```
**********************************************************
    9158 Sun Sep 27 09:37:18 2015
new/usr/src/uts/intel/asm/atomic.h
6263 add missing cc clobbers to intel atomic inlines
**********************************************************
```

```
/*
 * 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.
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
 * Copyright 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
#endif /* ! codereview */
 */

#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__)

/* BEGIN CSTYLED */
/*
 * 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"

#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)                                            \
            : /* no inputs */                                           \
            : "cc");                                                    \
            : "+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, reg)                       \
extern __GNU_INLINE void                                                \
fxn(volatile type1 *target, type2 delta)                                \
{                                                                       \
        __asm__ __volatile__(                                           \
            "lock; " op " %1,%0"                                        \
            : "+m" (*target)                                            \
            : "i" reg (delta)                                           \
            : "cc");                                                    \
            : "i" reg (delta));                                         \
}

__ATOMIC_OPXX(atomic_add_8,     uint8_t,  int8_t,      "add" SUF_8,    "q")
__ATOMIC_OPXX(atomic_add_16,    uint16_t, int16_t,     "add" SUF_16,   "r")
__ATOMIC_OPXX(atomic_add_32,    uint32_t, int32_t,     "add" SUF_32,   "r")
```

```
 126 __ATOMIC_OP64(atomic_add_64,    uint64_t, int64_t,     "add" SUF_64,   "r")
 127 __ATOMIC_OPXX(atomic_add_char,  uchar_t,  signed char, "add" SUF_8,    "q")
 128 __ATOMIC_OPXX(atomic_add_short, ushort_t, short,       "add" SUF_16,   "r")
 129 __ATOMIC_OPXX(atomic_add_int,   uint_t,   int,         "add" SUF_32,   "r")
 130 __ATOMIC_OPXX(atomic_add_long,  ulong_t,  long,        "add" SUF_LONG, "r")

 132 /*
 133  * We don't use the above macro here because atomic_add_ptr has an
 134  * inconsistent type.  The first argument should really be a 'volatile void
 135  * **'.
 136  */
 137 extern __GNU_INLINE void
 138 atomic_add_ptr(volatile void *target, ssize_t delta)
 139 {
 140         volatile void **tmp = (volatile void **)target;

 142         __asm__ __volatile__(
 143             "lock; add" SUF_PTR " %1,%0"
 144             : "+m" (*tmp)
 145             : "ir" (delta)
 146             : "cc");
  81             : "ir" (delta));
 147 }
```

_____***unchanged_portion_omitted_***