Virtual Memory/MMU

- Processor can be running in real mode or virtual mode
  - In real mode, the MMU does nothing, virtual addresses are identical to real addresses
  - In virtual mode, the MMU translates all memory addresses from the virtual addresses into physical addresses.
- Parts of the OS runs in real mode, user programs can run in virtual mode.
MMU

CPU

Address

MemData

AddrError

Comp

Length

Base

mode

MEM

Address

Data
Interrupt Styles

• Status Interrupt
  – Cause of interrupt is stored in a register

• Vectored Interrupt
  – Cause of interrupt is implicitly defined by the location jumped to when the interrupt occurs – the interrupt vector
MIPS Interrupt/Exception Implementation

- Access to Interrupt/Exception informations implemented as a co-processor
- All accessible information is stored in co-processor registers
- Special instructions
  - `mfc0 $localreg, $cpreg`  # move from co-processor
  - `mtc0 $localreg, $cpreg`  # move to co-processor
  - `eret`  # return from exception
## Coprocessor 0 Registers

<table>
<thead>
<tr>
<th>Register name</th>
<th>Register number</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BadVAddr</td>
<td>8</td>
<td>memory address at which an offending memory reference occurred</td>
</tr>
<tr>
<td>Count</td>
<td>9</td>
<td>timer</td>
</tr>
<tr>
<td>Compare</td>
<td>11</td>
<td>value compared against timer that causes interrupt when they match</td>
</tr>
<tr>
<td>Status</td>
<td>12</td>
<td>interrupt mask and enable bits</td>
</tr>
<tr>
<td>Cause</td>
<td>13</td>
<td>exception type and pending interrupt bits</td>
</tr>
<tr>
<td>EPC</td>
<td>14</td>
<td>address of instruction that caused exception</td>
</tr>
<tr>
<td>Config</td>
<td>16</td>
<td>configuration of machine</td>
</tr>
</tbody>
</table>
Status and Cause Reg Fields

FIGURE A.7.1 The Status register.

FIGURE A.7.2 The Cause register.
## Cause Flags in SPIM

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Cause of exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Int</td>
<td>interrupt (hardware)</td>
</tr>
<tr>
<td>4</td>
<td>AdEL</td>
<td>address error exception (load or instruction fetch)</td>
</tr>
<tr>
<td>5</td>
<td>AdES</td>
<td>address error exception (store)</td>
</tr>
<tr>
<td>6</td>
<td>IBE</td>
<td>bus error on instruction fetch</td>
</tr>
<tr>
<td>7</td>
<td>DBE</td>
<td>bus error on data load or store</td>
</tr>
<tr>
<td>8</td>
<td>Sys</td>
<td>syscall exception</td>
</tr>
<tr>
<td>9</td>
<td>Bp</td>
<td>breakpoint exception</td>
</tr>
<tr>
<td>10</td>
<td>RI</td>
<td>reserved instruction exception</td>
</tr>
<tr>
<td>11</td>
<td>CpU</td>
<td>coprocessor unimplemented</td>
</tr>
<tr>
<td>12</td>
<td>Ov</td>
<td>arithmetic overflow exception</td>
</tr>
<tr>
<td>13</td>
<td>Tr</td>
<td>trap</td>
</tr>
<tr>
<td>15</td>
<td>FPE</td>
<td>floating point</td>
</tr>
</tbody>
</table>
Simple Exception Handler

.ktext 0x80000180
mov  $k1, $at # Save $at register
sw  $a0, save0  # Handler is not re-entrant and can’t use
sw  $a1, save1  # stack to save $a0, $a1
# Don't need to save $k0/$k1
mfc0  $k0, $13  # Move Cause into $k0
srl  $a0, $k0, 2 # Extract ExcCode field
andi $a0, $a0, 0xf
bgtz  $a0, done  # Branch if ExcCode is Int (0)
mov  $a0, $k0  # Move Cause into $a0
mfc0 $a1, $14  # Move EPC into $a1
jal  print_excp  # Print exception error message
Simple Exception Handler 2

done: mfc0  $k0, $14 # Bump EPC
addiu $k0, $k0, 4 # Do not reexecute
# faulting instruction
mtc0  $k0, $14 # EPC
mtc0  $0, $13 # Clear Cause register
mfc0  $k0, $12 # Fix Status register
andi  $k0, 0xfffd # Clear EXL bit
ori $k0, 0x1 # Enable interrupts
mtc0 $k0, $12
lw  $a0, save0 # Restore registers
lw $a1, save1
mov $at, $k1
eret  # Return to EPC