AMD_EDX_TSCP) {
1505                                 add_x86_feature(featureset, X86FSET_TSCP);
1506                         }

1508                         if (cp->cp_ecx & CPUID_AMD_ECX_SVM) {
1509                                 add_x86_feature(featureset, X86FSET_SVM);
1510                         }

1512                         if (cp->cp_ecx & CPUID_AMD_ECX_TOPOEXT) {
```

```
1513                                    add_x86_feature(featureset, X86FSET_TOPOEXT);
1514                            }
1515                            break;
1516                    default:
1517                            break;
1518                    }

1520                    /*
1521                     * Get CPUID data about processor cores and hyperthreads.
1522                     */
1523                    switch (cpi->cpi_vendor) {
1524                    case X86_VENDOR_Intel:
1525                            if (cpi->cpi_maxeax >= 4) {
1526                                    cp = &cpi->cpi_std[4];
1527                                    cp->cp_eax = 4;
1528                                    cp->cp_ecx = 0;
1529                                    (void) __cpuid_insn(cp);
1530                                    platform_cpuid_mangle(cpi->cpi_vendor, 4, cp);
1531                            }
1532                            /*FALLTHROUGH*/
1533                    case X86_VENDOR_AMD:
1534                            if (cpi->cpi_xmaxeax < 0x80000008)
1535                                    break;
1536                            cp = &cpi->cpi_extd[8];
1537                            cp->cp_eax = 0x80000008;
1538                            (void) __cpuid_insn(cp);
1539                            platform_cpuid_mangle(cpi->cpi_vendor, 0x80000008, cp);

1541                            /*
1542                             * Virtual and physical address limits from
1543                             * cpuid override previously guessed values.
1544                             */
1545                            cpi->cpi_pabits = BITX(cp->cp_eax, 7, 0);
1546                            cpi->cpi_vabits = BITX(cp->cp_eax, 15, 8);
1547                            break;
1548                    default:
1549                            break;
1550                    }

1552                    /*
1553                     * Derive the number of cores per chip
1554                     */
1555                    switch (cpi->cpi_vendor) {
1556                    case X86_VENDOR_Intel:
1557                            if (cpi->cpi_maxeax < 4) {
1558                                    cpi->cpi_ncore_per_chip = 1;
1559                                    break;
1560                            } else {
1561                                    cpi->cpi_ncore_per_chip =
1562                                        BITX((cpi)->cpi_std[4].cp_eax, 31, 26) + 1;
1563                            }
1564                            break;
1565                    case X86_VENDOR_AMD:
1566                            if (cpi->cpi_xmaxeax < 0x80000008) {
1567                                    cpi->cpi_ncore_per_chip = 1;
1568                                    break;
1569                            } else {
1570                                    /*
1571                                     * On family 0xf cpuid fn 2 ECX[7:0] "NC" is
1572                                     * 1 less than the number of physical cores on
1573                                     * the chip.  In family 0x10 this value can
1574                                     * be affected by "downcoring" - it reflects
1575                                     * 1 less than the number of cores actually
1576                                     * enabled on this node.
1577                                     */
1578                                    cpi->cpi_ncore_per_chip =
```



```
1579                                        BITX((cpi)->cpi_extd[8].cp_ecx, 7, 0) + 1;
1580                            }
1581                            break;
1582                    default:
1583                            cpi->cpi_ncore_per_chip = 1;
1584                            break;
1585                    }

1587                    /*
1588                     * Get CPUID data about TSC Invariance in Deep C-State.
1589                     */
1590                    switch (cpi->cpi_vendor) {
1591                    case X86_VENDOR_Intel:
1592                            if (cpi->cpi_maxeax >= 7) {
1593                                    cp = &cpi->cpi_extd[7];
1594                                    cp->cp_eax = 0x80000007;
1595                                    cp->cp_ecx = 0;
1596                                    (void) __cpuid_insn(cp);
1597                            }
1598                            break;
1599                    default:
1600                            break;
1601                    }
1602            } else {
1603                    cpi->cpi_ncore_per_chip = 1;
1604            }

1606            /*
1607             * If more than one core, then this processor is CMP.
1608             */
1609            if (cpi->cpi_ncore_per_chip > 1) {
1610                    add_x86_feature(featureset, X86FSET_CMP);
1611            }

1613            /*
1614             * If the number of cores is the same as the number
1615             * of CPUs, then we cannot have HyperThreading.
1616             */
1617            if (cpi->cpi_ncpu_per_chip == cpi->cpi_ncore_per_chip) {
1618                    remove_x86_feature(featureset, X86FSET_HTT);
1619            }

1621            cpi->cpi_apicid = CPI_APIC_ID(cpi);
1622            cpi->cpi_procnodes_per_pkg = 1;
1623            cpi->cpi_cores_per_compunit = 1;
1624            if (is_x86_feature(featureset, X86FSET_HTT) == B_FALSE &&
1625                is_x86_feature(featureset, X86FSET_CMP) == B_FALSE) {
1626                    /*
1627                     * Single-core single-threaded processors.
1628                     */
1629                    cpi->cpi_chipid = -1;
1630                    cpi->cpi_clogid = 0;
1631                    cpi->cpi_coreid = cpu->cpu_id;
1632                    cpi->cpi_pkgcoreid = 0;
1633                    if (cpi->cpi_vendor == X86_VENDOR_AMD)
1634                            cpi->cpi_procnodeid = BITX(cpi->cpi_apicid, 3, 0);
1635                    else
1636                            cpi->cpi_procnodeid = cpi->cpi_chipid;
1637            } else if (cpi->cpi_ncpu_per_chip > 1) {
1638                    if (cpi->cpi_vendor == X86_VENDOR_Intel)
1639                            cpuid_intel_getids(cpu, featureset);
1640                    else if (cpi->cpi_vendor == X86_VENDOR_AMD)
1641                            cpuid_amd_getids(cpu);
1642                    else {
1643                            /*
1644                             * All other processors are currently
```

```
1645                          * assumed to have single cores.
1646                          */
1647                         cpi->cpi_coreid = cpi->cpi_chipid;
1648                         cpi->cpi_pkgcoreid = 0;
1649                         cpi->cpi_procnodeid = cpi->cpi_chipid;
1650                         cpi->cpi_compunitid = cpi->cpi_chipid;
1651                 }
1652         }

1654         /*
1655          * Synthesize chip "revision" and socket type
1656          */
1657         cpi->cpi_chiprev = _cpuid_chiprev(cpi->cpi_vendor, cpi->cpi_family,
1658             cpi->cpi_model, cpi->cpi_step);
1659         cpi->cpi_chiprevstr = _cpuid_chiprevstr(cpi->cpi_vendor,
1660             cpi->cpi_family, cpi->cpi_model, cpi->cpi_step);
1661         cpi->cpi_socket = _cpuid_skt(cpi->cpi_vendor, cpi->cpi_family,
1662             cpi->cpi_model, cpi->cpi_step);

1664 pass1_done:
1665         cpi->cpi_pass = 1;
1666 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   28905 Tue Jan  7 20:34:25 2014
new/usr/src/uts/intel/sys/x86_archext.h
4444 remove unused cpuid-related globals
**********************************************************
```

```
/*
 * 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) 1995, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2011 by Delphix. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */
/*
 * Copyright (c) 2012, Joyent, Inc. All rights reserved.
 * Copyright 2012 Jens Elkner <jel+illumos@cs.uni-magdeburg.de>
 * Copyright 2012 Hans Rosenfeld <rosenfeld@grumpf.hope-2000.org>
 */

#ifndef _SYS_X86_ARCHEXT_H
#define _SYS_X86_ARCHEXT_H

#if !defined(_ASM)
#include <sys/regset.h>
#include <sys/processor.h>
#include <vm/seg_enum.h>
#include <vm/page.h>
#endif  /* _ASM */

#ifdef  __cplusplus
extern "C" {
#endif

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

#define CPUID_INTC_EDX_FPU      0x00000001      /* x87 fpu present */
#define CPUID_INTC_EDX_VME      0x00000002      /* virtual-8086 extension */
#define CPUID_INTC_EDX_DE       0x00000004      /* debugging extensions */
#define CPUID_INTC_EDX_PSE      0x00000008      /* page size extension */
#define CPUID_INTC_EDX_TSC      0x00000010      /* time stamp counter */
#define CPUID_INTC_EDX_MSR      0x00000020      /* rdmsr and wrmsr */
#define CPUID_INTC_EDX_PAE      0x00000040      /* physical addr extension */
#define CPUID_INTC_EDX_MCE      0x00000080      /* machine check exception */
```



```
#define CPUID_INTC_EDX_CX8      0x00000100      /* cmpxchg8b instruction */
#define CPUID_INTC_EDX_APIC     0x00000200      /* local APIC */
                                                /* 0x400 - reserved */
#define CPUID_INTC_EDX_SEP      0x00000800      /* sysenter and sysexit */
#define CPUID_INTC_EDX_MTRR     0x00001000      /* memory type range reg */
#define CPUID_INTC_EDX_PGE      0x00002000      /* page global enable */
#define CPUID_INTC_EDX_MCA      0x00004000      /* machine check arch */
#define CPUID_INTC_EDX_CMOV     0x00008000      /* conditional move insns */
#define CPUID_INTC_EDX_PAT      0x00010000      /* page attribute table */
#define CPUID_INTC_EDX_PSE36    0x00020000      /* 36-bit pagesize extension */
#define CPUID_INTC_EDX_PSN      0x00040000      /* processor serial number */
#define CPUID_INTC_EDX_CLFSH    0x00080000      /* clflush instruction */
                                                /* 0x100000 - reserved */
#define CPUID_INTC_EDX_DS       0x00200000      /* debug store exists */
#define CPUID_INTC_EDX_ACPI     0x00400000      /* monitoring + clock ctrl */
#define CPUID_INTC_EDX_MMX      0x00800000      /* MMX instructions */
#define CPUID_INTC_EDX_FXSR     0x01000000      /* fxsave and fxrstor */
#define CPUID_INTC_EDX_SSE      0x02000000      /* streaming SIMD extensions */
#define CPUID_INTC_EDX_SSE2     0x04000000      /* SSE extensions */
#define CPUID_INTC_EDX_SS       0x08000000      /* self-snoop */
#define CPUID_INTC_EDX_HTT      0x10000000      /* Hyper Thread Technology */
#define CPUID_INTC_EDX_TM       0x20000000      /* thermal monitoring */
#define CPUID_INTC_EDX_IA64     0x40000000      /* Itanium emulating IA32 */
#define CPUID_INTC_EDX_PBE      0x80000000      /* Pending Break Enable */

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

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

#define CPUID_INTC_ECX_SSE3     0x00000001      /* Yet more SSE extensions */
#define CPUID_INTC_ECX_PCLMULQDQ 0x00000002     /* PCLMULQDQ insn */
                                                /* 0x00000004 - reserved */
#define CPUID_INTC_ECX_MON      0x00000008      /* MONITOR/MWAIT */
#define CPUID_INTC_ECX_DSCPL    0x00000010      /* CPL-qualified debug store */
#define CPUID_INTC_ECX_VMX      0x00000020      /* Hardware VM extensions */
#define CPUID_INTC_ECX_SMX      0x00000040      /* Secure mode extensions */
#define CPUID_INTC_ECX_EST      0x00000080      /* enhanced SpeedStep */
#define CPUID_INTC_ECX_TM2      0x00000100      /* thermal monitoring */
#define CPUID_INTC_ECX_SSSE3    0x00000200      /* Supplemental SSE3 insns */
#define CPUID_INTC_ECX_CID      0x00000400      /* L1 context ID */
                                                /* 0x00000800 - reserved */
                                                /* 0x00001000 - reserved */
#define CPUID_INTC_ECX_CX16     0x00002000      /* cmpxchg16 */
#define CPUID_INTC_ECX_ETPRD    0x00004000      /* extended task pri messages */
                                                /* 0x00008000 - reserved */
                                                /* 0x00010000 - reserved */
                                                /* 0x00020000 - reserved */
#define CPUID_INTC_ECX_DCA      0x00040000      /* direct cache access */
#define CPUID_INTC_ECX_SSE4_1   0x00080000      /* SSE4.1 insns */
#define CPUID_INTC_ECX_SSE4_2   0x00100000      /* SSE4.2 insns */
#define CPUID_INTC_ECX_MOVBE    0x00400000      /* MOVBE insn */
#define CPUID_INTC_ECX_POPCNT   0x00800000      /* POPCNT insn */
#define CPUID_INTC_ECX_AES      0x02000000      /* AES insns */
#define CPUID_INTC_ECX_XSAVE    0x04000000      /* XSAVE/XRESTOR insns */
#define CPUID_INTC_ECX_OSXSAVE  0x08000000      /* OS supports XSAVE insns */
#define CPUID_INTC_ECX_AVX      0x10000000      /* AVX supported */
#define CPUID_INTC_ECX_F16C     0x20000000      /* F16C supported */
#define CPUID_INTC_ECX_RDRAND   0x40000000      /* RDRAND supported */
#define CPUID_INTC_ECX_HV       0x80000000      /* Hypervisor */
```

```
#define FMT_CPUID_INTC_ECX                                      \
        "\20"                                                   \
        "\37rdrand\36f16c\35avx\34osxsav\33xsave"               \
        "\32aes"                                                \
        "\30popcnt\27movbe\25sse4.2\24sse4.1\23dca"             \
        "\20\17etprd\16cx16\13cid\12ssse3\11tm2"                \
        "\10est\7smx\6vmx\5dscpl\4mon\2pclmulqdq\1sse3"

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

#define CPUID_AMD_EDX_FPU       0x00000001      /* x87 fpu present */
#define CPUID_AMD_EDX_VME       0x00000002      /* virtual-8086 extension */
#define CPUID_AMD_EDX_DE        0x00000004      /* debugging extensions */
#define CPUID_AMD_EDX_PSE       0x00000008      /* page size extensions */
#define CPUID_AMD_EDX_TSC       0x00000010      /* time stamp counter */
#define CPUID_AMD_EDX_MSR       0x00000020      /* rdmsr and wrmsr */
#define CPUID_AMD_EDX_PAE       0x00000040      /* physical addr extension */
#define CPUID_AMD_EDX_MCE       0x00000080      /* machine check exception */
#define CPUID_AMD_EDX_CX8       0x00000100      /* cmpxchg8b instruction */
#define CPUID_AMD_EDX_APIC      0x00000200      /* local APIC */
                                                /* 0x00000400 - sysc on K6m6 */
#define CPUID_AMD_EDX_SYSC      0x00000800      /* AMD: syscall and sysret */
#define CPUID_AMD_EDX_MTRR      0x00001000      /* memory type and range reg */
#define CPUID_AMD_EDX_PGE       0x00002000      /* page global enable */
#define CPUID_AMD_EDX_MCA       0x00004000      /* machine check arch */
#define CPUID_AMD_EDX_CMOV      0x00008000      /* conditional move insns */
#define CPUID_AMD_EDX_PAT       0x00010000      /* K7: page attribute table */
#define CPUID_AMD_EDX_FCMOV     0x00010000      /* FCMOVcc etc. */
#define CPUID_AMD_EDX_PSE36     0x00020000      /* 36-bit pagesize extension */
                                /* 0x00040000 - reserved */
                                /* 0x00080000 - reserved */
#define CPUID_AMD_EDX_NX        0x00100000      /* AMD: no-execute page prot */
                                /* 0x00200000 - reserved */
#define CPUID_AMD_EDX_MMXamd    0x00400000      /* AMD: MMX extensions */
#define CPUID_AMD_EDX_MMX       0x00800000      /* MMX instructions */
#define CPUID_AMD_EDX_FXSR      0x01000000      /* fxsave and fxrstor */
#define CPUID_AMD_EDX_FFXSR     0x02000000      /* fast fxsave/fxrstor */
#define CPUID_AMD_EDX_1GPG      0x04000000      /* 1GB page */
#define CPUID_AMD_EDX_TSCP      0x08000000      /* rdtscp instruction */
                                /* 0x10000000 - reserved */
#define CPUID_AMD_EDX_LM        0x20000000      /* AMD: long mode */
#define CPUID_AMD_EDX_3DNowx    0x40000000      /* AMD: extensions to 3DNow! */
#define CPUID_AMD_EDX_3DNow     0x80000000      /* AMD: 3DNow! instructions */

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

#define CPUID_AMD_ECX_AHF64     0x00000001      /* LAHF and SAHF in long mode */
#define CPUID_AMD_ECX_CMP_LGCY  0x00000002      /* AMD: multicore chip */
#define CPUID_AMD_ECX_SVM       0x00000004      /* AMD: secure VM */
#define CPUID_AMD_ECX_EAS       0x00000008      /* extended apic space */
#define CPUID_AMD_ECX_CR8D      0x00000010      /* AMD: 32-bit mov %cr8 */
#define CPUID_AMD_ECX_LZCNT     0x00000020      /* AMD: LZCNT insn */
#define CPUID_AMD_ECX_SSE4A     0x00000040      /* AMD: SSE4A insns */
#define CPUID_AMD_ECX_MAS       0x00000080      /* AMD: MisAlignSse mnode */
#define CPUID_AMD_ECX_3DNP      0x00000100      /* AMD: 3DNowPrefectch */
#define CPUID_AMD_ECX_OSVW      0x00000200      /* AMD: OSVW */
#define CPUID_AMD_ECX_IBS       0x00000400      /* AMD: IBS */
#define CPUID_AMD_ECX_SSE5      0x00000800      /* AMD: SSE5 */
```



```
#define CPUID_AMD_ECX_SKINIT    0x00001000      /* AMD: SKINIT */
#define CPUID_AMD_ECX_WDT       0x00002000      /* AMD: WDT */
#define CPUID_AMD_ECX_TOPOEXT   0x00400000      /* AMD: Topology Extensions */

#define FMT_CPUID_AMD_ECX                                       \
        "\20"                                                   \
        "\22topoext"                                            \
        "\14wdt\13skinit\12sse5\11ibs\10osvw\93dnp\8mas"        \
        "\7sse4a\6lzcnt\5cr8d\3svm\2lcmplgcy\1ahf64"

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


#define P5_MCHADDR      0x0
#define P5_CESR         0x11
#define P5_CTR0         0x12
#define P5_CTR1         0x13

#define K5_MCHADDR      0x0
#define K5_MCHTYPE      0x01
#define K5_TSC          0x10
#define K5_TR12         0x12

#define REG_PAT         0x277

#define REG_MC0_CTL             0x400
#define REG_MC5_MISC            0x417
#define REG_PERFCTR0            0xc1
#define REG_PERFCTR1            0xc2

#define REG_PERFEVNT0           0x186
#define REG_PERFEVNT1           0x187

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

#if !defined(__xpv)
/*
 * AMD C1E
 */
#define MSR_AMD_INT_PENDING_CMP_HALT    0xC0010055
#define AMD_ACTONCMPHALT_SHIFT  27
#define AMD_ACTONCMPHALT_MASK   3
#endif

#define MSR_DEBUGCTL            0x1d9

#define DEBUGCTL_LBR            0x01
#define DEBUGCTL_BTF            0x02

/* Intel P6, AMD */
#define MSR_LBR_FROM            0x1db
#define MSR_LBR_TO              0x1dc
#define MSR_LEX_FROM            0x1dd
#define MSR_LEX_TO              0x1de

/* Intel P4 (pre-Prescott, non P4 M) */
#define MSR_P4_LBSTK_TOS        0x1da
```

```
#define MSR_P4_LBSTK_0          0x1db
#define MSR_P4_LBSTK_1          0x1dc
#define MSR_P4_LBSTK_2          0x1dd
#define MSR_P4_LBSTK_3          0x1de

/* Intel Pentium M */
#define MSR_P6M_LBSTK_TOS       0x1c9
#define MSR_P6M_LBSTK_0         0x040
#define MSR_P6M_LBSTK_1         0x041
#define MSR_P6M_LBSTK_2         0x042
#define MSR_P6M_LBSTK_3         0x043
#define MSR_P6M_LBSTK_4         0x044
#define MSR_P6M_LBSTK_5         0x045
#define MSR_P6M_LBSTK_6         0x046
#define MSR_P6M_LBSTK_7         0x047

/* Intel P4 (Prescott) */
#define MSR_PRP4_LBSTK_TOS      0x1da
#define MSR_PRP4_LBSTK_FROM_0   0x680
#define MSR_PRP4_LBSTK_FROM_1   0x681
#define MSR_PRP4_LBSTK_FROM_2   0x682
#define MSR_PRP4_LBSTK_FROM_3   0x683
#define MSR_PRP4_LBSTK_FROM_4   0x684
#define MSR_PRP4_LBSTK_FROM_5   0x685
#define MSR_PRP4_LBSTK_FROM_6   0x686
#define MSR_PRP4_LBSTK_FROM_7   0x687
#define MSR_PRP4_LBSTK_FROM_8   0x688
#define MSR_PRP4_LBSTK_FROM_9   0x689
#define MSR_PRP4_LBSTK_FROM_10  0x68a
#define MSR_PRP4_LBSTK_FROM_11  0x68b
#define MSR_PRP4_LBSTK_FROM_12  0x68c
#define MSR_PRP4_LBSTK_FROM_13  0x68d
#define MSR_PRP4_LBSTK_FROM_14  0x68e
#define MSR_PRP4_LBSTK_FROM_15  0x68f
#define MSR_PRP4_LBSTK_TO_0     0x6c0
#define MSR_PRP4_LBSTK_TO_1     0x6c1
#define MSR_PRP4_LBSTK_TO_2     0x6c2
#define MSR_PRP4_LBSTK_TO_3     0x6c3
#define MSR_PRP4_LBSTK_TO_4     0x6c4
#define MSR_PRP4_LBSTK_TO_5     0x6c5
#define MSR_PRP4_LBSTK_TO_6     0x6c6
#define MSR_PRP4_LBSTK_TO_7     0x6c7
#define MSR_PRP4_LBSTK_TO_8     0x6c8
#define MSR_PRP4_LBSTK_TO_9     0x6c9
#define MSR_PRP4_LBSTK_TO_10    0x6ca
#define MSR_PRP4_LBSTK_TO_11    0x6cb
#define MSR_PRP4_LBSTK_TO_12    0x6cc
#define MSR_PRP4_LBSTK_TO_13    0x6cd
#define MSR_PRP4_LBSTK_TO_14    0x6ce
#define MSR_PRP4_LBSTK_TO_15    0x6cf

#define MCI_CTL_VALUE           0xffffffff

#define MTRR_TYPE_UC            0
#define MTRR_TYPE_WC            1
#define MTRR_TYPE_WT            4
#define MTRR_TYPE_WP            5
#define MTRR_TYPE_WB            6
#define MTRR_TYPE_UC_           7

/*
 * For Solaris we set up the page attritubute table in the following way:
 * PAT0 Write-Back
 * PAT1 Write-Through
 * PAT2 Unchacheable-
 * PAT3 Uncacheable
```



```
 * PAT4 Write-Back
 * PAT5 Write-Through
 * PAT6 Write-Combine
 * PAT7 Uncacheable
 * The only difference from h/w default is entry 6.
 */
#define PAT_DEFAULT_ATTRIBUTE                   \
        ((uint64_t)MTRR_TYPE_WB |               \
        ((uint64_t)MTRR_TYPE_WT << 8) |         \
        ((uint64_t)MTRR_TYPE_UC_ << 16) |       \
        ((uint64_t)MTRR_TYPE_UC << 24) |        \
        ((uint64_t)MTRR_TYPE_WB << 32) |        \
        ((uint64_t)MTRR_TYPE_WT << 40) |        \
        ((uint64_t)MTRR_TYPE_WC << 48) |        \
        ((uint64_t)MTRR_TYPE_UC << 56))

#define X86FSET_LARGEPAGE       0
#define X86FSET_TSC             1
#define X86FSET_MSR             2
#define X86FSET_MTRR            3
#define X86FSET_PGE             4
#define X86FSET_DE              5
#define X86FSET_CMOV            6
#define X86FSET_MMX             7
#define X86FSET_MCA             8
#define X86FSET_PAE             9
#define X86FSET_CX8             10
#define X86FSET_PAT             11
#define X86FSET_SEP             12
#define X86FSET_SSE             13
#define X86FSET_SSE2            14
#define X86FSET_HTT             15
#define X86FSET_ASYSC           16
#define X86FSET_NX              17
#define X86FSET_SSE3            18
#define X86FSET_CX16            19
#define X86FSET_CMP             20
#define X86FSET_TSCP            21
#define X86FSET_MWAIT           22
#define X86FSET_SSE4A           23
#define X86FSET_CPUID           24
#define X86FSET_SSSE3           25
#define X86FSET_SSE4_1          26
#define X86FSET_SSE4_2          27
#define X86FSET_1GPG            28
#define X86FSET_CLFSH           29
#define X86FSET_64              30
#define X86FSET_AES             31
#define X86FSET_PCLMULQDQ       32
#define X86FSET_XSAVE           33
#define X86FSET_AVX             34
#define X86FSET_VMX             35
#define X86FSET_SVM             36
#define X86FSET_TOPOEXT         37
#define X86FSET_F16C            38
#define X86FSET_RDRAND          39

/*
 * flags to patch tsc_read routine.
 */
#define X86_NO_TSC              0x0
#define X86_HAVE_TSCP           0x1
#define X86_TSC_MFENCE          0x2
#define X86_TSC_LFENCE          0x4

/*
```

```
 392  * Intel Deep C-State invariant TSC in leaf 0x80000007.
 393  */
 394 #define CPUID_TSC_CSTATE_INVARIANCE     (0x100)

 396 /*
 397  * Intel Deep C-state always-running local APIC timer
 398  */
 399 #define CPUID_CSTATE_ARAT       (0x4)

 401 /*
 402  * Intel ENERGY_PERF_BIAS MSR indicated by feature bit CPUID.6.ECX[3].
 403  */
 404 #define CPUID_EPB_SUPPORT       (1 << 3)

 406 /*
 407  * Intel TSC deadline timer
 408  */
 409 #define CPUID_DEADLINE_TSC      (1 << 24)

 411 /*
 412  * x86_type is a legacy concept; this is supplanted
 413  * for most purposes by x86_featureset; modern CPUs
 414  * should be X86_TYPE_OTHER
 415  */
 416 #define X86_TYPE_OTHER          0
 417 #define X86_TYPE_486            1
 418 #define X86_TYPE_P5             2
 419 #define X86_TYPE_P6             3
 420 #define X86_TYPE_CYRIX_486      4
 421 #define X86_TYPE_CYRIX_6x86L    5
 422 #define X86_TYPE_CYRIX_6x86     6
 423 #define X86_TYPE_CYRIX_GXm      7
 424 #define X86_TYPE_CYRIX_6x86MX   8
 425 #define X86_TYPE_CYRIX_MediaGX  9
 426 #define X86_TYPE_CYRIX_MII      10
 427 #define X86_TYPE_VIA_CYRIX_III  11
 428 #define X86_TYPE_P4             12

 430 /*
 431  * x86_vendor allows us to select between
 432  * implementation features and helps guide
 433  * the interpretation of the cpuid instruction.
 434  */
 435 #define X86_VENDOR_Intel        0
 436 #define X86_VENDORSTR_Intel     "GenuineIntel"

 438 #define X86_VENDOR_IntelClone   1

 440 #define X86_VENDOR_AMD          2
 441 #define X86_VENDORSTR_AMD       "AuthenticAMD"

 443 #define X86_VENDOR_Cyrix        3
 444 #define X86_VENDORSTR_CYRIX     "CyrixInstead"

 446 #define X86_VENDOR_UMC          4
 447 #define X86_VENDORSTR_UMC       "UMC UMC UMC "

 449 #define X86_VENDOR_NexGen       5
 450 #define X86_VENDORSTR_NexGen    "NexGenDriven"

 452 #define X86_VENDOR_Centaur      6
 453 #define X86_VENDORSTR_Centaur   "CentaurHauls"

 455 #define X86_VENDOR_Rise         7
 456 #define X86_VENDORSTR_Rise      "RiseRiseRise"
```



```
 458 #define X86_VENDOR_SiS          8
 459 #define X86_VENDORSTR_SiS       "SiS SiS SiS "

 461 #define X86_VENDOR_TM           9
 462 #define X86_VENDORSTR_TM        "GenuineTMx86"

 464 #define X86_VENDOR_NSC          10
 465 #define X86_VENDORSTR_NSC       "Geode by NSC"

 467 /*
 468  * Vendor string max len + \0
 469  */
 470 #define X86_VENDOR_STRLEN       13

 472 /*
 473  * Some vendor/family/model/stepping ranges are commonly grouped under
 474  * a single identifying banner by the vendor.  The following encode
 475  * that "revision" in a uint32_t with the 8 most significant bits
 476  * identifying the vendor with X86_VENDOR_*, the next 8 identifying the
 477  * family, and the remaining 16 typically forming a bitmask of revisions
 478  * within that family with more significant bits indicating "later" revisions.
 479  */

 481 #define _X86_CHIPREV_VENDOR_MASK        0xff000000u
 482 #define _X86_CHIPREV_VENDOR_SHIFT       24
 483 #define _X86_CHIPREV_FAMILY_MASK        0x00ff0000u
 484 #define _X86_CHIPREV_FAMILY_SHIFT       16
 485 #define _X86_CHIPREV_REV_MASK           0x0000ffffu

 487 #define _X86_CHIPREV_VENDOR(x) \
 488         (((x) & _X86_CHIPREV_VENDOR_MASK) >> _X86_CHIPREV_VENDOR_SHIFT)
 489 #define _X86_CHIPREV_FAMILY(x) \
 490         (((x) & _X86_CHIPREV_FAMILY_MASK) >> _X86_CHIPREV_FAMILY_SHIFT)
 491 #define _X86_CHIPREV_REV(x) \
 492         ((x) & _X86_CHIPREV_REV_MASK)

 494 /* True if x matches in vendor and family and if x matches the given rev mask */
 495 #define X86_CHIPREV_MATCH(x, mask) \
 496         (_X86_CHIPREV_VENDOR(x) == _X86_CHIPREV_VENDOR(mask) && \
 497         _X86_CHIPREV_FAMILY(x) == _X86_CHIPREV_FAMILY(mask) && \
 498         ((_X86_CHIPREV_REV(x) & _X86_CHIPREV_REV(mask)) != 0))

 500 /* True if x matches in vendor and family, and rev is at least minx */
 501 #define X86_CHIPREV_ATLEAST(x, minx) \
 502         (_X86_CHIPREV_VENDOR(x) == _X86_CHIPREV_VENDOR(minx) && \
 503         _X86_CHIPREV_FAMILY(x) == _X86_CHIPREV_FAMILY(minx) && \
 504         _X86_CHIPREV_REV(x) >= _X86_CHIPREV_REV(minx))

 506 #define _X86_CHIPREV_MKREV(vendor, family, rev) \
 507         ((uint32_t)(vendor) << _X86_CHIPREV_VENDOR_SHIFT | \
 508         (family) << _X86_CHIPREV_FAMILY_SHIFT | (rev))

 510 /* True if x matches in vendor, and family is at least minx */
 511 #define X86_CHIPFAM_ATLEAST(x, minx) \
 512         (_X86_CHIPREV_VENDOR(x) == _X86_CHIPREV_VENDOR(minx) && \
 513         _X86_CHIPREV_FAMILY(x) >= _X86_CHIPREV_FAMILY(minx))

 515 /* Revision default */
 516 #define X86_CHIPREV_UNKNOWN     0x0

 518 /*
 519  * Definitions for AMD Family 0xf. Minor revisions C0 and CG are
 520  * sufficiently different that we will distinguish them; in all other
 521  * case we will identify the major revision.
 522  */
 523 #define X86_CHIPREV_AMD_F_REV_B _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0001)
```

```
524 #define X86_CHIPREV_AMD_F_REV_C0 _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0002)
525 #define X86_CHIPREV_AMD_F_REV_CG _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0004)
526 #define X86_CHIPREV_AMD_F_REV_D _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0008)
527 #define X86_CHIPREV_AMD_F_REV_E _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0010)
528 #define X86_CHIPREV_AMD_F_REV_F _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0020)
529 #define X86_CHIPREV_AMD_F_REV_G _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0040)

531 /*
532  * Definitions for AMD Family 0x10.  Rev A was Engineering Samples only.
533  */
534 #define X86_CHIPREV_AMD_10_REV_A \
535         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0001)
536 #define X86_CHIPREV_AMD_10_REV_B \
537         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0002)
538 #define X86_CHIPREV_AMD_10_REV_C2 \
539         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0004)
540 #define X86_CHIPREV_AMD_10_REV_C3 \
541         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0008)
542 #define X86_CHIPREV_AMD_10_REV_D0 \
543         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0010)
544 #define X86_CHIPREV_AMD_10_REV_D1 \
545         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0020)
546 #define X86_CHIPREV_AMD_10_REV_E \
547         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0040)

549 /*
550  * Definitions for AMD Family 0x11.
551  */
552 #define X86_CHIPREV_AMD_11_REV_B \
553         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x11, 0x0002)

555 /*
556  * Definitions for AMD Family 0x12.
557  */
558 #define X86_CHIPREV_AMD_12_REV_B \
559         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x12, 0x0002)

561 /*
562  * Definitions for AMD Family 0x14.
563  */
564 #define X86_CHIPREV_AMD_14_REV_B \
565         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x14, 0x0002)
566 #define X86_CHIPREV_AMD_14_REV_C \
567         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x14, 0x0004)

569 /*
570  * Definitions for AMD Family 0x15
571  */
572 #define X86_CHIPREV_AMD_15OR_REV_B2 \
573         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x15, 0x0001)

575 #define X86_CHIPREV_AMD_15TN_REV_A1 \
576         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x15, 0x0002)

578 /*
579  * Various socket/package types, extended as the need to distinguish
580  * a new type arises.  The top 8 byte identfies the vendor and the
581  * remaining 24 bits describe 24 socket types.
582  */

584 #define _X86_SOCKET_VENDOR_SHIFT        24
585 #define _X86_SOCKET_VENDOR(x)   ((x) >> _X86_SOCKET_VENDOR_SHIFT)
586 #define _X86_SOCKET_TYPE_MASK   0x00ffffff
587 #define _X86_SOCKET_TYPE(x)             ((x) & _X86_SOCKET_TYPE_MASK)

589 #define _X86_SOCKET_MKVAL(vendor, bitval) \
```



```
590         ((uint32_t)(vendor) << _X86_SOCKET_VENDOR_SHIFT | (bitval))

592 #define X86_SOCKET_MATCH(s, mask) \
593         (_X86_SOCKET_VENDOR(s) == _X86_SOCKET_VENDOR(mask) && \
594         (_X86_SOCKET_TYPE(s) & _X86_SOCKET_TYPE(mask)) != 0)

596 #define X86_SOCKET_UNKNOWN 0x0
597         /*
598          * AMD socket types
599          */
600 #define X86_SOCKET_754          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000001)
601 #define X86_SOCKET_939          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000002)
602 #define X86_SOCKET_940          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000004)
603 #define X86_SOCKET_S1g1         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000008)
604 #define X86_SOCKET_AM2          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000010)
605 #define X86_SOCKET_F1207        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000020)
606 #define X86_SOCKET_S1g2         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000040)
607 #define X86_SOCKET_S1g3         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000080)
608 #define X86_SOCKET_AM           _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000100)
609 #define X86_SOCKET_AM2R2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000200)
610 #define X86_SOCKET_AM3          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000400)
611 #define X86_SOCKET_G34          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000800)
612 #define X86_SOCKET_ASB2         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x001000)
613 #define X86_SOCKET_C32          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x002000)
614 #define X86_SOCKET_S1g4         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x004000)
615 #define X86_SOCKET_FT1          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x008000)
616 #define X86_SOCKET_FM1          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x010000)
617 #define X86_SOCKET_FS1          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x020000)
618 #define X86_SOCKET_AM3R2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x040000)
619 #define X86_SOCKET_FP2          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x080000)
620 #define X86_SOCKET_FS1R2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x100000)
621 #define X86_SOCKET_FM2          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x200000)

623 /*
624  * xgetbv/xsetbv support
625  */

627 #define XFEATURE_ENABLED_MASK   0x0
628 /*
629  * XFEATURE_ENABLED_MASK values (eax)
630  */
631 #define XFEATURE_LEGACY_FP      0x1
632 #define XFEATURE_SSE            0x2
633 #define XFEATURE_AVX            0x4
634 #define XFEATURE_MAX            XFEATURE_AVX
635 #define XFEATURE_FP_ALL \
636         (XFEATURE_LEGACY_FP|XFEATURE_SSE|XFEATURE_AVX)

638 #if !defined(_ASM)

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

642 #define NUM_X86_FEATURES        40
643 extern uchar_t x86_featureset[];

645 extern void free_x86_featureset(void *featureset);
646 extern boolean_t is_x86_feature(void *featureset, uint_t feature);
647 extern void add_x86_feature(void *featureset, uint_t feature);
648 extern void remove_x86_feature(void *featureset, uint_t feature);
649 extern boolean_t compare_x86_featureset(void *setA, void *setB);
650 extern void print_x86_featureset(void *featureset);


653 extern uint_t x86_type;
654 extern uint_t x86_vendor;
655 extern uint_t x86_clflush_size;
```

```
 657 extern uint_t pentiumpro_bug4046376;
 658 extern uint_t pentiumpro_bug4064495;

 660 extern uint_t enable486;

 659 extern const char CyrixInstead[];

 661 #endif

 663 #if defined(_KERNEL)

 665 /*
 666  * This structure is used to pass arguments and get return values back
 667  * from the CPUID instruction in __cpuid_insn() routine.
 668  */
 669 struct cpuid_regs {
 670         uint32_t        cp_eax;
 671         uint32_t        cp_ebx;
 672         uint32_t        cp_ecx;
 673         uint32_t        cp_edx;
 674 };
```

_____unchanged_portion_omitted_