```
**********************************************************
    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
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (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 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _ASM_ATOMIC_H
#define _ASM_ATOMIC_H

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

#ifdef  __cplusplus
extern "C" {
#endif

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

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

/*
 * Instruction suffixes for various operand sizes (assuming AMD64)
 */
#define SUF_8           "b"
#define SUF_16          "w"
#define SUF_32          "l"
#define SUF_64          "q"
```



```
#endif /* ! codereview */
#if defined(__amd64)
#define SUF_LONG        SUF_64
#define SUF_PTR         SUF_64
#define __ATOMIC_OP64(...)      __ATOMIC_OPXX(__VA_ARGS__)
#elif defined(__i386)
#define SUF_LONG        SUF_32
#define SUF_PTR         SUF_32
#define __ATOMIC_OP64(...)
#else
#error "port me"
#endif

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

#define __ATOMIC_OPXX(fxn, type, op)                                    \
extern __GNU_INLINE void                                                \
fxn(volatile type *target)                                              \
{                                                                       \
        __asm__ __volatile__(                                           \
            "lock; " op " %0"                                           \
            : "+m" (*target));                                          \
}

__ATOMIC_OPXX(atomic_inc_8,      uint8_t,  "inc" SUF_8)
__ATOMIC_OPXX(atomic_inc_16,     uint16_t, "inc" SUF_16)
__ATOMIC_OPXX(atomic_inc_32,     uint32_t, "inc" SUF_32)
__ATOMIC_OP64(atomic_inc_64,     uint64_t, "inc" SUF_64)
__ATOMIC_OPXX(atomic_inc_uchar,  uchar_t,  "inc" SUF_8)
__ATOMIC_OPXX(atomic_inc_ushort, ushort_t, "inc" SUF_16)
__ATOMIC_OPXX(atomic_inc_uint,   uint_t,   "inc" SUF_32)
__ATOMIC_OPXX(atomic_inc_ulong,  ulong_t,  "inc" SUF_LONG)

__ATOMIC_OPXX(atomic_dec_8,      uint8_t,  "dec" SUF_8)
__ATOMIC_OPXX(atomic_dec_16,     uint16_t, "dec" SUF_16)
__ATOMIC_OPXX(atomic_dec_32,     uint32_t, "dec" SUF_32)
__ATOMIC_OP64(atomic_dec_64,     uint64_t, "dec" SUF_64)
__ATOMIC_OPXX(atomic_dec_uchar,  uchar_t,  "dec" SUF_8)
__ATOMIC_OPXX(atomic_dec_ushort, ushort_t, "dec" SUF_16)
__ATOMIC_OPXX(atomic_dec_uint,   uint_t,   "dec" SUF_32)
__ATOMIC_OPXX(atomic_dec_ulong,  ulong_t,  "dec" SUF_LONG)

#undef __ATOMIC_OPXX

#define __ATOMIC_OPXX(fxn, type1, type2, op)                            \
extern __GNU_INLINE void                                                \
fxn(volatile type1 *target, type2 delta)                                \
{                                                                       \
        __asm__ __volatile__(                                           \
            "lock; " op " %1,%0"                                        \
            : "+m" (*target)                                            \
            : "ir" (delta));                                            \
}

__ATOMIC_OPXX(atomic_add_8,      uint8_t,  int8_t,      "add" SUF_8)
__ATOMIC_OPXX(atomic_add_16,     uint16_t, int16_t,     "add" SUF_16)
__ATOMIC_OPXX(atomic_add_32,     uint32_t, int32_t,     "add" SUF_32)
__ATOMIC_OP64(atomic_add_64,     uint64_t, int64_t,     "add" SUF_64)
__ATOMIC_OPXX(atomic_add_char,   uchar_t,  signed char, "add" SUF_8)
__ATOMIC_OPXX(atomic_add_short,  ushort_t, short,       "add" SUF_16)
__ATOMIC_OPXX(atomic_add_int,    uint_t,   int,         "add" SUF_32)
__ATOMIC_OPXX(atomic_add_long,   ulong_t,  long,        "add" SUF_LONG)

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

```
 * inconsistent type.  The first argument should really be a 'volatile void
 * **'.
 */
extern __GNU_INLINE void
atomic_add_ptr(volatile void *target, ssize_t delta)
{
        volatile void **tmp = (volatile void **)target;

        __asm__ __volatile__(
            "lock; add" SUF_PTR " %1,%0"
            : "+m" (*tmp)
            : "ir" (delta));
}

__ATOMIC_OPXX(atomic_or_8,       uint8_t,  uint8_t,  "or" SUF_8)
__ATOMIC_OPXX(atomic_or_16,      uint16_t, uint16_t, "or" SUF_16)
__ATOMIC_OPXX(atomic_or_32,      uint32_t, uint32_t, "or" SUF_32)
__ATOMIC_OP64(atomic_or_64,      uint64_t, uint64_t, "or" SUF_64)
__ATOMIC_OPXX(atomic_or_uchar,   uchar_t,  uchar_t,  "or" SUF_8)
__ATOMIC_OPXX(atomic_or_ushort,  ushort_t, ushort_t, "or" SUF_16)
__ATOMIC_OPXX(atomic_or_uint,    uint_t,   uint_t,   "or" SUF_32)
__ATOMIC_OPXX(atomic_or_ulong,   ulong_t,  ulong_t,  "or" SUF_LONG)

__ATOMIC_OPXX(atomic_and_8,      uint8_t,  uint8_t,  "and" SUF_8)
__ATOMIC_OPXX(atomic_and_16,     uint16_t, uint16_t, "and" SUF_16)
__ATOMIC_OPXX(atomic_and_32,     uint32_t, uint32_t, "and" SUF_32)
__ATOMIC_OP64(atomic_and_64,     uint64_t, uint64_t, "and" SUF_64)
__ATOMIC_OPXX(atomic_and_uchar,  uchar_t,  uchar_t,  "and" SUF_8)
__ATOMIC_OPXX(atomic_and_ushort, ushort_t, ushort_t, "and" SUF_16)
__ATOMIC_OPXX(atomic_and_uint,   uint_t,   uint_t,   "and" SUF_32)
__ATOMIC_OPXX(atomic_and_ulong,  ulong_t,  ulong_t,  "and" SUF_LONG)
#endif /* ! codereview */

#undef __ATOMIC_OPXX

#define __ATOMIC_OPXX(fxn, type, op, reg)                               \
extern __GNU_INLINE type                                                \
fxn(volatile type *target, type cmp, type new)                          \
{                                                                       \
        type ret;                                                       \
        __asm__ __volatile__(                                           \
            "lock; " op " %2,%0"                                        \
            : "+m" (*target), "=a" (ret)                                \
            : reg (new), "1" (cmp)                                      \
            : "cc");                                                    \
        return (ret);                                                   \
}

__ATOMIC_OPXX(atomic_cas_8,      uint8_t,  "cmpxchg" SUF_8,    "q")
__ATOMIC_OPXX(atomic_cas_16,     uint16_t, "cmpxchg" SUF_16,   "r")
__ATOMIC_OPXX(atomic_cas_32,     uint32_t, "cmpxchg" SUF_32,   "r")
__ATOMIC_OP64(atomic_cas_64,     uint64_t, "cmpxchg" SUF_64,   "r")
__ATOMIC_OPXX(atomic_cas_uchar,  uchar_t,  "cmpxchg" SUF_8,    "q")
__ATOMIC_OPXX(atomic_cas_ushort, ushort_t, "cmpxchg" SUF_16,   "r")
__ATOMIC_OPXX(atomic_cas_uint,   uint_t,   "cmpxchg" SUF_32,   "r")
__ATOMIC_OPXX(atomic_cas_ulong,  ulong_t,  "cmpxchg" SUF_LONG, "r")

#undef __ATOMIC_OPXX

/*
 * We don't use the above macro here because atomic_cas_ptr has an
 * inconsistent type.  The first argument should really be a 'volatile void
 * **'.
 */
extern __GNU_INLINE void *
atomic_cas_ptr(volatile void *target, void *cmp, void *new)
```



```
{
        volatile void **tmp = (volatile void **)target;
        void *ret;

        __asm__ __volatile__(
            "lock; cmpxchg" SUF_PTR " %2,%0"
            : "+m" (*tmp), "=a" (ret)
            : "r" (new), "1" (cmp)
            : "cc");

        return (ret);
}

#define __ATOMIC_OPXX(fxn, type, op, reg)                               \
extern __GNU_INLINE type                                                \
fxn(volatile type *target, type val)                                    \
{                                                                       \
        __asm__ __volatile__(                                           \
            op " %1,%0"                                                 \
            : "+m" (*target), "+" reg (val));                           \
        return (val);                                                   \
}

__ATOMIC_OPXX(atomic_swap_8,      uint8_t,  "xchg" SUF_8,    "q")
__ATOMIC_OPXX(atomic_swap_16,     uint16_t, "xchg" SUF_16,   "r")
__ATOMIC_OPXX(atomic_swap_32,     uint32_t, "xchg" SUF_32,   "r")
__ATOMIC_OP64(atomic_swap_64,     uint64_t, "xchg" SUF_64,   "r")
__ATOMIC_OPXX(atomic_swap_uchar,  uchar_t,  "xchg" SUF_8,    "q")
__ATOMIC_OPXX(atomic_swap_ushort, ushort_t, "xchg" SUF_16,   "r")
__ATOMIC_OPXX(atomic_swap_uint,   uint_t,   "xchg" SUF_32,   "r")
__ATOMIC_OPXX(atomic_swap_ulong,  ulong_t,  "xchg" SUF_LONG, "r")

#undef __ATOMIC_OPXX

/*
 * We don't use the above macro here because atomic_swap_ptr has an
 * inconsistent type.  The first argument should really be a 'volatile void
 * **'.
 */
extern __GNU_INLINE void *
atomic_swap_ptr(volatile void *target, void *val)
{
        volatile void **tmp = (volatile void **)target;

        __asm__ __volatile__(
            "xchg" SUF_PTR " %1,%0"
            : "+m" (*tmp), "+r" (val));

        return (val);
}
#elif defined(__i386)

#else
#error  "port me"
#endif

#undef SUF_8
#undef SUF_16
#undef SUF_32
#undef SUF_64
#undef SUF_LONG
#undef SUF_PTR

#undef __ATOMIC_OP64

#endif /* ! codereview */
```

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

261 #ifdef __cplusplus
262 }
263 #endif

265 #endif  /* _ASM_ATOMIC_H */
```