```
/*
 *      Basic kernel operations, with apologies to the Intel Corp.
 *
 *      Copyright (c) Division Ltd. 1991
 *
 *      Author Jeff Sullivan
 *
 *      SCCS ident: %W% %G%
 */
#include <sys/trap.h>
#include <sys/psl.h>
#include <sys/reg.h>

#define DB_ALIGN        .align  32
#define NOP6    nop;nop;nop;nop;nop;nop;nop

// special zero - page locations, expressed as signed quantities for
// the sake of the portland assembler, which must hold addresses as
// unsigned quantities, and so decides that ffffff80 is not within +-
// 32k of 0.!!


        SV_PSR = -128
        SV_FIR = -124
        SV_R1 =  -120
        SV_ISP = -116
        SV_KSP = -112
        SV_DB =  -108
        SV_TSP = -104
        SV_TSP1 = -100

        RS_PSR = -96
        RS_FIR = -92
        RS_R1  = -88
        RS_ISP  = -84

//      SV_PSR = 0xfffff000
//      SV_FIR = 0xfffff004
//      SV_R1 =  0xfffff008
//      SV_ISP = 0xfffff00c
//      SV_KSP = 0xfffff010
//      SV_DB =  0xfffff014
//      SV_TSP = 0xfffff018
//      SV_TSP1 = 0xfffff01c

//      RS_PSR = 0xfffff020
//      RS_FIR = 0xfffff024
//      RS_R1  = 0xfffff028
//      RS_ISP  = 0xfffff02c


        TLOCK = -80
        TRACE = -76

//
//      trap handler
//
        .globl  alltraps
alltraps:
// branch here from 0xffffff00

// save a few registers and the pc (without causing more traps)
// only place we have to store stuff without using a register is
// offset(r0) e.g. first or last 32k
        st.l    r1,SV_R1(r0)
        ld.c    db,r1
        st.l    r1,SV_DB(r0)
        st.c    r0,db
        ld.c    fir,r1
        st.l    r1,SV_FIR(r0)
        ld.c    psr,r1          //r1 = PSR
        st.l    r1,SV_PSR(r0)
        st.l    sp,SV_ISP(r0)

//
// Ensure all trap bits clear to allow indirect branches
//
        st.c    r0,psr



//
// CHRG DEBUG    we are always in Supervisor mode
//
// switch stacks if necessary
// e.g. were in User mode
//      and     PSR_PU,r1,r0
//      bc      waskern
//      ld.l    SV_KSP(r0),sp
waskern:
//      nop                     //fix chip timing bug
//      andnot  0xf,sp,sp       //re-align sp to quad boundary

//      ld.l    SV_TSP(r0),r1
//      st.l    r1,SV_TSP1(r0)
//      ld.l    SV_PSR(r0),r1
//      st.l    sp,SV_TSP(r0)

// now that we have a valid stack, save more state
        adds    -256,sp,sp
        adds    -64,sp,sp
        st.l    r16,0(sp)
        st.l    r17,4(sp)
        st.l    r18,8(sp)
        st.l    r19,12(sp)
        st.l    r20,16(sp)
        st.l    r21,20(sp)
        st.l    r22,24(sp)
        st.l    r23,28(sp)
        st.l    r24,32(sp)
        st.l    r25,36(sp)
        st.l    r26,40(sp)
        st.l    r27,44(sp)
        st.l    r28,48(sp)
        st.l    r29,52(sp)
        st.l    r30,56(sp)
        st.l    r31,60(sp)

        adds    -48,sp,sp
        st.l    r4,0(sp)
        st.l    r5,4(sp)
        st.l    r6,8(sp)
        st.l    r7,12(sp)
        st.l    r8,16(sp)
        st.l    r9,20(sp)
        st.l    r10,24(sp)
        st.l    r11,28(sp)
        st.l    r12,32(sp)
        st.l    r13,36(sp)
```

```
        st.l    r14,40(sp)
        st.l    r15,44(sp)

        adds    -16,sp,sp
        st.l    r0,0(sp)
        ld.l    SV_R1(r0),r16
        st.l    r16,4(sp)
        ld.l    SV_ISP(r0),r16
        st.l    r16,8(sp)
        st.l    fp,12(sp)



//
// figure out trap reason (put in r20)
//
        ld.l    SV_FIR(r0),r16  // r16 contains fir

// External Interrupt ?

        and     PSR_IN,r1,r0
        bc      no_intr
        or      T_INTRPT,r0,r20
        br      got_ttype
        nop
no_intr:

// Instruction Access Fault ?

        and     PSR_IAT,r1,r0
        bc      no_iat
        or      T_INSFLT,r0,r20
        br      got_ttype
        nop
no_iat:

// Instruction Trap ?

        and     PSR_IT,r1,r0
        bc      no_it
// was an IT, but what kind?
        ld.l    0(r16),r21      //this cant fault, already checked for IAT
        xorh    0x4400,r21,r0   // trap r0,r0,r0
        bnc     not_bpt
        or      T_BPTFLT,r0,r20
        br      got_ttype
        nop
not_bpt:
        andh    0xffff,r21,r31
        xorh    0x47e0,r31,r0   // trap r31,r31,r0
        bnc     not_sysc
        or      T_SYSCALL,r0,r20
        br      got_ttype
        nop
not_sysc:
        or      T_PRIVINFLT,r0,r20
        br      got_ttype
        nop
no_it:

// Floating Point Trap ?

        and     PSR_FT,r1,r0
        bc      no_ft
        or      T_ARITHTRAP,r0,r20
        br      got_ttype
        nop
no_ft:

// Data Access Trap ?

        and     PSR_DAT,r1,r0
        bc      no_tbits        //should never happen, except for RESET
        or      T_PAGEFLT,r0,r20


// figure out virtual address (r18)
//              r/w             (r19)
//              align required  (r21)

//
//      in dual mode must look at fir +4!!
//
        and     PSR_DIM,r1,r0
        bc      not_di_mode
        addu    4,r16,r16

not_di_mode:
        ld.l    0(r16),r17      // instruction (this cant trap since no IAT)

        andh    0x2000,r17,r0
        bnc     not_ld_st
//double check must be 000x.0x or 000x.11 (dont trust fir)
        andh    0xe800,r17,r21
        bc      ls_ok
        andh    0xec00,r17,r21
        xorh    0x0c00,r21,r0
        bc      ls_ok
        or      T_COMPATFLT,r0,r20
        br      got_ttype
        nop
ls_ok:
// ld or st
        mov     r0,r21
        andh    0x1000,r17,r0
        bc      got_ls_aln
        or      1,r21,r21
        and     1,r17,r0
        bc      got_ls_aln
        or      2,r21,r21
got_ls_aln:
        andh    0x0800,r17,r0
        bc      is_ld
// st
        or      1,r0,r19
// st EA
// const+src2 where const = dest|low11 sign extended
        shr     19,r17,r22
        and     0x7c,r22,r22
        ld.l    r22(sp),r22     //src2

        shr     5,r17,r23
        and     0xf800,r23,r23
        and     0x07ff,r17,r24
        or      r23,r24,r23
        shl     16,r23,r23
        shra    16,r23,r23
        and     1,r21,r24
        nop
        andnot  r24,r23,r23     // adjusted const
```

```
        br      got_datp
        adds    r23,r22,r18     // EA

// ld
is_ld:
        mov     r0,r19
// ld EA
        and     1,r21,r25
        mov     r0,r26

// at this point, we have:
//      r17 - the instruction
//      r25 - low order bits to throw away if const(reg)
//      r26 - bit 0 set if we need auto-increment fixup
//      r19,r21 already set
gen_ea:
        shr     19,r17,r27
        and     0x7c,r27,r27    //index of src2 * 4
        ld.l    r27(sp),r22     //src2

        andh    0x0400,r17,r0
        bc      gen_r_r
// const src1
        shl     16,r17,r23
        shra    16,r23,r23

        nop
        andnot  r25,r23,r23     // toss some low bits

        br      ai_fix
        nop

// reg src1
gen_r_r:
        shr     9,r17,r23
        and     0x7c,r23,r23
        ld.l    r23(sp),r23     // src1

//value of src2 is in r22
//value of src1 is in r23
//index of src2 * 4 is in r27
//flag for autoinc fix in r26
ai_fix:
        and     1,r26,r0
        bc      no_ai_fix
        subs    r22,r23,r22
        adds    r27,sp,r26
        st.l    r22,0(r26)

no_ai_fix:
        br      got_datp
        adds    r23,r22,r18

not_ld_st:
//dont trust chip!
//must be one of:
//      fld,fst 0010.xx
//      pst     0011.11
        andh    0xf000,r17,r21
        xorh    0x2000,r21,r0
        bc      fdat_ok
        andh    0xfc00,r17,r21
        xorh    0x3f00,r21,r0
        bc      fdat_ok
        or      T_COMPATFLT,r0,r20
        br      got_ttype
        nop
fdat_ok:
        and     1,r17,r26       //need autoinc fix?
        or      3,r0,r21
        and     2,r17,r0
        bnc     got_f_aln
        or      4,r21,r21
        and     4,r17,r0
        bc      got_f_aln
        or      8,r21,r21
got_f_aln:
        and     7,r21,r25
        andh    0x1000,r17,r0
        bnc     is_pix
        andh    0x0800,r17,r0
        bc      is_fld
// fst
        or      1,r0,r19
        br      gen_ea
        nop
// fld or pfld
is_fld:
        mov     r0,r19
        br      gen_ea
        nop
// pst
is_pix:
        or      1,r0,r19
        br      gen_ea
        nop

got_datp:
        br      got_ttype
        nop

//
// No Trap bits set so Must be Error or RESET
//
no_tbits:
        or      T_RESET,r0,r20



//
// OK WE HAVE THE TRAP TYPE
//


got_ttype:
        adds    -16,sp,sp
        st.l    r1,0(sp)        //PSR
        ld.l    SV_FIR(r0),r16
        st.l    r16,4(sp)       //FIR
        st.l    r20,8(sp)       //trap type
        ld.l    SV_DB(r0),r16
        st.l    r16,12(sp)      //DB

// clear trap bits in PSR, re-enable interrupts

//      andnot  0x1f00,r1,r30
//      st.c    r30,psr
```

```
// call trap()

        mov     sp,r16
        call    _trap

// call flush_inv()  to flush Data Cache and invalidate Instruction Cache

        nop
        call    _flush_inv
        nop




// restore state
        .globl  ret_user
ret_user:
        ld.c    psr,r1          //turn off interrupts for now
        nop
        andnot  PSR_IM,r1,r1
        st.c    r1,psr


        ld.l    0(sp),r1        //resume PSR
        nop
        andnot  PSR_IM,r1,r1    //the bri will set PSR_IM
        st.l    r1,RS_PSR(r0)
        ld.l    4(sp),r16
        st.l    r16,RS_FIR(r0)  //resume PC
        ld.l    12(sp),r16
        st.c    r16,db          //resume DB

        adds    16,sp,sp

        ld.l    4(sp),r16
        st.l    r16,RS_R1(r0)   //resume R1
        ld.l    8(sp),r16
        st.l    r16,RS_ISP(r0)  //resume SP
        ld.l    12(sp),fp       //resume FP
        adds    16,sp,sp


        ld.l    0(sp),r4
        ld.l    4(sp),r5
        ld.l    8(sp),r6
        ld.l    12(sp),r7
        ld.l    16(sp),r8
        ld.l    20(sp),r9
        ld.l    24(sp),r10
        ld.l    28(sp),r11
        ld.l    32(sp),r12
        ld.l    36(sp),r13
        ld.l    40(sp),r14
        ld.l    44(sp),r15
        adds    48,sp,sp

        ld.l    0(sp),r16
        ld.l    4(sp),r17
        ld.l    8(sp),r18
        ld.l    12(sp),r19
        ld.l    16(sp),r20
        ld.l    20(sp),r21
        ld.l    24(sp),r22
        ld.l    28(sp),r23
        ld.l    32(sp),r24
        ld.l    36(sp),r25
        ld.l    40(sp),r26
        ld.l    44(sp),r27
        ld.l    48(sp),r28
        ld.l    52(sp),r29
        ld.l    56(sp),r30
        ld.l    60(sp),r31
        adds    64,sp,sp


        adds    256,sp,sp

        or      PSR_IT,r1,r1
        st.c    r1,psr          //set return PSR (with trap bit set)

//
// We are always in Supervisor Mode - So no need to restore SP
//

// NBBB BUGGER   There seems to be a problem with this load immediately after
//               the store to the same page. This results in the sp being
//               set to an invalid value.

//      st.l    sp,SV_KSP(r0)   //save kernel sp
//      nop                             // ESSENTIAL FOR READ FIX
//      nop                             // ESSENTIAL FOR READ FIX
//      ld.l    RS_ISP(r0),sp   // return SP


        ld.l    RS_FIR(r0),r1   // return PC
        bri     r1
        ld.l    RS_R1(r0),r1

        .globl  _vstart
        .align  4
_vstart:

//
// Enable Address Translation

//      orh     h%_kpde,r0,r16
//      or      l%_kpde,r16,r16
        orh     0xffff,r0, r16
        or      0xE000,r16,r16

//
//      routine to turn on virtual memory
// (almost same as above -- may merge later)

        ld.c    dirbase,r17
        and     0xfff,r17,r17
        nop
        andnot  0xfff,r16,r16
        or      DIR_ITI|DIR_ATE,r16,r16
        or      r17,r16,r16

        mov     0xfffff000, r18
        or      0x33, r0, r19
        st.l    r19,4(r18)



        DB_ALIGN
```

```
        st.c    r16,dirbase
        NOP6
        orh     h%virt1, r0, r1
        or      l%virt1, r1, r1
        mov     0xfffff000, r18
        or      0x34, r0, r19
        st.l    r19,8(r18)


        bri     r1
        nop
virt1:

        mov     0xfffff000, r18
        or      0x35, r0, r19
        st.l    r19,12(r18)

        call    _flush_inv
        nop
        orh     h%_sbss,r0,r16  // start address of bss in r16
        or      l%_sbss,r16,r16
        mov     r0,r17          // zero character in r17
        orh     h%_zzzend,r0,r18
        or      l%_zzzend,r18,r18
        subs    r18,r16,r18     // length of bss in r18
        call    _memset
        nop
        mov     r0,r16
        mov     r0,r17
        mov     r0,r18


// enable interrupts
//
        call   __main
        nop
        call    _ienable
        nop

//
// Setup stdio
//
        call    __start_buffered_io
        nop

//
// set the secret dos variable
//

#ifdef TARGET_DOS
        or      1,r0,r31                // initialize DOS flag
        orh     ha%___PC_DOS,r0,r19
        st.l    r31,l%___PC_DOS(r19)
#endif


//
// Set up argc, argv, envp
//

        adds -8, sp, sp         //make room for argc, argv
//      call __break_cmd_line   // get command line from iserver
        adds r0, sp, r16        // pass pointer to mem on stack
        ld.l 0(sp), r16         //argc
        ld.l 4(sp), r17         //argv
        adds r0, r0, r18        //envp
        adds 8, sp, sp          //clear back stack

//
// Now Call MAIN -- i.e Application
//

        call    _main
        nop
        call    _exit           // main's return value is in r16 already
        nop


//      flush()
//              flush data cache

        .globl  _flush
_flush:
        mov     0xfffff000, r18
        or      0x06, r0, r19
        st.l    r19, 4(r18)

//      ld.c    psr,r22
//      nop

        mov     0xfffff000, r18
        or      0x09, r0, r19
        st.l    r19, 8(r18)

        or      0x03, r0, r19
        st.l    r19, 12(r18)


//      ld.c    dirbase,r17
//        nop

        mov     0xfffff000, r18
        or      0x07, r0, r19
        st.l    r19, 16(r18)

        adds    -1,r0,r19
        or      127,r0,r20
        or      l%_bad,r0,r16
        orh     h%_bad,r16,r16  //pick an unused virtual address
        adds    -32,r16,r16
        mov     r16,r21

        mov     0xfffff000, r18
        or      0x08, r0, r23
        st.l    r23, 12(r18)

        mov     r17,r18
        nop
        andnot  0xf00,r18,r18
        or      0x800,r18,r18
        mov      r18, r23
        st.c    r18,dirbase     //RC = 2, RB = 0

        bla     r19,r20,flp1
        nop
flp1:
        ld.l    32(r16),r0
        st.l    r1,32(r16)
        bla     r19,r20,flp1
        flush   32(r16)++
```

```
        mov     0xffffff00, r3
        or      0x09, r0, r2
        st.l    r2, 16(r3)


        or      0x100,r23,r18
        st.c    r18,dirbase     //RC = 2, RB = 1

        or      127,r0,r20
        mov     r21,r16
        adds    4096,r16,r16

        bla     r19,r20,flp2
        nop
flp2:
        ld.l    32(r16),r0
        st.l    r1,32(r16)
        bla     r19,r20,flp2
        flush   32(r16)++

        or      0x200,r23,r18
        st.c    r18,dirbase     //RC = 2, RB = 1

        or      127,r0,r20
        mov     r21,r16
        adds    4096,r16,r16

        bla     r19,r20,flp3
        nop
flp3:
        ld.l    32(r16),r0
        st.l    r1,32(r16)
        bla     r19,r20,flp3
        flush   32(r16)++


        or      0x300,r23,r18
        st.c    r18,dirbase     //RC = 2, RB = 1

        or      127,r0,r20
        mov     r21,r16
        adds    4096,r16,r16

        bla     r19,r20,flp4
        nop
flp4:
        ld.l    32(r16),r0
        st.l    r1,32(r16)
        bla     r19,r20,flp4
        flush   32(r16)++

        mov     0xffffff00, r3
        or      0x0a, r0, r2
        st.l    r2, 16(r3)


        st.c    r17,dirbase
        NOP6
        st.c    r22,psr
        bri     r1
        nop

//
//      flush data cache and invalidate tlb and instr cache
        .globl  _flush_inv
_flush_inv:
        adds    -4,sp,sp
        st.l    r1,0(sp)
        call    _flush
        nop
        ld.l    0(sp),r1
        adds    4,sp,sp
//
//      intentional fall through
//
//      invalidate tlb and instr cache
        .globl  _inv_tlb_ic
_inv_tlb_ic:
        ld.c    dirbase,r16
        or      DIR_ITI,r16,r16
        DB_ALIGN
        st.c    r16,dirbase
        NOP6
        DB_ALIGN
        bri     r1
        nop


        .globl  _ienable
_ienable:
        ld.c    psr,r17
        or      PSR_IM,r17,r17
        st.c    r17,psr
        bri     r1
        nop
```