```
**********************************************************
     811 Sat Feb  7 18:57:48 2015
new/usr/src/uts/armv6/bcm2835/ml/locore.s
armv6: bcm2835 & qvpb have nearly identical locore _start
It makes sense to common-ize _start for all armv6 machines.  They will all
have to do the same basic setup.  If there is any machine specific setup
they need to do, they can do so in the new _mach_start function.
**********************************************************
```

```
/*
 * 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 2015 (c) Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
 */

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

        ENTRY(_mach_start)
/*
 * Every story needs a beginning. This is ours.
 */

/*
 * We are in a primordial world here. The BMC2835 is going to come along and
 * boot us at _start. Normally we would go ahead and use a main() function, but
 * for now, we'll do that ourselves. As we've started the world, we also need to
 * set up a few things about us, for example our stack pointer. To help us out,
 * it's useful to remember the rough memory map. Remember, this is for physcial
 * addresses. There is no virtual memory here. These sizes are often manipulated
 * by the 'configuration' in the bootloader.
 *
 * +----------------+ <---- Max physical memory
 * |                |
 * |                |
 * |                |
 * +----------------+
 * |                |
 * |      I/O       |
 * |  Peripherals   |
 * |                |
 * +----------------+ <---- I/O base 0x20000000 (corresponds to 0x7E000000)
 * |                |
 * |     Main       |
 * |    Memory      |
 * |                |
 * +----------------+ <---- Top of SDRAM
 * |                |
 * |       VC       |
 * |     SDRAM      |
 * |                |
 * +----------------+ <---- Split determined by bootloader config
 * |                |
 * |      ARM       |
 * |     SDRAM      |
 * |                |
```



```
 * +----------------+ <---- Bottom of physical memory 0x00000000
 *
 * With the Raspberry Pi Model B, we have 512 MB of SDRAM. That means we have a
 * range of addresses from [0, 0x20000000). If we assume that the minimum amount
 * of DRAM is given to the GPU - 32 MB, that means we really have the following
 * range: [0, 0x1e000000).
 *
 * By default, this binary will be loaded into 0x8000. For now, that means we
 * will set our initial stack to 0x10000000.
 */

/*
 * Recall that _start is the traditional entry point for an ELF binary.
 */
        ENTRY(_start)
        ldr sp, =t0stack
        ldr r4, =DEFAULTSTKSZ
        add sp, r4
        bic sp, sp, #0xff

        /*
         * establish bogus stacks for exceptional CPU states, our exception
         * code should never make use of these, and we want loud and violent
         * failure should we accidentally try.
         */
        cps #(CPU_MODE_UND)
        mov sp, #-1
        cps #(CPU_MODE_ABT)
        mov sp, #-1
        cps #(CPU_MODE_FIQ)
        mov sp, #-1
        cps #(CPU_MODE_IRQ)
        mov sp, #-1
        cps #(CPU_MODE_SVC)

        /* Enable highvecs (moves the base of the exception vector) */
        mrc     p15, 0, r3, c1, c0, 0
        mov     r4, #1
        lsl     r4, r4, #13
        orr     r3, r3, r4
        mcr     p15, 0, r3, c1, c0, 0

        /* Enable access to p10 and p11 (privileged mode only) */
        mrc     p15, 0, r0, c1, c0, 2
        orr     r0, #0x00500000
        mcr     p15, 0, r0, c1, c0, 2

        bx      r14
        SET_SIZE(_mach_start)
        bl _fakebop_start
        SET_SIZE(_start)

        ENTRY(arm_reg_read)
        ldr r0, [r0]
        bx lr
        SET_SIZE(arm_reg_read)

        ENTRY(arm_reg_write)
        str r1, [r0]
        bx lr
        SET_SIZE(arm_reg_write)
```

```
**********************************************************
    4958 Sat Feb  7 18:57:48 2015
new/usr/src/uts/armv6/ml/glocore.s
armv6: bcm2835 & qvpb have nearly identical locore _start
It makes sense to common-ize _start for all armv6 machines.  They will all
have to do the same basic setup.  If there is any machine specific setup
they need to do, they can do so in the new _mach_start function.
**********************************************************
```

```
/*
 * 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"

/*
 * Every story needs a beginning. This is ours.
 */
#if defined(__lint)

/*
 * Each of the different machines has its own locore.s to take care of getting
 * the machine specific setup done.  Just before jumping into fakebop the
 * first time, we call this machine specific code.
 */
#endif

/*
 * We are in a primordial world here. The loader is going to come along and
 * boot us at _start. As we've started the world, we also need to set up a
 * few things about us, for example our stack pointer. To help us out, it's
 * useful to remember what the loader set up for us:
 *
 * - unaligned access are allowed (A = 0, U = 1)
 * - virtual memory is enabled
 *   - we (unix) are mapped right were we want to be
 *   - a UART has been enabled & any memory mapped registers have been 1:1
 *     mapped
 *   - ATAGs have been updated to tell us what the mappings are
 * - I/D L1 caches have been enabled
 * 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

/*
 * Recall that _start is the traditional entry point for an ELF binary.
 */
        ENTRY(_start)
        ldr     sp, =t0stack
        ldr     r4, =DEFAULTSTKSZ
        add     sp, r4
        bic     sp, sp, #0xff

        /*
         * establish bogus stacks for exceptional CPU states, our exception
         * code should never make use of these, and we want loud and violent
         * failure should we accidentally try.
         */
        cps     #(CPU_MODE_UND)
        mov     sp, #-1
        cps     #(CPU_MODE_ABT)
        mov     sp, #-1
        cps     #(CPU_MODE_FIQ)
        mov     sp, #-1
        cps     #(CPU_MODE_IRQ)
        mov     sp, #-1
        cps     #(CPU_MODE_SVC)

        /* Enable highvecs (moves the base of the exception vector) */
        mrc     p15, 0, r3, c1, c0, 0
        mov     r4, #1
        lsl     r4, r4, #13
        orr     r3, r3, r4
        mcr     p15, 0, r3, c1, c0, 0

        /* invoke machine specific setup */
        bl      _mach_start

        bl      _fakebop_start
        SET_SIZE(_start)


#endif /* ! codereview */
#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 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.
```

```
120          *  o Save t0 as curthread
121          *  o Set up a struct REGS for mlsetup
122          *  o Make sure that we're 8 byte aligned for the call
123          */
124
125         ENTRY(_locore_start)
126
127
128         /*
129          * We've been running in t0stack anyway, up to this point, but
130          * _locore_start represents what is in effect a fresh start in the
131          * real kernel -- We'll never return back through here.
132          *
133          * So reclaim those few bytes
134          */
135         ldr     sp, =t0stack
136         ldr     r4, =(DEFAULTSTKSZ - REGSIZE)
137         add     sp, r4
138         bic     sp, sp, #0xff
139
140         /*
141          * Save flags and arguments for potential debugging
142          */
143         str     r0, [sp, #REGOFF_R0]
144         str     r1, [sp, #REGOFF_R1]
145         str     r2, [sp, #REGOFF_R2]
146         str     r3, [sp, #REGOFF_R3]
147         mrs     r4, CPSR
148         str     r4, [sp, #REGOFF_CPSR]
149
150         /*
151          * Save back the bootops and boot_syscalls.
152          */
153         ldr     r2, =sysp
154         str     r0, [r2]
155         ldr     r2, =bootops
156         str     r1, [r2]
157         ldr     r2, =bootopsp
158         ldr     r2, [r2]
159         str     r1, [r2]
160
161         /*
162          * Set up our curthread pointer
163          */
164         ldr     r0, =t0
165         mcr     p15, 0, r0, c13, c0, 4
166
167         /*
168          * Go ahead now and enable the L1 I/D caches.
169          */
170         mrc     p15, 0, r0, c1, c0, 0
171         orr     r0, #0x04       /* D-cache */
172         orr     r0, #0x1000     /* I-cache */
173         mcr     p15, 0, r0, c1, c0, 0
174
175         /*
176          * mlsetup() takes the struct regs as an argument. main doesn't take
177          * any and should never return. Currently, we have an 8-byte aligned
178          * stack.  We want to push a zero frame pointer to terminate any
179          * stack walking, but that would cause us to end up with only a
180          * 4-byte aligned stack.  So, to keep things nice and correct, we
181          * push a zero value twice - it's similar to a typical function
182          * entry:
183          *      push { r9, lr }
184          */
185         mov     r9,#0
```



```
186         push    { r9 }          /* link register */
187         push    { r9 }          /* frame pointer */
188         mov     r0, sp
189         bl      mlsetup
190         bl      main
191         /* NOTREACHED */
192         ldr     r0,=__return_from_main
193         ldr     r0,[r0]
194         bl      panic
195         SET_SIZE(_locore_start)
196
197 __return_from_main:
198         .string "main() returned"
199 #endif  /* __lint */
200
201         ENTRY(arm_reg_read)
202         ldr r0, [r0]
203         bx lr
204         SET_SIZE(arm_reg_read)
205
206         ENTRY(arm_reg_write)
207         str r1, [r0]
208         bx lr
209         SET_SIZE(arm_reg_write)
210 #endif /* ! codereview */
```

```
**********************************************************
     697 Sat Feb  7 18:57:48 2015
new/usr/src/uts/armv6/qvpb/ml/locore.s
armv6: bcm2835 & qvpb have nearly identical locore _start
It makes sense to common-ize _start for all armv6 machines.  They will all
have to do the same basic setup.  If there is any machine specific setup
they need to do, they can do so in the new _mach_start function.
**********************************************************
```

```
/*
 * 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 2015 (c) Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
#endif /* ! codereview */
 */

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

        ENTRY(_mach_start)
        /* nothing to do */
        bx      r14
        SET_SIZE(_mach_start)
/*
 * Every story needs a beginning. This is ours.
 */

/*
 * We are in a primordial world here. The BMC2835 is going to come along and
 * boot us at _start. Normally we would go ahead and use a main() function, but
 * for now, we'll do that ourselves. As we've started the world, we also need to
 * set up a few things about us, for example our stack pointer. To help us out,
 * it's useful to remember the rough memory map. Remember, this is for physcial
 * addresses. There is no virtual memory here. These sizes are often manipulated
 * by the 'configuration' in the bootloader.
 *
 * +----------------+ <---- Max physical memory
 * |                |
 * |                |
 * |                |
 * +----------------+
 * |                |
 * |      I/O       |
 * |  Peripherals   |
 * |                |
 * +----------------+ <---- I/O base 0x20000000 (corresponds to 0x7E000000)
 * |                |
 * |     Main       |
 * |    Memory      |
 * |                |
 * +----------------+ <---- Top of SDRAM
 * |                |
 * |       VC       |
 * |     SDRAM      |
 * |                |
 * +----------------+ <---- Split determined by bootloader config
```



```
 * |                |
 * |      ARM       |
 * |     SDRAM      |
 * |                |
 * +----------------+ <---- Bottom of physical memory 0x00000000
 *
 * With the Raspberry Pi Model B, we have 512 MB of SDRAM. That means we have a
 * range of addresses from [0, 0x20000000). If we assume that the minimum amount
 * of DRAM is given to the GPU - 32 MB, that means we really have the following
 * range: [0, 0x1e000000).
 *
 * By default, this binary will be loaded into 0x8000. For now, that means we
 * will set our initial stack to 0x10000000.
 */

/*
 * Recall that _start is the traditional entry point for an ELF binary.
 */
        ENTRY(_start)
        ldr sp, =t0stack
        ldr r4, =DEFAULTSTKSZ
        add sp, r4
        bic sp, sp, #0xff

        /*
         * establish bogus stacks for exceptional CPU states, our exception
         * code should never make use of these, and we want loud and violent
         * failure should we accidentally try.
         */
        cps #(CPU_MODE_UND)
        mov sp, #-1
        cps #(CPU_MODE_ABT)
        mov sp, #-1
        cps #(CPU_MODE_FIQ)
        mov sp, #-1
        cps #(CPU_MODE_IRQ)
        mov sp, #-1
        cps #(CPU_MODE_SVC)

        /* Enable highvecs (moves the base of the exception vector) */
        mrc     p15, 0, r3, c1, c0, 0
        mov     r4, #1
        lsl     r4, r4, #13
        orr     r3, r3, r4
        mcr     p15, 0, r3, c1, c0, 0

        bl _fakebop_start
        SET_SIZE(_start)

        ENTRY(arm_reg_read)
        ldr r0, [r0]
        bx lr
        SET_SIZE(arm_reg_read)

        ENTRY(arm_reg_write)
        str r1, [r0]
        bx lr
        SET_SIZE(arm_reg_write)
```