

## [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-09/8952.html>

---

**From:** Hirokazu Takata (*takata\_at\_linux-m32r.org*)

**Date:** 09/30/04

Date: Thu, 30 Sep 2004 10:14:05 +0900 (JST)

To: Andrew Morton <akpm@osdl.org>

Hi,

I made a patch to upgrade some header files for m32r.

- Change to use temporary register variables allocated by the compiler, instead of fixed register variables.
- Change `__inline__` to `inline`.

Please apply.

Thank you.

Signed-off-by: Hirokazu Takata <takata@linux-m32r.org>

```
---
include/asm-m32r/bitops.h      | 156 ++++++-----
include/asm-m32r/semaphore.h   | 118 +++++-----
include/asm-m32r/spinlock.h    | 199 ++++++-----
3 files changed, 218 insertions(+), 255 deletions(-)
```

```
diff -ruNp a/include/asm-m32r/bitops.h b/include/asm-m32r/bitops.h
--- a/include/asm-m32r/bitops.h 2004-09-28 10:19:53.000000000 +0900
+++ b/include/asm-m32r/bitops.h 2004-09-28 12:38:24.000000000 +0900
@@ -1,17 +1,14 @@
 #ifndef _ASM_M32R_BITOPS_H
 #define _ASM_M32R_BITOPS_H

-/* $Id$ */
-
 /*
 * linux/include/asm-m32r/bitops.h
 *   orig : i386 2.4.10
 *
 * Copyright 1992, Linus Torvalds.
 *
 * M32R version:
 *   Copyright (C) 2001, 2002 Hitoshi Yamamoto
 *   Copyright (C) 2004 Hirokazu Takata
+ *   Copyright (C) 2004 Hirokazu Takata <takata at linux-m32r.org>
 */

#include <linux/config.h>
```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

@@ -50,24 +47,25 @@
 * Note that @nr may be almost arbitrarily large; this function is not
 * restricted to acting on a single-word quantity.
 */
-static __inline__ void set_bit(int nr, volatile void * addr)
+static inline void set_bit(int nr, volatile void * addr)
{
    __u32 mask;
    volatile __u32 *a = addr;
-   unsigned long flags;
+   unsigned long flags;
+   unsigned long tmp;

    a += (nr >> 5);
    mask = (1 << (nr & 0x1F));

    local_irq_save(flags);
    __asm__ __volatile__ (
-       DCACHE_CLEAR("r4", "r6", "%0")
-       LOAD"    r4, @%0;                \n\t"
-       "or     r4, %1;                \n\t"
-       STORE"   r4, @%0;                \n\t"
-       : /* no outputs */
+       DCACHE_CLEAR("%0", "r6", "%1")
+       LOAD"    %0, @%1;                \n\t"
+       "or     %0, %2;                \n\t"
+       STORE"   %0, @%1;                \n\t"
+       : "=&r" (tmp)
+       : "r" (a), "r" (mask)
-       : "memory", "r4"
+       : "memory"
#ifdef CONFIG_CHIP_M32700_TS1
        , "r6"
#endif /* CONFIG_CHIP_M32700_TS1 */
@@ -84,7 +82,7 @@ static __inline__ void set_bit(int nr, v
 * If it's called on the same region of memory simultaneously, the effect
 * may be that only one operation succeeds.
 */
-static __inline__ void __set_bit(int nr, volatile void * addr)
+static inline void __set_bit(int nr, volatile void * addr)
{
    __u32 mask;
    volatile __u32 *a = addr;
@@ -104,11 +102,12 @@ static __inline__ void __set_bit(int nr,
 * you should call smp_mb__before_clear_bit() and/or smp_mb__after_clear_bit()
 * in order to ensure changes are visible on other processors.
 */
-static __inline__ void clear_bit(int nr, volatile void * addr)
+static inline void clear_bit(int nr, volatile void * addr)
{
    __u32 mask;
    volatile __u32 *a = addr;
-   unsigned long flags;
+   unsigned long flags;
+   unsigned long tmp;

    a += (nr >> 5);
    mask = (1 << (nr & 0x1F));
@@ -116,13 +115,13 @@ static __inline__ void clear_bit(int nr,
    local_irq_save(flags);

    __asm__ __volatile__ (

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

-         DCACHE_CLEAR("r4", "r6", "%0")
-         LOAD"    r4, @%0;                \n\t"
-         "and    r4, %1;                  \n\t"
-         STORE"  r4, @%0;                \n\t"
-         : /* no outputs */
+         DCACHE_CLEAR("%0", "r6", "%1")
+         LOAD"    %0, @%1;                \n\t"
+         "and    %0, %2;                  \n\t"
+         STORE"  %0, @%1;                \n\t"
+         : "=&r" (tmp)
+         : "r" (a), "r" (~mask)
-         : "memory", "r4"
+         : "memory"
#ifdef CONFIG_CHIP_M32700_TS1
    , "r6"
#endif /* CONFIG_CHIP_M32700_TS1 */
@@ -130,7 +129,7 @@ static __inline__ void clear_bit(int nr,
    local_irq_restore(flags);
}

-static __inline__ void __clear_bit(int nr, volatile unsigned long * addr)
+static inline void __clear_bit(int nr, volatile unsigned long * addr)
{
    unsigned long mask;
    volatile unsigned long *a = addr;
@@ -152,7 +151,7 @@ static __inline__ void __clear_bit(int n
    * If it's called on the same region of memory simultaneously, the effect
    * may be that only one operation succeeds.
    */
-static __inline__ void __change_bit(int nr, volatile void * addr)
+static inline void __change_bit(int nr, volatile void * addr)
{
    __u32 mask;
    volatile __u32 *a = addr;
@@ -171,24 +170,25 @@ static __inline__ void __change_bit(int
    * Note that @nr may be almost arbitrarily large; this function is not
    * restricted to acting on a single-word quantity.
    */
-static __inline__ void change_bit(int nr, volatile void * addr)
+static inline void change_bit(int nr, volatile void * addr)
{
    __u32 mask;
    volatile __u32 *a = addr;
-    unsigned long flags;
+    unsigned long flags;
+    unsigned long tmp;

    a += (nr >> 5);
    mask = (1 << (nr & 0x1F));

    local_irq_save(flags);
    __asm__ __volatile__ (
-         DCACHE_CLEAR("r4", "r6", "%0")
-         LOAD"    r4, @%0;                \n\t"
-         "xor    r4, %1;                  \n\t"
-         STORE"  r4, @%0;                \n\t"
-         : /* no outputs */
+         DCACHE_CLEAR("%0", "r6", "%1")
+         LOAD"    %0, @%1;                \n\t"
+         "xor    %0, %2;                  \n\t"
+         STORE"  %0, @%1;                \n\t"
+         : "=&r" (tmp)

```

Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

        : "r" (a), "r" (mask)
-       : "memory", "r4"
+       : "memory"
#ifdef CONFIG_CHIP_M32700_TS1
    , "r6"
#endif /* CONFIG_CHIP_M32700_TS1 */
@@ -204,28 +204,30 @@ static __inline__ void change_bit(int nr
    * This operation is atomic and cannot be reordered.
    * It also implies a memory barrier.
    */
-static __inline__ int test_and_set_bit(int nr, volatile void * addr)
+static inline int test_and_set_bit(int nr, volatile void * addr)
{
    __u32 mask, oldbit;
    volatile __u32 *a = addr;
-   unsigned long flags;
+   unsigned long flags;
+   unsigned long tmp;

    a += (nr >> 5);
    mask = (1 << (nr & 0x1F));

    local_irq_save(flags);
    __asm__ __volatile__ (
-   DCACHE_CLEAR("%0", "r4", "%1")
-   LOAD"    %0, @%1;                \n\t"
-   "mv     r4, %0;                  \n\t"
-   "and    %0, %2;                  \n\t"
-   "or     r4, %2;                  \n\t"
-   STORE"  r4, @%1;                \n\t"
-   : "=&r" (oldbit)
+   DCACHE_CLEAR("%0", "%1", "%2")
+   LOAD"    %0, @%2;                \n\t"
+   "mv     %1, %0;                  \n\t"
+   "and    %0, %3;                  \n\t"
+   "or     %1, %3;                  \n\t"
+   STORE"  %1, @%2;                \n\t"
+   : "=&r" (oldbit), "=&r" (tmp)
    : "r" (a), "r" (mask)
-   : "memory", "r4"
+   : "memory"
    );
    local_irq_restore(flags);

+   return (oldbit != 0);
}

@@ -238,7 +240,7 @@ static __inline__ int test_and_set_bit(i
    * If two examples of this operation race, one can appear to succeed
    * but actually fail.  You must protect multiple accesses with a lock.
    */
-static __inline__ int __test_and_set_bit(int nr, volatile void * addr)
+static inline int __test_and_set_bit(int nr, volatile void * addr)
{
    __u32 mask, oldbit;
    volatile __u32 *a = addr;
@@ -259,11 +261,12 @@ static __inline__ int __test_and_set_bit
    * This operation is atomic and cannot be reordered.
    * It also implies a memory barrier.
    */
-static __inline__ int test_and_clear_bit(int nr, volatile void * addr)
+static inline int test_and_clear_bit(int nr, volatile void * addr)

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

{
    __u32 mask, oldbit;
    volatile __u32 *a = addr;
-   unsigned long flags;
+   unsigned long flags;
+   unsigned long tmp;

    a += (nr >> 5);
    mask = (1 << (nr & 0x1F));
@@ -271,16 +274,16 @@ static __inline__ int test_and_clear_bit
    local_irq_save(flags);

    __asm__ __volatile__ (
-       DCACHE_CLEAR("%0", "r4", "%2")
-       LOAD"    %0, @%2;                \n\t"
-       "mv      r4, %0; \n\t"
-       "and     %0, %1; \n\t"
-       "not     %1, %1; \n\t"
-       "and     r4, %1; \n\t"
-       STORE"   r4, @%2;                \n\t"
-       : "=&r" (oldbit), "+r" (mask)
+       DCACHE_CLEAR("%0", "%1", "%3")
+       LOAD"    %0, @%3;                \n\t"
+       "mv      %1, %0; \n\t"
+       "and     %0, %2; \n\t"
+       "not     %2, %2; \n\t"
+       "and     %1, %2; \n\t"
+       STORE"   %1, @%3;                \n\t"
+       : "=&r" (oldbit), "&r" (tmp), "+r" (mask)
+       : "r" (a)
-       : "memory", "r4"
+       : "memory"
    );
    local_irq_restore(flags);

@@ -296,7 +299,7 @@ static __inline__ int test_and_clear_bit
    * If two examples of this operation race, one can appear to succeed
    * but actually fail.  You must protect multiple accesses with a lock.
    */
-static __inline__ int __test_and_clear_bit(int nr, volatile void * addr)
+static inline int __test_and_clear_bit(int nr, volatile void * addr)
    {
        __u32 mask, oldbit;
        volatile __u32 *a = addr;
@@ -310,7 +313,7 @@ static __inline__ int __test_and_clear_b
    }

    /* WARNING: non atomic and it can be reordered! */
-static __inline__ int __test_and_change_bit(int nr, volatile void * addr)
+static inline int __test_and_change_bit(int nr, volatile void * addr)
    {
        __u32 mask, oldbit;
        volatile __u32 *a = addr;
@@ -331,28 +334,30 @@ static __inline__ int __test_and_change_
    * This operation is atomic and cannot be reordered.
    * It also implies a memory barrier.
    */
-static __inline__ int test_and_change_bit(int nr, volatile void * addr)
+static inline int test_and_change_bit(int nr, volatile void * addr)
    {
        __u32 mask, oldbit;
        volatile __u32 *a = addr;

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

-     unsigned long flags;
+     unsigned long flags;
+     unsigned long tmp;

    a += (nr >> 5);
    mask = (1 << (nr & 0x1F));

    local_irq_save(flags);
    __asm__ __volatile__ (
-     DCACHE_CLEAR("%0", "r4", "%1")
-     LOAD"    %0, @%1;                \n\t"
-     "mv     r4, %0;                  \n\t"
-     "and    %0, %2;                  \n\t"
-     "xor    r4, %2;                  \n\t"
-     STORE"  r4, @%1;                \n\t"
-     : "=&r" (oldbit)
+     DCACHE_CLEAR("%0", "%1", "%2")
+     LOAD"    %0, @%2;                \n\t"
+     "mv     %1, %0;                  \n\t"
+     "and    %0, %3;                  \n\t"
+     "xor    %1, %3;                  \n\t"
+     STORE"  %1, @%2;                \n\t"
+     : "=&r" (oldbit), "=&r" (tmp)
+     : "r" (a), "r" (mask)
-     : "memory", "r4"
+     : "memory"
    );
    local_irq_restore(flags);
+
    return (oldbit != 0);
}

@@ -365,7 +370,7 @@ static __inline__ int test_and_change_bi
static int test_bit(int nr, const volatile void * addr);
#endif

-static __inline__ int test_bit(int nr, const volatile void * addr)
+static inline int test_bit(int nr, const volatile void * addr)
{
    __u32 mask;
    const volatile __u32 *a = addr;
@@ -382,7 +387,7 @@ static __inline__ int test_bit(int nr, c
*
* Undefined if no zero exists, so code should check against ~0UL first.
*/
-static __inline__ unsigned long ffz(unsigned long word)
+static inline unsigned long ffz(unsigned long word)
{
    int k;

@@ -415,7 +420,7 @@ static __inline__ unsigned long ffz(unsig
* @offset: The bitnumber to start searching at
* @size: The maximum size to search
*/
-static __inline__ int find_next_zero_bit(void *addr, int size, int offset)
+static inline int find_next_zero_bit(void *addr, int size, int offset)
{
    unsigned long *p = ((unsigned long *) addr) + (offset >> 5);
    unsigned long result = offset & ~31UL;
@@ -457,7 +462,7 @@ found_middle:
*
* Undefined if no bit exists, so code should check against 0 first.

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

*/
-static __inline__ unsigned long __ffs(unsigned long word)
+static inline unsigned long __ffs(unsigned long word)
{
    int k = 0;

@@ -483,7 +488,7 @@ static __inline__ unsigned long __ffs(un
    * unlikely to be set. It's guaranteed that at least one of the 140
    * bits is cleared.
*/
-static __inline__ int sched_find_first_bit(unsigned long *b)
+static inline int sched_find_first_bit(unsigned long *b)
{
    if (unlikely(b[0]))
        return __ffs(b[0]);
@@ -502,7 +507,7 @@ static __inline__ int sched_find_first_b
    * @offset: The bitnumber to start searching at
    * @size: The maximum size to search
*/
-static __inline__ unsigned long find_next_bit(const unsigned long *addr,
+static inline unsigned long find_next_bit(const unsigned long *addr,
    unsigned long size, unsigned long offset)
{
    unsigned int *p = ((unsigned int *) addr) + (offset >> 5);
@@ -589,7 +594,7 @@ found_middle:
#define ext2_find_first_zero_bit      find_first_zero_bit
#define ext2_find_next_zero_bit      find_next_zero_bit
#else
-static __inline__ int ext2_set_bit(int nr, volatile void * addr)
+static inline int ext2_set_bit(int nr, volatile void * addr)
{
    __u8 mask, oldbit;
    volatile __u8 *a = addr;
@@ -602,7 +607,7 @@ static __inline__ int ext2_set_bit(int n
    return (oldbit != 0);
}

-static __inline__ int ext2_clear_bit(int nr, volatile void * addr)
+static inline int ext2_clear_bit(int nr, volatile void * addr)
{
    __u8 mask, oldbit;
    volatile __u8 *a = addr;
@@ -615,7 +620,7 @@ static __inline__ int ext2_clear_bit(int
    return (oldbit != 0);
}

-static __inline__ int ext2_test_bit(int nr, const volatile void * addr)
+static inline int ext2_test_bit(int nr, const volatile void * addr)
{
    __u32 mask;
    const volatile __u8 *a = addr;
@@ -629,7 +634,7 @@ static __inline__ int ext2_test_bit(int
#define ext2_find_first_zero_bit(addr, size) \
    ext2_find_next_zero_bit((addr), (size), 0)

-static __inline__ unsigned long ext2_find_next_zero_bit(void *addr,
+static inline unsigned long ext2_find_next_zero_bit(void *addr,
    unsigned long size, unsigned long offset)
{
    unsigned long *p = ((unsigned long *) addr) + (offset >> 5);
@@ -709,4 +714,3 @@ found_middle:
#endif /* __KERNEL__ */

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```
#endif /* _ASM_M32R_BITOPS_H */
-
diff -ruNp a/include/asm-m32r/semaphore.h b/include/asm-m32r/semaphore.h
--- a/include/asm-m32r/semaphore.h      2004-09-28 10:19:53.000000000 +0900
+++ b/include/asm-m32r/semaphore.h      2004-09-28 12:38:24.000000000 +0900
@@ -1,8 +1,6 @@
 #ifndef _ASM_M32R_SEMAPHORE_H
 #define _ASM_M32R_SEMAPHORE_H

-/* $Id$ */
-
 #include <linux/linkage.h>

 #ifdef __KERNEL__
@@ -10,39 +8,15 @@
 /*
  * SMP- and interrupt-safe semaphores..
  *
  * (C) Copyright 1996 Linus Torvalds
  *
  * Modified 1996-12-23 by Dave Grothe <dave@gcom.com> to fix bugs in
  * the original code and to make semaphore waits
  * interruptible so that processes waiting on
  * semaphores can be killed.
  * Modified 1999-02-14 by Andrea Arcangeli, split the sched.c helper
  * functions in asm/semaphore-helper.h while fixing a
  * potential and subtle race discovered by Ulrich Schmid
  * in down_interruptible(). Since I started to play here I
  * also implemented the 'trylock' semaphore operation.
  * 1999-07-02 Artur Skawina <skawina@geocities.com>
  * Optimized "0(ecx)" -> "(ecx)" (the assembler does not
  * do this). Changed calling sequences from push/jmp to
  * traditional call/ret.
  * Modified 2001-01-01 Andreas Franck <afranck@gmx.de>
  * Some hacks to ensure compatibility with recent
  * GCC snapshots, to avoid stack corruption when compiling
  * with -fomit-frame-pointer. It's not sure if this will
  * be fixed in GCC, as our previous implementation was a
  * bit dubious.
  *
  * If you would like to see an analysis of this implementation, please
  * ftp to gcom.com and download the file
  * /pub/linux/src/semaphore/semaphore-2.0.24.tar.gz.
  *
  * Copyright (C) 1996 Linus Torvalds
  * Copyright (C) 2004 Hirokazu Takata <takata at linux-m32r.org>
  */

 #include <linux/config.h>
-#include <asm/system.h>
-#include <asm/atomic.h>
 #include <linux/wait.h>
 #include <linux/rwsem.h>
+#include <asm/system.h>
+#include <asm/atomic.h>

 #undef LOAD
 #undef STORE
@@ -58,21 +32,14 @@ struct semaphore {
     atomic_t count;
     int sleepers;

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

        wait_queue_head_t wait;
-#ifdef WAITQUEUE_DEBUG
-    long __magic;
-#endif
    };

-#ifdef WAITQUEUE_DEBUG
-# define __SEM_DEBUG_INIT(name) \
-    , (int)&(name).__magic
-#else
-# define __SEM_DEBUG_INIT(name)
-#endif
-
-#define __SEMAPHORE_INITIALIZER(name,count) \
- { ATOMIC_INIT(count), 0, __WAIT_QUEUE_HEAD_INITIALIZER((name).wait) \
-   __SEM_DEBUG_INIT(name) }
+#define __SEMAPHORE_INITIALIZER(name, n) \
+{ \
+    .count          = ATOMIC_INIT(n), \
+    .sleepers       = 0, \
+    .wait           = __WAIT_QUEUE_HEAD_INITIALIZER((name).wait) \
+}

#define __MUTEX_INITIALIZER(name) \
    __SEMAPHORE_INITIALIZER(name,1)
@@ -83,7 +50,7 @@ struct semaphore {
#define DECLARE_MUTEX(name) __DECLARE_SEMAPHORE_GENERIC(name,1)
#define DECLARE_MUTEX_LOCKED(name) __DECLARE_SEMAPHORE_GENERIC(name,0)

-static __inline__ void sema_init (struct semaphore *sem, int val)
+static inline void sema_init (struct semaphore *sem, int val)
{
    /*
     * sem = (struct semaphore)__SEMAPHORE_INITIALIZER(*sem,val);
@@ -94,17 +61,14 @@ static __inline__ void sema_init (struct
    atomic_set(&sem->count, val);
    sem->sleepers = 0;
    init_waitqueue_head(&sem->wait);
-#ifdef WAITQUEUE_DEBUG
-    sem->__magic = (int)&sem->__magic;
-#endif
}

-static __inline__ void init_MUTEX (struct semaphore *sem)
+static inline void init_MUTEX (struct semaphore *sem)
{
    sema_init(sem, 1);
}

-static __inline__ void init_MUTEX_LOCKED (struct semaphore *sem)
+static inline void init_MUTEX_LOCKED (struct semaphore *sem)
{
    sema_init(sem, 0);
}
@@ -120,19 +84,15 @@ asmlinkage int __down_trylock(struct se
asmlinkage void __up(struct semaphore * sem);

/*
- * This is ugly, but we want the default case to fall through.
- * "__down_failed" is a special asm handler that calls the C
- * routine that actually waits. See arch/i386/kernel/semaphore.c
+ * Atomically decrement the semaphore's count. If it goes negative,

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

+ * block the calling thread in the TASK_UNINTERRUPTIBLE state.
+ */
-static __inline__ void down(struct semaphore * sem)
+static inline void down(struct semaphore * sem)
{
    unsigned long flags;
-    int temp;
-
-#ifdef WAITQUEUE_DEBUG
-    CHECK_MAGIC(sem->__magic);
-#endif
+    long count;

+    might_sleep();
    local_irq_save(flags);
    __asm__ __volatile__ (
        "# down                                \n\t"
@@ -140,7 +100,7 @@ static __inline__ void down(struct semap
        LOAD"    %0, @%1;                        \n\t"
        "addi    %0, #-1;                        \n\t"
        STORE"   %0, @%1;                        \n\t"
-        : "=&r" (temp)
+        : "=&r" (count)
+        : "r" (&sem->count)
+        : "memory"
#ifdef CONFIG_CHIP_M32700_TS1
@@ -149,7 +109,7 @@ static __inline__ void down(struct semap
    );
    local_irq_restore(flags);

-    if (temp < 0)
+    if (unlikely(count < 0))
        __down(sem);
}

@@ -157,16 +117,13 @@ static __inline__ void down(struct semap
 * Interruptible try to acquire a semaphore.  If we obtained
 * it, return zero.  If we were interrupted, returns -EINTR
 */
-static __inline__ int down_interruptible(struct semaphore * sem)
+static inline int down_interruptible(struct semaphore * sem)
{
    unsigned long flags;
-    int temp;
+    long count;
    int result = 0;

-#ifdef WAITQUEUE_DEBUG
-    CHECK_MAGIC(sem->__magic);
-#endif
-
+    might_sleep();
    local_irq_save(flags);
    __asm__ __volatile__ (
        "# down_interruptible                    \n\t"
@@ -174,7 +131,7 @@ static __inline__ int down_interruptible
        LOAD"    %0, @%1;                        \n\t"
        "addi    %0, #-1;                        \n\t"
        STORE"   %0, @%1;                        \n\t"
-        : "=&r" (temp)
+        : "=&r" (count)
+        : "r" (&sem->count)

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

                : "memory"
#ifdef CONFIG_CHIP_M32700_TS1
@@ -183,7 +140,7 @@ static __inline__ int down_interruptible
    );
    local_irq_restore(flags);

-    if (temp < 0)
+    if (unlikely(count < 0))
        result = __down_interruptible(sem);

    return result;
@@ -193,16 +150,12 @@ static __inline__ int down_interruptible
    * Non-blockingly attempt to down() a semaphore.
    * Returns zero if we acquired it
    */
-static __inline__ int down_trylock(struct semaphore * sem)
+static inline int down_trylock(struct semaphore * sem)
    {
        unsigned long flags;
-    int temp;
+    long count;
        int result = 0;

-#ifndef WAITQUEUE_DEBUG
-    CHECK_MAGIC(sem->__magic);
-#endif
-
        local_irq_save(flags);
        __asm__ __volatile__ (
            "# down_trylock                \n\t"
@@ -210,7 +163,7 @@ static __inline__ int down_trylock(struc
        LOAD"    %0, @%1;                \n\t"
        "addi    %0, #-1;                \n\t"
        STORE"   %0, @%1;                \n\t"
-    : "=&r" (temp)
+    : "=&r" (count)
        : "r" (&sem->count)
        : "memory"
#ifdef CONFIG_CHIP_M32700_TS1
@@ -219,7 +172,7 @@ static __inline__ int down_trylock(struc
    );
    local_irq_restore(flags);

-    if (temp < 0)
+    if (unlikely(count < 0))
        result = __down_trylock(sem);

    return result;
@@ -231,14 +184,10 @@ static __inline__ int down_trylock(struc
    * The default case (no contention) will result in NO
    * jumps for both down() and up().
    */
-static __inline__ void up(struct semaphore * sem)
+static inline void up(struct semaphore * sem)
    {
        unsigned long flags;
-    int temp;
-
-#ifndef WAITQUEUE_DEBUG
-    CHECK_MAGIC(sem->__magic);
-#endif
+    long count;

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

        local_irq_save(flags);
        __asm__ __volatile__ (
@@ -247,7 +196,7 @@ static __inline__ void up(struct semapho
                LOAD"    %0, @%1;                \n\t"
                "addi   %0, #1;                \n\t"
                STORE"  %0, @%1;                \n\t"
-               : "=&r" (temp)
+               : "=&r" (count)
                : "r" (&sem->count)
                : "memory"
#ifdef CONFIG_CHIP_M32700_TS1
@@ -256,11 +205,10 @@ static __inline__ void up(struct semapho
        );
        local_irq_restore(flags);

-       if (temp <= 0)
+       if (unlikely(count <= 0))
                __up(sem);
    }

    #endif /* __KERNEL__ */

    #endif /* _ASM_M32R_SEMAPHORE_H */
-
diff -ruNp a/include/asm-m32r/spinlock.h b/include/asm-m32r/spinlock.h
--- a/include/asm-m32r/spinlock.h      2004-09-28 10:19:53.000000000 +0900
+++ b/include/asm-m32r/spinlock.h      2004-09-28 12:38:24.000000000 +0900
@@ -1,14 +1,12 @@
    #ifndef _ASM_M32R_SPINLOCK_H
    #define _ASM_M32R_SPINLOCK_H

-/* $Id$ */
-
    /*
    * linux/include/asm-m32r/spinlock.h
- *   orig : i386 2.4.10
    *
    * M32R version:
    *   Copyright (C) 2001, 2002 Hitoshi Yamamoto
+ *   Copyright (C) 2004 Hirokazu Takata <takata at linux-m32r.org>
    */

    #include <linux/config.h> /* CONFIG_DEBUG_SPINLOCK, CONFIG_SMP */
@@ -41,6 +39,9 @@ typedef struct {
    #if SPINLOCK_DEBUG
        unsigned magic;
    #endif
+ #ifdef CONFIG_PREEMPT
+     unsigned int break_lock;
+ #endif
    } spinlock_t;

    #define SPINLOCK_MAGIC 0xdead4ead
@@ -66,22 +67,17 @@ typedef struct {
    #define spin_unlock_wait(x)    do { barrier(); } while(spin_is_locked(x))
    #define _raw_spin_lock_flags(lock, flags) _raw_spin_lock(lock)

-/*
- * This works. Despite all the confusion.
+/**
+ * _raw_spin_trylock - Try spin lock and return a result

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

+ * @lock: Pointer to the lock variable
+ *
+ * _raw_spin_trylock() tries to get the lock and returns a result.
+ * On the m32r, the result value is 1 (= Success) or 0 (= Failure).
+ */
-
-/*=====
- * Try spin lock
- *=====
- * Argument:
- *   arg0: lock
- * Return value:
- *   =1: Success
- *   =0: Failure
- *=====*/
-static __inline__ int _raw_spin_trylock(spinlock_t *lock)
+static inline int _raw_spin_trylock(spinlock_t *lock)
{
    int oldval;
+    unsigned long tmp1, tmp2;

    /*
     * lock->lock : =1 : unlock
@@ -93,16 +89,16 @@ static __inline__ int _raw_spin_trylock(
    /*
     __asm__ __volatile__ (
-        "# spin_trylock                \n\t"
-        "ldi    r4, #0;                  \n\t"
-        "mvfc   r5, psw;                 \n\t"
+        "ldi    %1, #0;                  \n\t"
+        "mvfc   %2, psw;                 \n\t"
+        "clrpsw #0x40 -> nop;           \n\t"
-        DCACHE_CLEAR("%0", "r6", "%1")
-        "lock   %0, @%1;                 \n\t"
-        "unlock r4, @%1;                 \n\t"
-        "mvtc   r5, psw;                 \n\t"
-        : "=&r" (oldval)
+        DCACHE_CLEAR("%0", "r6", "%3")
+        "lock   %0, @%3;                 \n\t"
+        "unlock %1, @%3;                 \n\t"
+        "mvtc   %2, psw;                 \n\t"
+        : "=&r" (oldval), "=&r" (tmp1), "=&r" (tmp2)
+        : "r" (&lock->lock)
+        : "memory", "r4", "r5"
+        : "memory"
#ifdef CONFIG_CHIP_M32700_TS1
        , "r6"
#endif /* CONFIG_CHIP_M32700_TS1 */
@@ -111,8 +107,10 @@ static __inline__ int _raw_spin_trylock(
    return (oldval > 0);
}

-static __inline__ void _raw_spin_lock(spinlock_t *lock)
+static inline void _raw_spin_lock(spinlock_t *lock)
{
+    unsigned long tmp0, tmp1;
+
    #if SPINLOCK_DEBUG
        __label__ here;
    here:
@@ -135,31 +133,31 @@ here:
    # spin_lock                \n\t"

```

Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

        ".fillinsn                                \n"
        "1:                                       \n\t"
-       "mvfc   r5, psw;                          \n\t"
+       "mvfc   %1, psw;                          \n\t"
        "clrpsw #0x40 -> nop;                     \n\t"
-       DCACHE_CLEAR("r4", "r6", "%0")
        "lock   r4, @%0;                          \n\t"
-       "addi   r4, #-1;                          \n\t"
-       "unlock r4, @%0;                          \n\t"
-       "mvtc   r5, psw;                          \n\t"
-       "bltz   r4, 2f;                           \n\t"
+       DCACHE_CLEAR("%0", "r6", "%2")
+       "lock   %0, @%2;                          \n\t"
+       "addi   %0, #-1;                          \n\t"
+       "unlock %0, @%2;                          \n\t"
+       "mvtc   %1, psw;                          \n\t"
+       "bltz   %0, 2f;                           \n\t"
        LOCK_SECTION_START(".balign 4 \n\t")
        ".fillinsn                                \n"
        "2:                                       \n\t"
-       "ld     r4, @%0;                          \n\t"
-       "bgtz   r4, 1b;                          \n\t"
+       "ld     %0, @%2;                          \n\t"
+       "bgtz   %0, 1b;                          \n\t"
        "bra    2b;                               \n\t"
        LOCK_SECTION_END
-       : /* no outputs */
+       : "&r" (tmp0), "&r" (tmp1)
        : "r" (&lock->lock)
-       : "memory", "r4", "r5"
+       : "memory"
#ifdef CONFIG_CHIP_M32700_TS1
        , "r6"
#endif /* CONFIG_CHIP_M32700_TS1 */
    );
}

-static __inline__ void _raw_spin_unlock(spinlock_t *lock)
+static inline void _raw_spin_unlock(spinlock_t *lock)
{
    #if SPINLOCK_DEBUG
        BUG_ON(lock->magic != SPINLOCK_MAGIC);
@@ -184,6 +182,9 @@ typedef struct {
    #if SPINLOCK_DEBUG
        unsigned magic;
    #endif
+ifdef CONFIG_PREEMPT
+    unsigned int break_lock;
+endif
} rwlock_t;

#define RWLOCK_MAGIC 0xdeafleed
@@ -211,8 +212,10 @@ typedef struct {
    */
    /* the spinlock helpers are in arch/i386/kernel/semaphore.c */

-static __inline__ void _raw_read_lock(rwlock_t *rw)
+static inline void _raw_read_lock(rwlock_t *rw)
{
+    unsigned long tmp0, tmp1;
+
    #if SPINLOCK_DEBUG

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

        BUG_ON(rw->magic != RWLOCK_MAGIC);
#endif
@@ -231,40 +234,42 @@ static __inline__ void _raw_read_lock(rw
        "# read_lock                                \n\t"
        ".fillinsn                                  \n"
        "1:                                           \n\t"
-       "mvfc   r5, psw;                               \n\t"
+       "mvfc   %1, psw;                               \n\t"
        "clrpsw #0x40 -> nop;                         \n\t"
-       DCACHE_CLEAR("r4", "r6", "%0")
-       "lock   r4, @%0;                               \n\t"
-       "addi   r4, #-1;                               \n\t"
-       "unlock r4, @%0;                               \n\t"
-       "mvtc   r5, psw;                               \n\t"
-       "bltz   r4, 2f;                                 \n\t"
+       DCACHE_CLEAR("%0", "r6", "%2")
+       "lock   %0, @%2;                               \n\t"
+       "addi   %0, #-1;                               \n\t"
+       "unlock %0, @%2;                               \n\t"
+       "mvtc   %1, psw;                               \n\t"
+       "bltz   %0, 2f;                                 \n\t"
        LOCK_SECTION_START(".balign 4 \n\t")
        ".fillinsn                                  \n"
        "2:                                           \n\t"
        "clrpsw #0x40 -> nop;                         \n\t"
-       DCACHE_CLEAR("r4", "r6", "%0")
-       "lock   r4, @%0;                               \n\t"
-       "addi   r4, #1;                                \n\t"
-       "unlock r4, @%0;                               \n\t"
-       "mvtc   r5, psw;                               \n\t"
+       DCACHE_CLEAR("%0", "r6", "%2")
+       "lock   %0, @%2;                               \n\t"
+       "addi   %0, #1;                                \n\t"
+       "unlock %0, @%2;                               \n\t"
+       "mvtc   %1, psw;                               \n\t"
        ".fillinsn                                  \n"
        "3:                                           \n\t"
-       "ld     r4, @%0;                               \n\t"
-       "bgtz   r4, 1b;                               \n\t"
+       "ld     %0, @%2;                               \n\t"
+       "bgtz   %0, 1b;                               \n\t"
+       "bra    3b;                                    \n\t"
        LOCK_SECTION_END
        : /* no outputs */
+       : "=&r" (tmp0), "=&r" (tmp1)
        : "r" (&rw->lock)
-       : "memory", "r4", "r5"
+       : "memory"
#ifdef CONFIG_CHIP_M32700_TS1
        , "r6"
#endif /* CONFIG_CHIP_M32700_TS1 */
    );
}

-static __inline__ void _raw_write_lock(rwlock_t *rw)
+static inline void _raw_write_lock(rwlock_t *rw)
{
    unsigned long tmp0, tmp1, tmp2;
+
    #if SPINLOCK_DEBUG
        BUG_ON(rw->magic != RWLOCK_MAGIC);
    #endif

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

@@ -281,85 +286,91 @@ static __inline__ void _raw_write_lock(r
    */
    __asm__ __volatile__ (
        "# write_lock                                \n\t"
-       "seth   r5, #high(" RW_LOCK_BIAS_STR ");      \n\t"
-       "or3    r5, r5, #low(" RW_LOCK_BIAS_STR ");   \n\t"
+       "seth   %1, #high(" RW_LOCK_BIAS_STR ");      \n\t"
+       "or3    %1, %1, #low(" RW_LOCK_BIAS_STR ");   \n\t"
        ".fillinsn                                    \n"
        "1:                                           \n\t"
-       "mvfc   r6, psw;                               \n\t"
+       "mvfc   %2, psw;                               \n\t"
        "clrpsw #0x40 -> nop;                          \n\t"
        DCACHE_CLEAR("r4", "r7", "%0")
-       "lock   r4, @%0;                               \n\t"
-       "sub    r4, r5;                               \n\t"
-       "unlock r4, @%0;                              \n\t"
-       "mvtc   r6, psw;                               \n\t"
-       "bnez   r4, 2f;                               \n\t"
+       DCACHE_CLEAR("%0", "r7", "%3")
+       "lock   %0, @%3;                               \n\t"
+       "sub    %0, %1;                               \n\t"
+       "unlock %0, @%3;                              \n\t"
+       "mvtc   %2, psw;                               \n\t"
+       "bnez   %0, 2f;                               \n\t"
        LOCK_SECTION_START(".balign 4 \n\t")
        ".fillinsn                                    \n"
        "2:                                           \n\t"
        "clrpsw #0x40 -> nop;                          \n\t"
-       DCACHE_CLEAR("r4", "r7", "%0")
-       "lock   r4, @%0;                               \n\t"
-       "add    r4, r5;                               \n\t"
-       "unlock r4, @%0;                              \n\t"
-       "mvtc   r6, psw;                               \n\t"
+       DCACHE_CLEAR("%0", "r7", "%3")
+       "lock   %0, @%3;                               \n\t"
+       "add    %0, %1;                               \n\t"
+       "unlock %0, @%3;                              \n\t"
+       "mvtc   %2, psw;                               \n\t"
        ".fillinsn                                    \n"
        "3:                                           \n\t"
-       "ld     r4, @%0;                               \n\t"
-       "beq    r4, r5, 1b;                             \n\t"
+       "ld     %0, @%3;                               \n\t"
+       "beq    %0, %1, 1b;                             \n\t"
+       "bra    3b;                                     \n\t"
        LOCK_SECTION_END
-       : /* no outputs */
+       : "=&r" (tmp0), "=&r" (tmp1), "=&r" (tmp2)
        : "r" (&rw->lock)
-       : "memory", "r4", "r5", "r6"
+       : "memory"
#ifdef CONFIG_CHIP_M32700_TS1
        , "r7"
#endif /* CONFIG_CHIP_M32700_TS1 */
    );
}

-static __inline__ void _raw_read_unlock(rwlock_t *rw)
+static inline void _raw_read_unlock(rwlock_t *rw)
{
+   unsigned long tmp0, tmp1;

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```

+
+     __asm__ __volatile__ (
-         "# read_unlock                                \n\t"
+         "mvfc   r5, psw;                               \n\t"
-         "mvfc   %1, psw;                               \n\t"
+         "clrpsw #0x40 -> nop;                          \n\t"
-         DCACHE_CLEAR("r4", "r6", "%0")
-         "lock   r4, @%0;                               \n\t"
-         "addi   r4, #1;                                 \n\t"
-         "unlock r4, @%0;                               \n\t"
-         "mvtc   r5, psw;                               \n\t"
-         : /* no outputs */
+         DCACHE_CLEAR("%0", "r6", "%2")
+         "lock   %0, @%2;                               \n\t"
+         "addi   %0, #1;                                 \n\t"
+         "unlock %0, @%2;                               \n\t"
+         "mvtc   %1, psw;                               \n\t"
+         : "=&r" (tmp0), "=&r" (tmp1)
-         : "r" (&rw->lock)
+         : "memory", "r4", "r5"
+         : "memory"
+     #ifdef CONFIG_CHIP_M32700_TS1
+         , "r6"
+     #endif /* CONFIG_CHIP_M32700_TS1 */
+     );
+ }

-static __inline__ void _raw_write_unlock(rwlock_t *rw)
+static inline void _raw_write_unlock(rwlock_t *rw)
+ {
+     unsigned long tmp0, tmp1, tmp2;
+
+     __asm__ __volatile__ (
-         "# write_unlock                                \n\t"
+         "seth   r5, #high(" RW_LOCK_BIAS_STR ");       \n\t"
-         "or3    r5, r5, #low(" RW_LOCK_BIAS_STR ");    \n\t"
+         "mvfc   r6, psw;                               \n\t"
+         "seth   %1, #high(" RW_LOCK_BIAS_STR ");       \n\t"
+         "or3    %1, %1, #low(" RW_LOCK_BIAS_STR ");    \n\t"
+         "mvfc   %2, psw;                               \n\t"
+         "clrpsw #0x40 -> nop;                          \n\t"
-         DCACHE_CLEAR("r4", "r7", "%0")
-         "lock   r4, @%0;                               \n\t"
-         "add    r4, r5;                                 \n\t"
-         "unlock r4, @%0;                               \n\t"
-         "mvtc   r6, psw;                               \n\t"
-         : /* no outputs */
+         DCACHE_CLEAR("%0", "r7", "%3")
+         "lock   %0, @%3;                               \n\t"
+         "add    %0, %1;                                 \n\t"
+         "unlock %0, @%3;                               \n\t"
+         "mvtc   %2, psw;                               \n\t"
+         : "=&r" (tmp0), "=&r" (tmp1), "=&r" (tmp2)
-         : "r" (&rw->lock)
+         : "memory", "r4", "r5", "r6"
+         : "memory"
+     #ifdef CONFIG_CHIP_M32700_TS1
+         , "r7"
+     #endif /* CONFIG_CHIP_M32700_TS1 */
+     );
+ }

```

## Linux-Kernel: [PATCH 2.6.9-rc2-mm4] [m32r] Change to use temporary register variables

```
-static __inline__ int _raw_write_trylock(rwlock_t *lock)
+#define _raw_read_trylock(lock) generic_raw_read_trylock(lock)
+
+static inline int _raw_write_trylock(rwlock_t *lock)
+{
+    atomic_t *count = (atomic_t *)lock;
+    if (atomic_sub_and_test(RW_LOCK_BIAS, count))
+
--
```

Hirokazu Takata <takata@linux-m32r.org>

Linux/M32R Project: <http://www.linux-m32r.org/>

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>