```
**********************************************************
    5396 Sat Feb  7 18:57:44 2015
new/usr/src/uts/armv6/ml/cache.s
armv6: bit 2 (0x4) enables the dcache
This fixes a pretty simple typo.  Sadly, this still isn't enough to get
bcm2835 past mutex_enter.
**********************************************************
______unchanged_portion_omitted_

        ENTRY(armv6_dcache_enable)
        mrc     p15, 0, r0, c1, c0, 0
        orr     r0, #0x4
        orr     r0, #0x2
        mcr     p15, 0, r0, c1, c0, 0
        SET_SIZE(armv6_dcache_enable)
______unchanged_portion_omitted_

        ENTRY(armv6_dcache_disable)
        mrc     p15, 0, r0, c1, c0, 0
        bic     r0, #0x4
        bic     r0, #0x2
        mcr     p15, 0, r0, c1, c0, 0
        SET_SIZE(armv6_dcache_disable)
______unchanged_portion_omitted_
```

```
**********************************************************
    3435 Sat Feb  7 18:57:44 2015
new/usr/src/uts/armv6/ml/glocore.s
armv6: bit 2 (0x4) enables the dcache
This fixes a pretty simple typo.  Sadly, this still isn't enough to get
bcm2835 past mutex_enter.
**********************************************************
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2013 (c) Joyent, Inc. All rights reserved.
 * Copyright (c) 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
 */

#include <sys/asm_linkage.h>
#include <sys/machparam.h>
#include <sys/cpu_asm.h>

#include "assym.h"

#if defined(__lint)

#endif

/*
 * Each of the different machines has its own locore.s to take care of getting
 * us into fakebop for the first time. After that, they all return here to a
 * generic locore to take us into mlsetup and then to main forever more.
 */

        /*
         * External globals
         */
        .globl  _locore_start
        .globl  mlsetup
        .globl  sysp
        .globl  bootops
        .globl  bootopsp
        .globl  t0

        .data
        .comm   t0stack, DEFAULTSTKSZ, 32
        .comm   t0, 4094, 32

#if defined(__lint)

/* ARGSUSED */
void
_locore_start(struct boot_syscalls *sysp, struct bootops *bop)
{}

#else   /* __lint */

        /*
         * We got here from _kobj_init() via exitto().  We have a few different
         * tasks that we need to take care of before we hop into mlsetup and
         * then main. We're never going back so we shouldn't feel compelled to
```

```
         * preserve any registers.
         *
         *  o Enable unaligned access
         *  o Enable our I/D-caches
         *  o Save the boot syscalls and bootops for later
         *  o Set up our stack to be the real stack of t0stack.
         *  o Save t0 as curthread
         *  o Set up a struct REGS for mlsetup
         *  o Make sure that we're 8 byte aligned for the call
         */

        ENTRY(_locore_start)


        /*
         * We've been running in t0stack anyway, up to this point, but
         * _locore_start represents what is in effect a fresh start in the
         * real kernel -- We'll never return back through here.
         *
         * So reclaim those few bytes
         */
        ldr     sp, =t0stack
        ldr     r4, =(DEFAULTSTKSZ - REGSIZE)
        add     sp, r4
        bic     sp, sp, #0xff

        /*
         * Save flags and arguments for potential debugging
         */
        str     r0, [sp, #REGOFF_R0]
        str     r1, [sp, #REGOFF_R1]
        str     r2, [sp, #REGOFF_R2]
        str     r3, [sp, #REGOFF_R3]
        mrs     r4, CPSR
        str     r4, [sp, #REGOFF_CPSR]

        /*
         * Save back the bootops and boot_syscalls.
         */
        ldr     r2, =sysp
        str     r0, [r2]
        ldr     r2, =bootops
        str     r1, [r2]
        ldr     r2, =bootopsp
        ldr     r2, [r2]
        str     r1, [r2]

        /*
         * Set up our curthread pointer
         */
        ldr     r0, =t0
        mcr     p15, 0, r0, c13, c0, 4

        /*
         * Go ahead now and enable the L1 I/D caches.
         * Go ahead now and enable unaligned access, the L1 I/D caches.
         *
         * Bit 2 is for the D cache
         * Bit 12 is for the I cache
         * Bit 22 is for unaligned access
         */
        mrc     p15, 0, r0, c1, c0, 0
        orr     r0, #0x04       /* D-cache */
        orr     r0, #0x1000     /* I-cache */
        orr     r0, #0x02
        orr     r0, #0x1000
        orr     r0, #0x400000
        mcr     p15, 0, r0, c1, c0, 0

        /*
         * mlsetup() takes the struct regs as an argument. main doesn't take
         * any and should never return. Currently, we have an 8-byte aligned
         * stack.  We want to push a zero frame pointer to terminate any
         * stack walking, but that would cause us to end up with only a
         * 4-byte aligned stack.  So, to keep things nice and correct, we
         * push a zero value twice - it's similar to a typical function
         * entry:
         *      push { r9, lr }
         */
        mov     r9,#0
        push    { r9 }          /* link register */
        push    { r9 }          /* frame pointer */
        mov     r0, sp
        bl      mlsetup
        bl      main
        /* NOTREACHED */
        ldr     r0,=__return_from_main
        ldr     r0,[r0]
        bl      panic
        SET_SIZE(_locore_start)
```
______***unchanged_portion_omitted_***