```
/*
 *  linux/arch/arm/kernel/head.S
 *
 *  Copyright (C) 1994-2002 Russell King
 *  Copyright (c) 2003 ARM Limited
 *  All Rights Reserved
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 *
 *  Kernel startup code for all 32-bit CPUs
 */
#include <linux/linkage.h>
#include <linux/init.h>

#include <asm/assembler.h>
#include <asm/domain.h>
#include <asm/ptrace.h>
#include <asm/asm-offsets.h>
#include <asm/memory.h>
#include <asm/thread_info.h>
#include <asm/system.h>

#if (PHYS_OFFSET & 0x001fffff)
#error "PHYS_OFFSET must be at an even 2MiB boundary!"
#endif

#define KERNEL_RAM_VADDR    (PAGE_OFFSET + TEXT_OFFSET)
#define KERNEL_RAM_PADDR    (PHYS_OFFSET + TEXT_OFFSET)


/*
 * swapper_pg_dir is the virtual address of the initial page table.
 * We place the page tables 16K below KERNEL_RAM_VADDR.  Therefore, we must
 * make sure that KERNEL_RAM_VADDR is correctly set.  Currently, we expect
 * the least significant 16 bits to be 0x8000, but we could probably
 * relax this restriction to KERNEL_RAM_VADDR >= PAGE_OFFSET + 0x4000.
 */
#if (KERNEL_RAM_VADDR & 0xffff) != 0x8000
#error KERNEL_RAM_VADDR must start at 0xXXXX8000
#endif

    .globl  swapper_pg_dir
    .equ    swapper_pg_dir, KERNEL_RAM_VADDR - 0x4000

    .macro  pgtbl, rd
    ldr \rd, =(KERNEL_RAM_PADDR - 0x4000)
    .endm

#ifdef CONFIG_XIP_KERNEL
#define KERNEL_START    XIP_VIRT_ADDR(CONFIG_XIP_PHYS_ADDR)
#define KERNEL_END  _edata_loc
#else
```

```
#define KERNEL_START    KERNEL_RAM_VADDR
#define KERNEL_END  _end
#endif

/*
 * Kernel startup entry point.
 * ---------------------------
 *
 * This is normally called from the decompressor code.  The requirements
 * are: MMU = off, D-cache = off, I-cache = dont care, r0 = 0,
 * r1 = machine nr, r2 = atags pointer.
 *
 * This code is mostly position independent, so if you link the kernel at
 * 0xc0008000, you call this at __pa(0xc0008000).
 *
 * See linux/arch/arm/tools/mach-types for the complete list of machine
 * numbers for r1.
 *
 * We're trying to keep crap to a minimum; DO NOT add any machine specific
 * crap here - that's what the boot loader (or in extreme, well justified
 * circumstances, zImage) is for.
 */
    .section ".text.head", "ax"
ENTRY(stext)
    msr cpsr_c, #PSR_F_BIT | PSR_I_BIT | SVC_MODE @ ensure svc mode
                        @ and irqs disabled
    mrc p15, 0, r9, c0, c0      @ get processor id
    bl  __lookup_processor_type     @ r5=procinfo r9=cpuid
    movs    r10, r5             @ invalid processor (r5=0)?
    beq __error_p           @ yes, error 'p'
    bl  __lookup_machine_type       @ r5=machinfo
    movs    r8, r5              @ invalid machine (r5=0)?
    beq __error_a           @ yes, error 'a'
    bl  __vet_atags
    bl  __create_page_tables

    /*
     * The following calls CPU specific code in a position independent
     * manner.  See arch/arm/mm/proc-*.S for details.  r10 = base of
     * xxx_proc_info structure selected by __lookup_machine_type
     * above.  On return, the CPU will be ready for the MMU to be
     * turned on, and r0 will hold the CPU control register value.
     */
    ldr r13, __switch_data      @ address to jump to after
                        @ mmu has been enabled
    adr lr, __enable_mmu        @ return (PIC) address
    add pc, r10, #PROCINFO_INITFUNC
ENDPROC(stext)

#if defined(CONFIG_SMP)
ENTRY(secondary_startup)
    /*
     * Common entry point for secondary CPUs.
     *
```

```
	 * Ensure that we're in SVC mode, and IRQs are disabled.  Lookup
	 * the processor type - there is no need to check the machine type
	 * as it has already been validated by the primary processor.
	 */
	msr	cpsr_c, #PSR_F_BIT | PSR_I_BIT | SVC_MODE
	mrc	p15, 0, r9, c0, c0		@ get processor id
	bl	__lookup_processor_type
	movs	r10, r5				@ invalid processor?
	moveq	r0, #'p'			@ yes, error 'p'
	beq	__error

	/*
	 * Use the page tables supplied from  __cpu_up.
	 */
	adr	r4, __secondary_data
	ldmia	r4, {r5, r7, r13}		@ address to jump to after
	sub	r4, r4, r5			@ mmu has been enabled
	ldr	r4, [r7, r4]			@ get secondary_data.pgdir
	adr	lr, __enable_mmu		@ return address
	add	pc, r10, #PROCINFO_INITFUNC	@ initialise processor
						@ (return control reg)
ENDPROC(secondary_startup)

	/*
	 * r6  = &secondary_data
	 */
ENTRY(__secondary_switched)
	ldr	sp, [r7, #4]			@ get secondary_data.stack
	mov	fp, #0
	b	secondary_start_kernel
ENDPROC(__secondary_switched)

	.type	__secondary_data, %object
__secondary_data:
	.long	.
	.long	secondary_data
	.long	__secondary_switched
#endif /* defined(CONFIG_SMP) */



/*
 * Setup common bits before finally enabling the MMU.  Essentially
 * this is just loading the page table pointer and domain access
 * registers.
 */
__enable_mmu:
#ifdef CONFIG_ALIGNMENT_TRAP
	orr	r0, r0, #CR_A
#else
	bic	r0, r0, #CR_A
#endif
#ifdef CONFIG_CPU_DCACHE_DISABLE
	bic	r0, r0, #CR_C
```

```
#endif
#ifdef CONFIG_CPU_BPREDICT_DISABLE
    bic r0, r0, #CR_Z
#endif
#ifdef CONFIG_CPU_ICACHE_DISABLE
    bic r0, r0, #CR_I
#endif
    mov r5, #(domain_val(DOMAIN_USER, DOMAIN_MANAGER) | \
              domain_val(DOMAIN_KERNEL, DOMAIN_MANAGER) | \
              domain_val(DOMAIN_TABLE, DOMAIN_MANAGER) | \
              domain_val(DOMAIN_IO, DOMAIN_CLIENT))
    mcr p15, 0, r5, c3, c0, 0       @ load domain access register
    mcr p15, 0, r4, c2, c0, 0       @ load page table pointer
    b   __turn_mmu_on
ENDPROC(__enable_mmu)

/*
 * Enable the MMU.  This completely changes the structure of the visible
 * memory space.  You will not be able to trace execution through this.
 * If you have an enquiry about this, *please* check the linux-arm-kernel
 * mailing list archives BEFORE sending another post to the list.
 *
 *  r0  = cp#15 control register
 *  r13 = *virtual* address to jump to upon completion
 *
 * other registers depend on the function called upon completion
 */
    .align  5
__turn_mmu_on:
    mov r0, r0
    mcr p15, 0, r0, c1, c0, 0       @ write control reg
    mrc p15, 0, r3, c0, c0, 0       @ read id reg
    mov r3, r3
    mov r3, r3
    mov pc, r13
ENDPROC(__turn_mmu_on)



/*
 * Setup the initial page tables.  We only setup the barest
 * amount which are required to get the kernel running, which
 * generally means mapping in the kernel code.
 *
 * r8  = machinfo
 * r9  = cpuid
 * r10 = procinfo
 *
 * Returns:
 *  r0, r3, r6, r7 corrupted
 *  r4 = physical page table address
 */
__create_page_tables:
    pgtbl   r4              @ page table address
    /*
     * Clear the 16K level 1 swapper page table
     */
    mov r0, r4
    mov r3, #0
    add r6, r0, #0x4000
1:  str r3, [r0], #4
    str r3, [r0], #4
    str r3, [r0], #4
    str r3, [r0], #4
    teq r0, r6
    bne 1b

    ldr r7, [r10, #PROCINFO_MM_MMUFLAGS] @ mm_mmuflags

    /*
     * Create identity mapping for first MB of kernel to
     * cater for the MMU enable.  This identity mapping
     * will be removed by paging_init().  We use our current program
     * counter to determine corresponding section base address.
     */
    mov r6, pc, lsr #20         @ start of kernel section
    orr r3, r7, r6, lsl #20     @ flags + kernel base
    str r3, [r4, r6, lsl #2]        @ identity mapping

    /*
     * Now setup the pagetables for our kernel direct
     * mapped region.
     */
    add r0, r4,  #(KERNEL_START & 0xff000000) >> 18
    str r3, [r0, #(KERNEL_START & 0x00f00000) >> 18]!
    ldr r6, =(KERNEL_END - 1)
    add r0, r0, #4
    add r6, r4, r6, lsr #18
1:  cmp r0, r6
    add r3, r3, #1 << 20
    strls   r3, [r0], #4
    bls 1b

#ifdef CONFIG_XIP_KERNEL
    /*
     * Map some ram to cover our .data and .bss areas.
     */
    orr r3, r7, #(KERNEL_RAM_PADDR & 0xff000000)
    .if (KERNEL_RAM_PADDR & 0x00f00000)
    orr r3, r3, #(KERNEL_RAM_PADDR & 0x00f00000)
    .endif
    add r0, r4,  #(KERNEL_RAM_VADDR & 0xff000000) >> 18
    str r3, [r0, #(KERNEL_RAM_VADDR & 0x00f00000) >> 18]!
    ldr r6, =(_end - 1)
    add r0, r0, #4
    add r6, r4, r6, lsr #18
1:  cmp r0, r6
    add r3, r3, #1 << 20
    strls   r3, [r0], #4
    bls 1b
#endif

    /*
     * Then map first 1MB of ram in case it contains our boot params.
     */
    add r0, r4, #PAGE_OFFSET >> 18
    orr r6, r7, #(PHYS_OFFSET & 0xff000000)
    .if (PHYS_OFFSET & 0x00f00000)
    orr r6, r6, #(PHYS_OFFSET & 0x00f00000)
    .endif
    str r6, [r0]

#if defined(CONFIG_DEBUG_LL) || defined(CONFIG_DEBUG_SPINLOCK)
    ldr r7, [r10, #PROCINFO_IO_MMUFLAGS] @ io_mmuflags
    /*
     * Map in IO space for serial debugging.
     * This allows debug messages to be output
     * via a serial console before paging_init.
     */
    ldr r3, [r8, #MACHINFO_PGOFFIO]
    add r0, r4, r3
    rsb r3, r3, #0x4000         @ PTRS_PER_PGD*sizeof(long)
    cmp r3, #0x0800         @ limit to 512MB
    movhi   r3, #0x0800
    add r6, r0, r3
    ldr r3, [r8, #MACHINFO_PHYSIO]
    orr r3, r3, r7
1:  str r3, [r0], #4
    add r3, r3, #1 << 20
    teq r0, r6
    bne 1b
#if defined(CONFIG_ARCH_NETWINDER) || defined(CONFIG_ARCH_CATS)
    /*
     * If we're using the NetWinder or CATS, we also need to map
     * in the 16550-type serial port for the debug messages
     */
    add r0, r4, #0xff000000 >> 18
    orr r3, r7, #0x7c000000
    str r3, [r0]
#endif
#ifdef CONFIG_ARCH_RPC
    /*
     * Map in screen at 0x02000000 & SCREEN2_BASE
     * Similar reasons here - for debug.  This is
     * only for Acorn RiscPC architectures.
     */
    add r0, r4, #0x02000000 >> 18
    orr r3, r7, #0x02000000
    str r3, [r0]
    add r0, r4, #0xd8000000 >> 18
    str r3, [r0]
#endif
```

```
#endif
    mov pc, lr
ENDPROC(__create_page_tables)
    .ltorg

#include "head-common.S"
```