```
# SPIM S20 MIPS simulator.
# The default exception handler for spim.
#
# Copyright (C) 1990-2004 James Larus, larus@cs.wisc.edu.
# ALL RIGHTS RESERVED.
#
# SPIM is distributed under the following conditions:
#
# You may make copies of SPIM for your own use and modify those copies.
#
# All copies of SPIM must retain my name and copyright notice.
#
# You may not sell SPIM or distributed SPIM in conjunction with a commerical
# product or service without the expressed written consent of James Larus.
#
# THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
# IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
# WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
# PURPOSE.
#

######################################################################
# NOTE: Comments added and expanded by Neal Wagner, April 4, 1999
#       and by Matthew Patitz, October 21, 2008
#   ("Text" below refers to Patterson and Hennessy, _Computer
#    Organization and Design_, Morgan Kaufmann.)
#
# INTERRUPT HANDLING IN MIPS:
# Coprocessor 0 has extra registers useful in handling exceptions
# There are four useful coprocessor 0 registers:
#-----------------------------------------------------------------|
#  REG NAME | NUMBER |   USAGE                                    |
#-----------------------------------------------------------------|
#  BadVAddr |   8    | Memory addr at which addr exception occurred |
#  Status   |  12    | Interrupt mask and enable bits               |
#  Cause    |  13    | Exception type and pending interrupt bits    |
#  EPC      |  14    | Address of instruction that caused exception |
#----------|--------|----------------------------------------------|
# Details:
#   Status register: has an interrupt mask with a bit for each of
#      five interrupt levels.  If a bit is one, interrupts at that
#      level are allowed.  If a bit is zero, interrupts at that level
#      are disabled.  The low order 6 bits of the Status register
#      implement a three-level stack for the "kernel/user" and
#      "interrupt enable" bits.  The "kernel/user" bit is 0 if the
#      program was running in the kernel when the interrupt occurred
#      and 1 if it was in user mode.  If the "interrupt enable" bit is 1,
#      interrupts are allowed.  If it is 0, they are disabled.  At an
#      interrupt, these six bits are shifted left by two bits.
#   Cause register: The value in bits 2-5 of the Cause register
#      describes the particular type of exception.  The error messages
#      below describe these values.  Thus a 7 in bits 2-5 corresponds
#      to message __e7_ below, or a "bad address in data/stack read".
#
# There are special machine instructions for accessing these
# coprocessor 0 registers:
#      mfc0  Rdest, CPsrc: "move from coprocessor 0" moves data
#         from the special coprocessor 0 register CPsrc into the
#         general purpose register Rdest.
#      mtc0  Rsrc, CPdest: "move to coprocessor 0" moves data
#         from the general purpose register Rsrc into the special
#         coprocessor 0 register CPdest.
# (There are also coprocessor load and store instructions.)
#
# ACTIONS BY THE TRAP HANDLER CODE BELOW:
#  Branch to address 0x80000180 and execute handler there:
```

```
#  1. Save $a0 and $v0 in s0 and s1 and $at in $k1.
#  2. Move Cause into register $k0.
#  3. Do action such as print an error message.
#  4. Increment EPC value so offending instruction is skipped after
#     return from exception.
#  5. Restore $a0, $v0, and $at.
#  6. Clear the Cause register and re-enable interrupts in the Status
#     register.
#  6. Execute "eret" instruction to return execution to the instruction
#     at EPC.
#
##############################################################################

#
    .kdata
__m1_:  .asciiz "  Exception "
__m2_:  .asciiz " occurred and ignored\n"
__e0_:  .asciiz "  [Interrupt] "
__e1_:  .asciiz "  [TLB]"
__e2_:  .asciiz "  [TLB]"
__e3_:  .asciiz "  [TLB]"
__e4_:  .asciiz "  [Address error in inst/data fetch] "
__e5_:  .asciiz "  [Address error in store] "
__e6_:  .asciiz "  [Bad instruction address] "
__e7_:  .asciiz "  [Bad data address] "
__e8_:  .asciiz "  [Error in syscall] "
__e9_:  .asciiz "  [Breakpoint] "
__e10_: .asciiz "  [Reserved instruction] "
__e11_: .asciiz ""
__e12_: .asciiz "  [Arithmetic overflow] "
__e13_: .asciiz "  [Trap] "
__e14_: .asciiz ""
__e15_: .asciiz "  [Floating point] "
__e16_: .asciiz ""
__e17_: .asciiz ""
__e18_: .asciiz "  [Coproc 2]"
__e19_: .asciiz ""
__e20_: .asciiz ""
__e21_: .asciiz ""
__e22_: .asciiz "  [MDMX]"
__e23_: .asciiz "  [Watch]"
__e24_: .asciiz "  [Machine check]"
__e25_: .asciiz ""
__e26_: .asciiz ""
__e27_: .asciiz ""
__e28_: .asciiz ""
__e29_: .asciiz ""
__e30_: .asciiz "  [Cache]"
__e31_: .asciiz ""
__excp: .word __e0_, __e1_, __e2_, __e3_, __e4_, __e5_, __e6_, __e7_, __e8_, __e9_
    .word __e10_, __e11_, __e12_, __e13_, __e14_, __e15_, __e16_, __e17_, __e18_,
    .word __e19_, __e20_, __e21_, __e22_, __e23_, __e24_, __e25_, __e26_, __e27_,
    .word __e28_, __e29_, __e30_, __e31_
s1: .word 0
s2: .word 0

#####################################################
# This is the exception handler code that the processor runs when
# an exception occurs. It only prints some information about the
# exception, but can serve as a model of how to write a handler.
#
# Because we are running in the kernel, we can use $k0/$k1 without
# saving their old values.

# This is the exception vector address for MIPS32:
    .ktext 0x80000180
```

```
#####################################################
# Save $at, $v0, and $a0
#

    move $k1 $at            # Save $at

    sw $v0 s1               # Not re-entrant and we can't trust $sp
    sw $a0 s2               # But we need to use these registers


#####################################################
# Print information about exception
#
    li $v0 4                # syscall 4 (print_str)
    la $a0 __m1_
    syscall

    li $v0 1                # syscall 1 (print_int)
    mfc0 $k0 $13            # Get Cause register
    srl $a0 $k0 2           # Extract ExcCode Field
    andi $a0 $a0 0xf
    syscall

    li $v0 4                # syscall 4 (print_str)
    andi $a0 $k0 0x3c
    lw $a0 __excp($a0)      # $a0 has the index into
                            # the __excp array (exception
                            # number * 4)
    nop
    syscall

#####################################################
# Bad PC exception requires special checks
#
    bne $k0 0x18 ok_pc
    nop

    mfc0 $a0 $14            # EPC
    andi $a0 $a0 0x3        # Is EPC word-aligned?
    beq $a0 0 ok_pc
    nop

    li $v0 10               # Exit on really bad PC
    syscall

#####################################################
#  PC is alright to continue
#
ok_pc:

    li $v0 4                # syscall 4 (print_str)
    la $a0 __m2_            # "occurred and ignored" message
    syscall

    srl $a0 $k0 2           # Extract ExcCode Field
    andi $a0 $a0 0xf
    bne $a0 0 ret           # 0 means exception was an interrupt
    nop

#####################################################
# Interrupt-specific code goes here!
# Don't skip instruction at EPC since it has not executed.
#  -> not implemented
#
```

```
#####################################################
# Return from (non-interrupt) exception. Skip offending
# instruction at EPC to avoid infinite loop.
#
ret:

    mfc0 $k0 $14            # Get EPC register value
    addiu $k0 $k0 4         # Skip faulting instruction by skipping
                            # forward by one instruction
                          # (Need to handle delayed branch case here)
    mtc0 $k0 $14            # Reset the EPC register

regresar:
#####################################################
# Restore registers and reset procesor state
#
    lw $v0 s1               # Restore $v0 and $a0
    lw $a0 s2

    .set noat
    move $at $k1            # Restore $at
    .set at

    mtc0 $0 $13             # Clear Cause register

    mfc0 $k0 $12            # Set Status register
    ori  $k0 0x1            # Interrupts enabled
    mtc0 $k0 $12


#####################################################
# Return from exception on MIPS32
#
    eret

# End of exception handling
#####################################################



#
```