```
/*****************************************************************************
*    File : giet.s
*    Author : Franck Wajsburt & Alain Greiner & Joel Porquet
*    Date : 2009 - 2010
*****************************************************************************
*    Interruption/Exception/Trap Handler for MIPS32 processor
*    The base address of the segment containing this code
*    MUST be 0x80000000, in order to have the entry point
*    at address 0x80000180 !!!
*    All messages are printed on the TTY defined by the processor ID.
*****************************************************************************
*    History :
*    15/09/2009 : The GIET entry point has been modified to comply with
*        the MIPS32 specification : 0x80000180
*    5/10/2009  : The syscall handler has been modified to comply with the
*        MIPS32 specification : the value stored in EPC register is the
*        syscall instruction address => it must be incremented by 4
*        to obtain the return address.
*    15/10/2009 : The Interrupt handler has been modified to comply with the
*        VCI_ICU specification : The IRQ index is obtained by a read
*        to (icu_base_address + 16).
*    26/10/2009 : The interrupt handler has been modified to support
*        multi-processors architectures with one ICU per processor.
*        Finally, the mfc0 instruction uses now the select parameter
*        to comply with the MIPS32 specification when accessing the
*        processor_id (mtfc0 $x, $15, 1)
*    08/11/2009 : The syscall handler has been extended to support 32 values
*        for the syscall index, and to enable interrupts when processing
*        a system call.
*        Five new syscalls have been introduced to access the frame buffer
*        Three new syscalls have been introduced to access the block device
*        The two syscalls associated to the DMA have been removed.
*    18/11/2009 : The syscall handler has been modified to save the SR in
*        the stack before enabling interrupts.
*    15/03/2010 : replace the itoa_print assembler function by the itoa_hex()
*        function defined in the syscalls.c file.
*    10/04/2010 : modify the interrupt handler to use the new ICU component
*        supporting up to 8 output IRQs for 8 processors.
*        The active IRQ index is obtained as ICU[32*PROC_ID+16].
*    12/09/2010 : The ctx_switch functionhas been included in this file to
*        simplify the compilation process.
*   25/09/2010 : add '.end' directive to end the _giet function
*       and modify the 'sharp' comment character into the regular "slash
*       asterix ... asterix slash" comment syntax
*   27/09/2010 : respect stack convention with 4 minimum slots before calling
*       C functions
*   28/09/2010 : Save all the non-persistant registers in the int_handler
*****************************************************************************/
```



```
    .section .giet,"ax",@progbits
    .align 2
    .global _interrupt_vector    /* makes interrupt_vector an external symbol */

    .extern seg_icu_base
    .extern seg_tty_base

    .extern isr_default

    .extern _procid
    .extern _proctime
    .extern _tty_write
    .extern _tty_read
    .extern _tty_read_irq
    .extern _timer_write
    .extern _timer_read
    .extern _icu_write
    .extern _icu_read
    .extern _gcd_write
    .extern _gcd_read
    .extern _locks_read
    .extern _locks_write
    .extern _exit
    .extern _fb_sync_write
    .extern _fb_sync_read
    .extern _fb_write
    .extern _fb_read
    .extern _fb_completed
    .extern _ioc_write
    .extern _ioc_read
    .extern _ioc_completed
    .extern _itoa_hex

    .ent _giet

/*************************************************************
*    Cause Table (indexed by the Cause register)
*************************************************************/
tab_causes:
    .word _int_handler  /* 0000 : external interrupt */
    .word _cause_ukn    /* 0001 : undefined exception */
    .word _cause_ukn    /* 0010 : undefined exception */
    .word _cause_ukn    /* 0011 : undefined exception */
    .word _cause_adel   /* 0100 : illegal address read exception */
    .word _cause_ades   /* 0101 : illegal address write exception */
    .word _cause_ibe    /* 0110 : instruction bus error exception */
    .word _cause_dbe    /* 0111 : data bus error exception */
    .word _sys_handler  /* 1000 : system call */
    .word _cause_bp     /* 1001 : breakpoint exception */
```

```
    .word _cause_ri     /* 1010 : illegal codop exception */
    .word _cause_cpu    /* 1011 : illegal coprocessor access */
    .word _cause_ovf    /* 1100 : arithmetic overflow exception */
    .word _cause_ukn    /* 1101 : undefined exception */
    .word _cause_ukn    /* 1110 : undefined exception */
    .word _cause_ukn    /* 1111 : undefined exception */

    .space 320

/************************************************************
*    Entry point (at address 0x80000180)
************************************************************/
_giet:
    mfc0    $27,    $13             /* Cause Register analysis */
    lui     $26,    0x8000          /* $26 <= tab_causes */
    andi    $27,    $27,    0x3c
    addu    $26,    $26,    $27
    lw      $26,    ($26)
    jr      $26                     /* Jump indexed by CR */
    .end _giet

/************************************************************
*   System Call Handler
* A system call is handled as a special function call.
*   - $2 contains the system call index (< 16).
*   - $3 is used to store the syscall address
*   - $4, $5, $6, $7 contain the arguments values.
*   - The return address (EPC) iand the SR are saved in the stack.
*   - Interrupts are enabled before branching to the syscall.
*   - All syscalls must return to the syscall handler.
*   - $2, $3, $4, $5, $6, $7 as well as $26 & $27 can be modified.
*
* In case of undefined system call, an error message displays
* the value of EPC on the TTY corresponding to the processor,
* and the user program is killed.
************************************************************/
_sys_handler:
    addiu   $29,    $29,    -24     /* 2 slots for SR&EPC, 4 slots for args passing */
    mfc0    $26,    $12             /* load SR */
    sw      $26,    16($29)         /* save it in the stack */
    mfc0    $27,    $14             /* load EPC */
    addiu   $27,    $27,    4       /* increment EPC for return address */
    sw      $27,    20($29)         /* save it in the stack */

    andi    $26,    $2,     0x1F    /* $26 <= syscall index (i < 32) */
    sll     $26,    $26,    2       /* $26 <= index * 4 */
    la      $27,    tab_syscalls    /* $27 <= &tab_syscalls[0] */
    addu    $27,    $27,    $26     /* $27 <= &tab_syscalls[i] */
    lw      $3,     0($27)          /* $3  <= syscall address */
```



```
    li      $27,    0xFFFFFFED      /* Mask for UM & EXL bits */
    mfc0    $26,    $12             /* $26 <= SR */
    and     $26,    $26,    $27     /* UM = 0 / EXL = 0 */
    mtc0    $26,    $12             /* interrupt enabled */
    jalr    $3                      /* jump to the proper syscall */
    mtc0    $0,     $12             /* interrupt disbled */

    lw      $26,    16($29)         /* load SR from stack */
    mtc0    $26,    $12             /* restore SR */
    lw      $26,    20($29)         /* load EPC from stack */
    mtc0    $26,    $14             /* restore EPC */
    addiu   $29,    $29,     24     /* restore stack pointer */
    eret                            /* exit GIET */

_sys_ukn:                           /* undefined system call */
    la      $4,     msg_uknsyscall  /* $4 <= message address */
    li      $5,    36               /* $5 <= message length */
    jal     _tty_write              /* print unknown message */

    la      $4,     msg_epc         /* $4 <= message address */
    li      $5,    8                /* $5 <= message length */
    jal     _tty_write              /* print EPC message */

    mfc0    $4,     $14             /* $4 <= EPC */
    la      $5,     itoa_buffer     /* $5 <= buffer address */
    addiu   $5,     $5,     2       /* skip the 0x prefix */
    jal     _itoa_hex               /* fill the buffer */

    la      $4,     itoa_buffer     /* $4 <= buffer address */
    li      $5,     10              /* $5 <= buffer length */
    jal     _tty_write              /* print EPC value */

    j       _exit                   /* end of program */

itoa_buffer: .ascii "0x00000000"

    .align 2

/************************************************************
* System Call Table (indexed by syscall index) */
tab_syscalls:
    .word _procid           /* 0x00 */
    .word _proctime         /* 0x01 */
    .word _tty_write        /* 0x02 */
    .word _tty_read         /* 0x03 */
    .word _timer_write      /* 0x04 */
    .word _timer_read       /* 0x05 */
    .word _gcd_write        /* 0x06 */
```

```
    .word _gcd_read         /* 0x07 */
    .word _icu_write        /* 0x08 */
    .word _icu_read         /* 0x09 */
    .word _tty_read_irq     /* 0x0A */
    .word _sys_ukn          /* 0x0B */
    .word _locks_write      /* 0x0C */
    .word _locks_read       /* 0x0D */
    .word _exit             /* 0x0E */
    .word _sys_ukn          /* 0x0F */
    .word _fb_sync_write    /* 0x10 */
    .word _fb_sync_read     /* 0x11 */
    .word _fb_write         /* 0x12 */
    .word _fb_read          /* 0x13 */
    .word _fb_completed     /* 0x14 */
    .word _ioc_write        /* 0x15 */
    .word _ioc_read         /* 0x16 */
    .word _ioc_completed    /* 0x17 */
    .word _sys_ukn          /* 0x18 */
    .word _sys_ukn          /* 0x19 */
    .word _sys_ukn          /* 0x1A */
    .word _sys_ukn          /* 0x1B */
    .word _sys_ukn          /* 0x1C */
    .word _sys_ukn          /* 0x1D */
    .word _sys_ukn          /* 0x1E */
    .word _sys_ukn          /* 0x1F */

/*****************************************************************
*    Interrupt Handler
* This simple interrupt handler cannot be interrupted.
* It uses an external ICU component (Interrupt Controler Unit)
* that concentrates up to 32 interrupts lines to a single IRQ
* line that can be connected to any of the 6 MIPS IT inputs.
* This component returns the highest priority active interrupt index
* (smaller indexes have the highest priority).
*
* In case of a multi-ptocessor architecture, there is one ICU
* per processor. The base address of the ICU segment is
* computed as : seg_icu_base + 0x00100000 * proc_id
*
* The interrupt handler reads the ICU_IT_VECTOR register,
* using the offset  16.
* This component returns the highest priority interrupt index
* (smaller indexes have the highest priority).
* Any value larger than 31 means "no active interrupt", and
* the default ISR (that does nothing) is executed.
* The interrupt vector (32 ISR addresses array stored at
* _interrupt_vector address) is initialised with the default
* ISR address. The actual ISR addresses are supposed to be written
* in the interrupt vector array by the boot code.
```



```
* All non persistant registers, such as $1 to $15, and $24 to $25,
* as well as register $31 and EPC, are saved in the interrupted
* program stack, before calling the
* Interrupt Service Routine, and can be used by the ISR code.
*****************************************************************/
_int_handler:
    addiu   $29,    $29,    -23*4   /* stack space reservation */
    .set noat
    sw      $1,     4*4($29)        /* save $1 */
    .set at
    sw      $2,     4*5($29)        /* save $2 */
    sw      $3,     4*6($29)        /* save $3 */
    sw      $4,     4*7($29)        /* save $4 */
    sw      $5,     4*8($29)        /* save $5 */
    sw      $6,     4*9($29)        /* save $6 */
    sw      $7,     4*10($29)       /* save $7 */
    sw      $8,     4*11($29)       /* save $8 */
    sw      $9,     4*12($29)       /* save $9 */
    sw      $10,    4*13($29)       /* save $10 */
    sw      $11,    4*14($29)       /* save $11 */
    sw      $12,    4*15($29)       /* save $12 */
    sw      $13,    4*16($29)       /* save $13 */
    sw      $14,    4*17($29)       /* save $14 */
    sw      $15,    4*18($29)       /* save $15 */
    sw      $24,    4*19($29)       /* save $24 */
    sw      $25,    4*20($29)       /* save $25 */
    sw      $31,    4*21($29)       /* save $31 */
    mfc0    $27,    $14
    sw      $27,    4*22($29)       /* save EPC */
    mfc0    $27,    $15,    1       /* $27 <= proc_id */
    andi    $27,    $27,    0x7     /* at most 8 processors */
    sll     $27,    $27,    20      /* $27 <= proc_id * 0x100000 */
    la      $26,    seg_icu_base    /* $26 <= seg_icu_base */
    add     $26,    $26,    $27     /* $26 <= seg_icu_base + 0x100000 * proc_id */
    lw      $26,    16($26)         /* $26 <= interrupt_index */
    srl     $27,    $26,    5
    bne     $27,    $0,     restore /* do nothing if index > 31 */
    sll     $26,    $26,    2       /* $26 <= interrupt_index * 4 */
    la      $27,    _interrupt_vector
    addu    $26,    $26,    $27
    lw      $26,    ($26)           /* read ISR address */
    jalr    $26                     /* call ISR */
restore:
    .set noat
    lw      $1,     4*4($29)        /* restore $1 */
    .set at
    lw      $2,     4*5($29)        /* restore $2 */
    lw      $3,     4*6($29)        /* restore $3 */
    lw      $4,     4*7($29)        /* restore $4 */
```

```
    lw      $5,     4*8($29)        /* restore $5 */
    lw      $6,     4*9($29)        /* restore $6 */
    lw      $7,     4*10($29)       /* restore $7 */
    lw      $8,     4*11($29)       /* restore $8 */
    lw      $9,     4*12($29)       /* restore $9 */
    lw      $10,    4*13($29)       /* restore $10 */
    lw      $11,    4*14($29)       /* restore $11 */
    lw      $12,    4*15($29)       /* restore $12 */
    lw      $13,    4*16($29)       /* restore $13 */
    lw      $14,    4*17($29)       /* restore $14 */
    lw      $15,    4*18($29)       /* restore $15 */
    lw      $24,    4*19($29)       /* restore $24 */
    lw      $25,    4*20($29)       /* restore $25 */
    lw      $31,    4*21($29)       /* restore $31 */
    lw      $27,    4*22($29)       /* return address (EPC) */
    addiu   $29,    $29,    23*4    /* restore stack pointer */
    mtc0    $27,    $14             /* restore EPC */
    eret                            /* exit GIET */

/* The default ISR is called when no specific ISR has been installed in the
* interrupt vector. It simply displays a message on TTY 0. */

isr_default:
    addiu   $29,    $29,    -20     /* get space in stack */
    sw      $31,    16($29)         /* to save the return address */
    la      $4,     msg_default     /* $4 <= string address */
    addi    $5,     $0,     36      /* $5 <= string length */
    jal     _tty_write              /* print */
    lw      $31,    16($29)         /* restore return address */
    addiu   $29,    $29,    20      /* free space */
    jr      $31                     /* returns to interrupt handler */

/* Interrupt Vector Table (indexed by interrupt index)
* 32 words corresponding to 32 ISR addresses */
_interrupt_vector:
    .word isr_default   /* ISR 0 */
    .word isr_default   /* ISR 1 */
    .word isr_default   /* ISR 2 */
    .word isr_default   /* ISR 3 */
    .word isr_default   /* ISR 4 */
    .word isr_default   /* ISR 5 */
    .word isr_default   /* ISR 6 */
    .word isr_default   /* ISR 7 */
    .word isr_default   /* ISR 8 */
    .word isr_default   /* ISR 9 */
    .word isr_default   /* ISR 10 */
    .word isr_default   /* ISR 11 */
    .word isr_default   /* ISR 12 */
    .word isr_default   /* ISR 13 */
```



```
    .word isr_default   /* ISR 14 */
    .word isr_default   /* ISR 15 */
    .word isr_default   /* ISR 16 */
    .word isr_default   /* ISR 17 */
    .word isr_default   /* ISR 18 */
    .word isr_default   /* ISR 19 */
    .word isr_default   /* ISR 20 */
    .word isr_default   /* ISR 21 */
    .word isr_default   /* ISR 22 */
    .word isr_default   /* ISR 23 */
    .word isr_default   /* ISR 24 */
    .word isr_default   /* ISR 25 */
    .word isr_default   /* ISR 26 */
    .word isr_default   /* ISR 27 */
    .word isr_default   /* ISR 28 */
    .word isr_default   /* ISR 29 */
    .word isr_default   /* ISR 30 */
    .word isr_default   /* ISR 31 */

/***********************************************************
*    Exception Handler
* Same code for all fatal exceptions :
* Print the exception type and the values of EPC & BAR
* on the TTY correspondintg to the processor PROCID,
* and the user program is killed.
***********************************************************/
_cause_bp:
_cause_ukn:
_cause_ri:
_cause_ovf:
_cause_adel:
_cause_ades:
_cause_ibe:
_cause_dbe:
_cause_cpu:
    mfc0    $26,    $13             /* $26 <= CR */
    andi    $26,    $26,    0x3C    /* $26 <= _cause_index * 4 */
    la      $27,    mess_causes     /* mess_cause table base address */
    addu    $27,    $26,    $27     /* pointer on the message base address */

    lw      $4,     ($27)           /* $4 <= message address */
    li      $5,     36              /* $5 <= message length */
    jal     _tty_write              /* print message cause */

    la      $4,     msg_epc         /* $4 <= message address */
    li      $5,     8               /* $5 <= message length */
    jal     _tty_write              /* print message EPC */

    mfc0    $4,     $14             /* $4 <= EPC value */
```

```
    la      $5,     itoa_buffer     /* $5 <= buffer address */
    addiu   $5,     $5,     2       /* skip 0x prefix */
    jal     _itoa_hex               /* fill buffer */

    la      $4,     itoa_buffer     /* $4 <= buffer address */
    li      $5,     10              /* $5 <= buffer length */
    jal     _tty_write              /* print EPC value */

    la      $4,     msg_bar         /* $4 <= mesage address */
    li      $5,     8               /* $5 <= message length */
    jal     _tty_write              /* print message BAR */

    mfc0    $4,     $8              /* $4 <= BAR value */
    la      $5,     itoa_buffer     /* $5 <= buffer address */
    addiu   $5,     $5,     2       /* skip 0x prefix */
    jal     _itoa_hex               /* fill buffer */

    la      $4,     itoa_buffer     /* $4 <= mesage address */
    li      $5,     10              /* $5 <= message length */
    jal     _tty_write              /* print BAR value */

    j       _exit                   /* end program */

/* Exceptions Messages table (indexed by CAUSE) */
mess_causes:
    .word msg_ukncause
    .word msg_ukncause
    .word msg_ukncause
    .word msg_ukncause
    .word msg_adel
    .word msg_ades
    .word msg_ibe
    .word msg_dbe
    .word msg_ukncause
    .word msg_bp
    .word msg_ri
    .word msg_cpu
    .word msg_ovf
    .word msg_ukncause
    .word msg_ukncause
    .word msg_ukncause

/*****************************************************************
*    All messages
* Messages length are fixed : 8 or 36 characters...
*****************************************************************/
msg_bar:        .asciiz "\nBAR  = "
msg_epc:        .asciiz "\nEPC  = "
msg_default:    .asciiz "\n\n  !!! Default ISR  !!!          \n"
```



```
msg_uknsyscall: .asciiz "\n\n  !!! Undefined System Call !!!  \n"
msg_ukncause:   .asciiz "\n\nException : strange unknown cause\n"
msg_adel:       .asciiz "\n\nException : illegal read address \n"
msg_ades:       .asciiz "\n\nException : illegal write address\n"
msg_ibe:        .asciiz "\n\nException : inst bus error       \n"
msg_dbe:        .asciiz "\n\nException : data bus error       \n"
msg_bp:         .asciiz "\n\nException : breakpoint           \n"
msg_ri:         .asciiz "\n\nException : reserved instruction \n"
msg_ovf:        .asciiz "\n\nException : arithmetic overflow  \n"
msg_cpu:        .asciiz "\n\nException : illegal coproc access\n"
    .align 2


/*****************************************************************************
*       _ctx_switch
*   The _ctx_switch function performs a context switch between the
*   current task and another task.
*   It can be used in a multi-processor architecture, with the assumption
*   that the tasks are statically allocated to processors.
*   The max number of processorsi is 8, and the max number of tasks is 4.
*   The scheduling policy is very simple : For each processor, the task index
*   is incremented,  modulo the number of tasks allocated to the processor.
*
*   It has no argument, and no return value.
*
*   It uses three global variables:
*    - _current_task_array :  an array of 8 task index:
*      (index of the task actually running on each processor)
*    - _task_number_array : an array of 8 numbers:
*      (the number of tasks allocated to each processor)
*       - _task_context_array : an array of 32 task contexts:
*      (at most 8 processors / each processor can run up to 4 tasks)
*   A task context is an array of 64 words = 256 bytes.
*   It is indexed by m = (proc_id*4 + task_id)
*   It contains copies of the processor registers.
*   As much as possible a register is stored at the index defined by its number
*   ( for example, $8 is saved in ctx[8]).
*   The exception are :
*   $0 is not saved since always 0
*   $26, $27 are not saved since not used by the task
*
*   0*4(ctx) SR    8*4(ctx) $8    16*4(ctx) $16   24*4(ctx) $24   32*4(ctx) EPC
*   1*4(ctx) $1    9*4(ctx) $9    17*4(ctx) $17   25*4(ctx) $25   33*4(ctx) CR
*   2*4(ctx) $2   10*4(ctx) $10   18*4(ctx) $18   26*4(ctx) LO    34*4(ctx) reserved
*   3*4(ctx) $3   11*4(ctx) $11   19*4(ctx) $19   27*4(ctx) HI    35*4(ctx) reserved
*   4*4(ctx) $4   12*4(ctx) $12   20*4(ctx) $20   28*4(ctx) $28   36*4(ctx) reserved
*   5*4(ctx) $5   13*4(ctx) $13   21*4(ctx) $21   29*4(ctx) $29   37*4(ctx) reserved
*   6*4(ctx) $6   14*4(ctx) $14   22*4(ctx) $22   30*4(ctx) $30   38*4(ctx) reserved
*   7*4(ctx) $7   15*4(ctx) $15   23*4(ctx) $23   31*4(ctx) $31   39*4(ctx) reserved
```

```
*
*   The return address contained in $31 is saved in the _current task context
*   (in the ctx[31] slot), and the function actually returns to the address contained
*   in the ctx[31] slot of the new task context.
*
*   Caution : This function is intended to be used with periodic interrupts.
*   It can be directly called by the OS, but interrupts must be disabled before calling.
*******************************************************************************/

    .section .ksave

    .global _task_context_array /* initialised in reset.s */
    .global _current_task_array /* initialised in reset.s */
    .global _task_number_array  /* initialised in reset.s */

_task_context_array:    /* 32 contexts : indexed by (proc_id*4 + task_id) */
    .space  8192

_current_task_array:    /* 8 words : indexed by the proc_id */
    .word   0           /* _current_task_array[0] <= 0 */
    .word   0           /* _current_task_array[1] <= 0 */
    .word   0           /* _current_task_array[2] <= 0 */
    .word   0           /* _current_task_array[3] <= 0 */
    .word   0           /* _current_task_array[4] <= 0 */
    .word   0           /* _current_task_array[5] <= 0 */
    .word   0           /* _current_task_array[6] <= 0 */
    .word   0           /* _current_task_array[7] <= 0 */

_task_number_array:     /* 8 words : indexed by the proc_id */
    .word   1           /* _task_number_array[0] <= 1 */
    .word   1           /* _task_number_array[1] <= 1 */
    .word   1           /* _task_number_array[2] <= 1 */
    .word   1           /* _task_number_array[3] <= 1 */
    .word   1           /* _task_number_array[4] <= 1 */
    .word   1           /* _task_number_array[5] <= 1 */
    .word   1           /* _task_number_array[6] <= 1 */
    .word   1           /* _task_number_array[7] <= 1 */

/*******************************************************************************/

    .section .switch

    .global _ctx_switch /* makes it an external symbol */
    .align  2

_ctx_switch:

    /* test if more than one task on the processor */
```



```
    mfc0    $26,    $15,    1
    andi    $26,    $26,    0x7         /* $26 <= proc_id */
    sll     $26,    $26,    2           /* $26 <= 4*proc_id */
    la      $27,    _task_number_array  /* $27 <= base address of _task_number_array */
    addu    $27,    $27,    $26         /* $27 <= _task_number_array + 4*proc_id */
    lw      $27,    ($27)               /* $27 <= task number */
    addi    $26,    $27,    -1          /* $26 <= _task_number - 1 */
    bnez    $26,    do_it               /* 0 if only one task */
    jr      $31                         /* return */
do_it:
    /* save _current task context */

    mfc0    $26,    $15,    1
    andi    $26,    $26,    0x7         /* $26 <= proc_id */
    sll     $26,    $26,    2           /* $26 <= 4*proc_id */
    la      $27,    _current_task_array /* $27 <= base address of _current_task_array */
    addu    $27,    $27,    $26         /* $27 <= _current_task_array + 4*proc_id */
    lw      $26,    ($27)               /* $26 <= current task index */
    sll     $26,    $26,    8           /* $26 <= 256*task_id */
    la      $27,    _task_context_array /* $27 <= base address of context array */
    addu    $27,    $27,    $26         /* $27 <= _task_context_array + 256*task_id */
    mfc0    $26,    $15,    1
    andi    $26,    $26,    0x7         /* $26 <= proc_id */
    sll     $26,    $26,    10          /* $26 <= 1024*proc_id */
    addu    $27,    $27,    $26         /* $27 <= taxk_context_array + 256*(proc_id*4 + task_i

    mfc0    $26,    $12         /* $26 <= SR */
    sw      $26,    0*4($27)    /* ctx[0] <= SR */
    .set noat
    sw      $1,     1*4($27)    /* ctx[1] <= $1 */
    .set at
    sw      $2,     2*4($27)    /* ctx[2] <= $2 */
    sw      $3,     3*4($27)    /* ctx[3] <= $3 */
    sw      $4,     4*4($27)    /* ctx[4] <= $4 */
    sw      $5,     5*4($27)    /* ctx[5] <= $5 */
    sw      $6,     6*4($27)    /* ctx[6] <= $6 */
    sw      $7,     7*4($27)    /* ctx[7] <= $7 */
    sw      $8,     8*4($27)    /* ctx[8] <= $8 */
    sw      $9,     9*4($27)    /* ctx[9] <= $9 */
    sw      $10,    10*4($27)   /* ctx[10] <= $10 */
    sw      $11,    11*4($27)   /* ctx[11] <= $11 */
    sw      $12,    12*4($27)   /* ctx[12] <= $12 */
    sw      $13,    13*4($27)   /* ctx[13] <= $13 */
    sw      $14,    14*4($27)   /* ctx[14] <= $14 */
    sw      $15,    15*4($27)   /* ctx[15] <= $15 */
    sw      $16,    16*4($27)   /* ctx[16] <= $16 */
    sw      $17,    17*4($27)   /* ctx[17] <= $17 */
    sw      $18,    18*4($27)   /* ctx[18] <= $18 */
    sw      $19,    19*4($27)   /* ctx[19] <= $19 */
```

```
    sw      $20,    20*4($27)   /* ctx[20] <= $20 */
    sw      $21,    21*4($27)   /* ctx[21] <= $21 */
    sw      $22,    22*4($27)   /* ctx[22] <= $22 */
    sw      $23,    23*4($27)   /* ctx[23] <= $23 */
    sw      $24,    24*4($27)   /* ctx[24] <= $24 */
    sw      $25,    25*4($27)   /* ctx[25] <= $25 */
    mflo    $26
    sw      $26,    26*4($27)   /* ctx[26] <= LO */
    mfhi    $26
    sw      $26,    27*4($27)   /* ctx[27] <= H1 */
    sw      $28,    28*4($27)   /* ctx[28] <= $28 */
    sw      $29,    29*4($27)   /* ctx[29] <= $29 */
    sw      $30,    30*4($27)   /* ctx[30] <= $30 */
    sw      $31,    31*4($27)   /* ctx[31] <= $31 */
    mfc0    $26,    $14
    sw      $26,    32*4($27)   /* ctx[32] <= EPC */
    mfc0    $26,    $13
    sw      $26,    33*4($27)   /* ctx[33] <= CR */

    /* select  the new task */

    mfc0    $15,    $15,    1
    andi    $15,    $15,    0x7         /* $15 <= proc_id */
    sll     $16,    $15,    2           /* $16 <= 4*proc_id */
    la      $17,    _current_task_array /* $17 <= base address of _current_task_array */
    addu    $17,    $17,    $16         /* $17 <= _current_task_array + 4*proc_id */
    lw      $18,    ($17)               /* $18 <= _current task index */
    la      $19,    _task_number_array  /* $19 <= base address of _task_number_array */
    addu    $19,    $19,    $16         /* $19 <= _task_number_array + 4*proc_id */
    lw      $20,    ($19)               /* $20 <= max = number of tasks */
    addiu   $18,    $18,    1           /* $18 <= new task index */
    sub     $2,     $18,    $20         /* test modulo max */
    bne     $2,     $0,     no_wrap
    add     $18,    $0,     $0          /* $18 <= new task index */
no_wrap:
    sw      $18,    ($17)               /* update _current_task_array */

    /* restore next task context */

    sll     $19,    $18,    8           /* $19 <= 256*task_id */
    la      $27,    _task_context_array /* $27 <= base address of context array */
    addu    $27,    $27,    $19         /* $27 <= _task_context_array + 256*task_id */
    sll     $19,    $15,    10          /* $19 <= 1024*proc_id */
    addu    $27,    $27,    $19         /* $27 <= _task_context_array + 256*(proc_id*4 + task_

    lw      $26,    0*4($27)
    mtc0    $26,    $12                 /* restore SR */
    .set noat
    lw      $1,     1*4($27)            /* restore $1 */
```



```
    .set at
    lw      $2,     2*4($27)            /* restore $2 */
    lw      $3,     3*4($27)            /* restore $3 */
    lw      $4,     4*4($27)            /* restore $4 */
    lw      $5,     5*4($27)            /* restore $5 */
    lw      $6,     6*4($27)            /* restore $6 */
    lw      $7,     7*4($27)            /* restore $7 */
    lw      $8,     8*4($27)            /* restore $8 */
    lw      $9,     9*4($27)            /* restore $9 */
    lw      $10,    10*4($27)           /* restore $10 */
    lw      $11,    11*4($27)           /* restore $11 */
    lw      $12,    12*4($27)           /* restore $12 */
    lw      $13,    13*4($27)           /* restore $13 */
    lw      $14,    14*4($27)           /* restore $14 */
    lw      $15,    15*4($27)           /* restore $15 */
    lw      $16,    16*4($27)           /* restore $16 */
    lw      $17,    17*4($27)           /* restore $17 */
    lw      $18,    18*4($27)           /* restore $18 */
    lw      $19,    19*4($27)           /* restore $19 */
    lw      $20,    20*4($27)           /* restore $20 */
    lw      $21,    21*4($27)           /* restore $21 */
    lw      $22,    22*4($27)           /* restore $22 */
    lw      $23,    23*4($27)           /* restore $23 */
    lw      $24,    24*4($27)           /* restore $24 */
    lw      $25,    25*4($27)           /* restore $25 */
    lw      $26,    26*4($27)
    mtlo    $26                         /* restore LO */
    lw      $26,    27*4($27)
    mthi    $26                         /* restore HI */
    lw      $28,    28*4($27)           /* restore $28 */
    lw      $29,    29*4($27)           /* restore $29 */
    lw      $30,    30*4($27)           /* restore $30 */
    lw      $31,    31*4($27)           /* restore $31 */
    lw      $26,    32*4($27)
    mtc0    $26,    $14                 /* restore EPC */
    lw      $26,    33*4($27)
    mtc0    $26,    $13                 /* restore CR */

    jr      $31                         /* returns to caller */

/* Local Variables:
   tab-width: 4;
   c-basic-offset: 4;
   c-file-offsets:((innamespace . 0)(inline-open . 0));
   indent-tabs-mode: nil;
   End: */

/* vim: set filetype=asm expandtab shiftwidth=4 tabstop=4 softtabstop=4: */
```