

```

*****
8739 Mon Jul 28 07:44:09 2014
new/usr/src/uts/intel/asm/atomic.h
5044 define static inlines for most often used atomic functions
*****
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, Version 1.0 only
6  * (the "License"). You may not use this file except in compliance
7  * 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 2005 Sun Microsystems, Inc. All rights reserved.
24 * Use is subject to license terms.
25 */

27 #ifndef _ASM_ATOMIC_H
28 #define _ASM_ATOMIC_H

30 #include <sys/ccompile.h>
31 #include <sys/types.h>

33 #ifdef __cplusplus
34 extern "C" {
35 #endif

37 #if !defined(__lint) && defined(__GNUC__)

39 /*
40  * This file contains a number of static inline functions implementing
41  * various atomic variable functions. Note that these are *not* all of the
42  * atomic * functions as defined in usr/src/uts/common/sys/atomic.h. All
43  * possible atomic_* functions are implemented in usr/src/common/atomic in
44  * pure assembly. In the absence of an identically named function in this
45  * header file, any use of the function will result in the compiler emitting
46  * a function call as usual. On the other hand, if an identically named
47  * function exists in this header as a static inline, the compiler will
48  * inline its contents and the linker never sees the symbol reference. We
49  * use this to avoid implementing some of the more complex and less used
50  * functions and instead falling back to function calls. Note that in some
51  * cases (e.g., atomic_inc_64) we implement a static inline only on AMD64
52  * but not i386.
53  */

55 /*
56  * Instruction suffixes for various operand sizes (assuming AMD64)
57  */
58 #define SUF_8      "b"
59 #define SUF_16    "w"
60 #define SUF_32    "l"
61 #define SUF_64    "q"

```

```

63 #endif /* ! codereview */
64 #if defined(__amd64)
65 #define SUF_LONG      SUF_64
66 #define SUF_PTR       SUF_64
67 #define __ATOMIC_OP64(...) __ATOMIC_OPXX(__VA_ARGS__)
68 #elif defined(__i386)
69 #define SUF_LONG      SUF_32
70 #define SUF_PTR       SUF_32
71 #define __ATOMIC_OP64(...)
72 #else
73 #error "port me"
74 #endif

76 #if defined(__amd64) || defined(__i386)

78 #define __ATOMIC_OPXX(fxn, type, op)
79 extern __GNU_INLINE void
80 fxn(volatile type *target)
81 {
82     __asm__ __volatile__(
83         "lock; " op " %0"
84         : "+m" (*target));
85 }

87 __ATOMIC_OPXX(atomic_inc_8,      uint8_t,  "inc" SUF_8)
88 __ATOMIC_OPXX(atomic_inc_16,     uint16_t, "inc" SUF_16)
89 __ATOMIC_OPXX(atomic_inc_32,     uint32_t, "inc" SUF_32)
90 __ATOMIC_OPXX(atomic_inc_64,     uint64_t, "inc" SUF_64)
91 __ATOMIC_OPXX(atomic_inc_uchar,  uchar_t,  "inc" SUF_8)
92 __ATOMIC_OPXX(atomic_inc_ushort, ushort_t, "inc" SUF_16)
93 __ATOMIC_OPXX(atomic_inc_uint,   uint_t,   "inc" SUF_32)
94 __ATOMIC_OPXX(atomic_inc_ulong,  ulong_t,  "inc" SUF_LONG)

96 __ATOMIC_OPXX(atomic_dec_8,      uint8_t,  "dec" SUF_8)
97 __ATOMIC_OPXX(atomic_dec_16,     uint16_t, "dec" SUF_16)
98 __ATOMIC_OPXX(atomic_dec_32,     uint32_t, "dec" SUF_32)
99 __ATOMIC_OPXX(atomic_dec_64,     uint64_t, "dec" SUF_64)
100 __ATOMIC_OPXX(atomic_dec_uchar,  uchar_t,  "dec" SUF_8)
101 __ATOMIC_OPXX(atomic_dec_ushort, ushort_t, "dec" SUF_16)
102 __ATOMIC_OPXX(atomic_dec_uint,   uint_t,   "dec" SUF_32)
103 __ATOMIC_OPXX(atomic_dec_ulong,  ulong_t,  "dec" SUF_LONG)

105 #undef __ATOMIC_OPXX

107 #define __ATOMIC_OPXX(fxn, type1, type2, op)
108 extern __GNU_INLINE void
109 fxn(volatile type1 *target, type2 delta)
110 {
111     __asm__ __volatile__(
112         "lock; " op " %1,%0"
113         : "+m" (*target)
114         : "ir" (delta));
115 }

117 __ATOMIC_OPXX(atomic_add_8,      uint8_t,  int8_t,   "add" SUF_8)
118 __ATOMIC_OPXX(atomic_add_16,     uint16_t, int16_t,  "add" SUF_16)
119 __ATOMIC_OPXX(atomic_add_32,     uint32_t, int32_t,  "add" SUF_32)
120 __ATOMIC_OPXX(atomic_add_64,     uint64_t, int64_t,  "add" SUF_64)
121 __ATOMIC_OPXX(atomic_add_char,   uchar_t,  signed char, "add" SUF_8)
122 __ATOMIC_OPXX(atomic_add_short,  ushort_t, short,    "add" SUF_16)
123 __ATOMIC_OPXX(atomic_add_int,    uint_t,   int,      "add" SUF_32)
124 __ATOMIC_OPXX(atomic_add_long,   ulong_t,  long,     "add" SUF_LONG)

126 /*
127  * We don't use the above macro here because atomic_add_ptr has an

```

```

128 * inconsistent type. The first argument should really be a 'volatile void
129 * ***'.
130 */
131 extern __GNU_INLINE void
132 atomic_add_ptr(volatile void *target, ssize_t delta)
133 {
134     volatile void **tmp = (volatile void **)target;
135
136     __asm__ __volatile__(
137         "lock; add" SUF_PTR " %1,%0"
138         : "+m" (*tmp)
139         : "ir" (delta));
140 }
141
142 __ATOMIC_OPXX(atomic_or_8,      uint8_t,  uint8_t,  "or" SUF_8)
143 __ATOMIC_OPXX(atomic_or_16,   uint16_t, uint16_t, "or" SUF_16)
144 __ATOMIC_OPXX(atomic_or_32,   uint32_t, uint32_t, "or" SUF_32)
145 __ATOMIC_OPXX(atomic_or_64,   uint64_t, uint64_t, "or" SUF_64)
146 __ATOMIC_OPXX(atomic_or_uchar, uchar_t,  uchar_t,  "or" SUF_8)
147 __ATOMIC_OPXX(atomic_or_ushort, ushort_t, ushort_t, "or" SUF_16)
148 __ATOMIC_OPXX(atomic_or_uint, uint_t,   uint_t,   "or" SUF_32)
149 __ATOMIC_OPXX(atomic_or_ulong, ulong_t,  ulong_t,  "or" SUF_LONG)
150
151 __ATOMIC_OPXX(atomic_and_8,    uint8_t,  uint8_t,  "and" SUF_8)
152 __ATOMIC_OPXX(atomic_and_16,  uint16_t, uint16_t, "and" SUF_16)
153 __ATOMIC_OPXX(atomic_and_32,  uint32_t, uint32_t, "and" SUF_32)
154 __ATOMIC_OPXX(atomic_and_64,  uint64_t, uint64_t, "and" SUF_64)
155 __ATOMIC_OPXX(atomic_and_uchar, uchar_t,  uchar_t,  "and" SUF_8)
156 __ATOMIC_OPXX(atomic_and_ushort, ushort_t, ushort_t, "and" SUF_16)
157 __ATOMIC_OPXX(atomic_and_uint, uint_t,   uint_t,   "and" SUF_32)
158 __ATOMIC_OPXX(atomic_and_ulong, ulong_t,  ulong_t,  "and" SUF_LONG)
159 #endif /* ! codereview */
160
161 #undef __ATOMIC_OPXX
162
163 #define __ATOMIC_OPXX(fxn, type, op, reg) \
164 extern __GNU_INLINE type \
165 fxn(volatile type *target, type cmp, type new) \
166 { \
167     type ret; \
168     __asm__ __volatile__( \
169         "lock; " op " %2,%0" \
170         : "+m" (*target), "=a" (ret) \
171         : reg (new), "1" (cmp) \
172         : "cc"); \
173     return (ret); \
174 }
175
176 __ATOMIC_OPXX(atomic_cas_8,    uint8_t,  "cmpxchg" SUF_8,  "q")
177 __ATOMIC_OPXX(atomic_cas_16,  uint16_t, "cmpxchg" SUF_16, "r")
178 __ATOMIC_OPXX(atomic_cas_32,  uint32_t, "cmpxchg" SUF_32, "r")
179 __ATOMIC_OPXX(atomic_cas_64,  uint64_t, "cmpxchg" SUF_64, "r")
180 __ATOMIC_OPXX(atomic_cas_uchar, uchar_t,  "cmpxchg" SUF_8,  "q")
181 __ATOMIC_OPXX(atomic_cas_ushort, ushort_t, "cmpxchg" SUF_16, "r")
182 __ATOMIC_OPXX(atomic_cas_uint, uint_t,   "cmpxchg" SUF_32, "r")
183 __ATOMIC_OPXX(atomic_cas_ulong, ulong_t,  "cmpxchg" SUF_LONG, "r")
184
185 #undef __ATOMIC_OPXX
186
187 /*
188 * We don't use the above macro here because atomic_cas_ptr has an
189 * inconsistent type. The first argument should really be a 'volatile void
190 * ***'.
191 */
192 extern __GNU_INLINE void *
193 atomic_cas_ptr(volatile void *target, void *cmp, void *new)

```

```

194 {
195     volatile void **tmp = (volatile void **)target;
196     void *ret;
197
198     __asm__ __volatile__(
199         "lock; cmpxchg" SUF_PTR " %2,%0"
200         : "+m" (*tmp), "=a" (ret)
201         : "r" (new), "1" (cmp)
202         : "cc");
203
204     return (ret);
205 }
206
207 #define __ATOMIC_OPXX(fxn, type, op, reg) \
208 extern __GNU_INLINE type \
209 fxn(volatile type *target, type val) \
210 { \
211     __asm__ __volatile__( \
212         op " %1,%0" \
213         : "+m" (*target), "+" reg (val)); \
214     return (val); \
215 }
216
217 __ATOMIC_OPXX(atomic_swap_8,    uint8_t,  "xchg" SUF_8,  "q")
218 __ATOMIC_OPXX(atomic_swap_16,  uint16_t, "xchg" SUF_16, "r")
219 __ATOMIC_OPXX(atomic_swap_32,  uint32_t, "xchg" SUF_32, "r")
220 __ATOMIC_OPXX(atomic_swap_64,  uint64_t, "xchg" SUF_64, "r")
221 __ATOMIC_OPXX(atomic_swap_uchar, uchar_t,  "xchg" SUF_8,  "q")
222 __ATOMIC_OPXX(atomic_swap_ushort, ushort_t, "xchg" SUF_16, "r")
223 __ATOMIC_OPXX(atomic_swap_uint, uint_t,   "xchg" SUF_32, "r")
224 __ATOMIC_OPXX(atomic_swap_ulong, ulong_t,  "xchg" SUF_LONG, "r")
225
226 #undef __ATOMIC_OPXX
227
228 /*
229 * We don't use the above macro here because atomic_swap_ptr has an
230 * inconsistent type. The first argument should really be a 'volatile void
231 * ***'.
232 */
233 extern __GNU_INLINE void *
234 atomic_swap_ptr(volatile void *target, void *val)
235 {
236     volatile void **tmp = (volatile void **)target;
237
238     __asm__ __volatile__(
239         "xchg" SUF_PTR " %1,%0"
240         : "+m" (*tmp), "+r" (val));
241
242     return (val);
243 }
244 #elif defined(__i386)
245 #else
246 #error "port me"
247 #endif
248
249 #undef SUF_8
250 #undef SUF_16
251 #undef SUF_32
252 #undef SUF_64
253 #undef SUF_LONG
254 #undef SUF_PTR
255
256 #undef __ATOMIC_OP64
257
258 #endif /* ! codereview */

```

```
259 #endif /* !__lint && __GNUC__ */
```

```
261 #ifdef __cplusplus
```

```
262 }
```

```
263 #endif
```

```
265 #endif /* _ASM_ATOMICS_H */
```