# File: /usr/local/spim//exceptions.s
# For use with the new version of ... __e14_, __e15_, __e16_,          __e17_, __e18_,        .word __e19_, __e20_, __e21_, __e22_, __e23_, __e24_, __e25_,
# 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 express 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.

# $Header: $

# Define the exception handling code. This must go first!

.kdata
__m1__: .asciz "Exception"
__m2__: .asciz "occurred and ignored"
__e0__: .asciz "[Interrupt]"
__e1__: .asciz "[TLB]"
__e2__: .asciz "[TLB]"
__e3__: .asciz "[Address error in instr/data fetch]"
__e4__: .asciz "[Address error in store]"
__e5__: .asciz "[Bad instruction address]"
__e6__: .asciz "[Bad data address]"
__e7__: .asciz "[Error in syscall]"
__e9__: .asciz "[Breakpoint]"
__e10__: .asciz "[Reserved instruction]"
__e11__: .asciz "[Arithmetic overflow]"
__e13__: .asciz "[Trap]"
__e14__: .asciz "[Bad pc]
__e15__: .asciz "[Floating point]"
__e16__: .asciz "[Coproc 2]"
__e17__: .asciz "[Cp
__e18__: .asciz "[MDMX]"
__e23__: .asciz "[Watch]"
__e24__: .asciz "[Machine check]"
__e25__: .asciz "[Cache]"
__e26__: .asciz "[Cache]"
__e30__: .asciz "[Cache]"
__e31__: .asciz "[Cache]"
__e26__: .asciz "[Cache]"
__e27__: .asciz "[Cache]"
__e28__: .asciz "[Cache]"
__e29__: .asciz "[Cache]"
__e30__: .asciz "[Cache]"
__e31__: .asciz "[Cache]"

_exceptions.s

# This is the exception handler code that the processor runs when
# an exception occurs. It only prints some information about the
# exception, but can server 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 MIPS-1 (R2000):
# This is the exception vector address for MIPS32:
# Select the appropriate one for the mode in which SPIM is compiled.
.net note
move $k1 $at # Save $at
.set at
sw $v0 4l # Not re-entrant and we can't trust $sp
sw $a0 $2 # But, we need to use these registers
mfc0 $k0 $13 # Cause register
sr1 $a0 $k0 2 # Extract ExcCode Field
andi $a0 $a0 0x0f
# Print information about exception.
li $v0 4 # syscall 4 (print_str)
la $a0 __m1_
 syscall
li $v0 1 # syscall 1 (print_int)
srl $a0 $k0 2 # Extract ExcCode Field
andi $a0 $a0 0xf
syscall
li $v0 4 # syscall 4 (print_str)
andi $a0 $k0 0x3c
lw $a0 __exc($a0)
nop
syscall
bne $k0 $x18 ok_pc # Bad PC exception requires special checks	nop
mfc0 $a0 $14 # EPC
andi $a0 $a0 0x3
beq $a0 0 ok_pc	nop
li $v0 10 # Exit on really bad PC
syscall
ok_pc:
li $v0 4 # syscall 4 (print_str)
la $a0 __m2_
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.
ret:
# Return from (non−interrupt) exception. Skip offending instruction
# at EPC to avoid infinite loop.
  # mfc0 $k0 $14        # Bump EPC register
addiu $k0 $k0 4        # Skip faulting instruction
mtc0 $k0 $14        # (Need to handle delayed branch case here)

# Restore registers and reset processor state
#
.set noat
move $at $kl        # Restore $at

.set at
lw $v0 a1        # Restore other registers
lw $a0 a2
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

# Return sequence for MIPS−I (R2000):
#  rfe        # Return from exception handler
#  jr $k0        # Should be in jr’s delay slot
#  nop

# Standard startup code. Invoke the routine "main" with arguments:
# main(argc, argv, envp)
#
.text
.globl __start
__start:
  lw $a0 0($sp)        # argc
addiu $a1 $sp 4        # argv
addiu $a2 $a1 4        # envp
sll $v0 $a0 2
addu $a2 $a2 $v0
jal main
nop
  li $v0 10
syscall        # syscall 10 (exit)
.globl __eoth
__eoth: