MIPS Exception Handling

Cptr280

Dr Curtis Nelson

QTSpim Exception Handler

- Up until now, your programs have been running with QTSpim used as a "bare machine" — a computer with no machine code in it but your own. Most computer systems run under the control of an operating system. Application programs use the services of the operating system to do input and output and other system tasks.
- QTSpim does not have an operating system, but it includes a small exception handler that provides a set of services that can be a great help in writing assembly programs.
**Syscall**

- In even the simplest computer, putting a character on the screen involves many instructions and a detailed knowledge of the video card. QTSpim includes an "exception handler" that simulates a tiny operating system that can do input from the keyboard and output to the console window.
- Assembly language programs request operating system services using the syscall instruction. The syscall instruction transfers control to the operating system which then performs the requested service. Then control usually returns to the program. This description leaves out many details.
- The *print services* write characters to the simulated monitor (console window) of QTSpim. The *read services* read characters from the keyboard and convert character strings into the appropriate type.

**Print String**

- The *exit service* is normally used when you want your program to terminate. A real computer is always running, and control has to go somewhere, so this is how control is sent back to the OS. The OS then goes about its own tasks, possibly starting another user program. On QTSpim, the exit service stops all execution.
- The *print string* QTSpim exception handler service, prints a null terminated string to the simulated monitor. The address of the string is loaded into register $a0. Typically the string is in the data section of memory.

```
li $v0,4   # code 4 — print string
la $a0,string  # $a0 — address of the string
syscall    # Ask the operating system to
           # perform the service

.data
string:  .ascii "Hello QTSpim!\n"
```

- The *print string* service sends bytes to the simulated monitor one by one starting with the byte pointed to by $a0. It continues until it hits a null byte. It does not check that the bytes are ASCII (more shortly). You will probably print garbage if you point $a0 to the wrong location.
- If you want to advance to a new line, use the newline character 'n' inside or at the end of the string.
ASCII Table

- ASCII (American Standard Code for Information Interchange) is a standard 8-bit code for representing certain characters and control actions.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\x00</td>
<td>\x01</td>
<td>\x02</td>
<td>\x03</td>
<td>\x04</td>
<td>\x05</td>
<td>\x06</td>
</tr>
<tr>
<td>1</td>
<td>\x07</td>
<td>\x08</td>
<td>\x09</td>
<td>\x0A</td>
<td>\x0B</td>
<td>\x0C</td>
<td>\x0D</td>
</tr>
<tr>
<td>2</td>
<td>\x0E</td>
<td>\x0F</td>
<td>\x10</td>
<td>\x11</td>
<td>\x12</td>
<td>\x13</td>
<td>\x14</td>
</tr>
<tr>
<td>3</td>
<td>\x15</td>
<td>\x16</td>
<td>\x17</td>
<td>\x18</td>
<td>\x19</td>
<td>\x1A</td>
<td>\x1B</td>
</tr>
<tr>
<td>4</td>
<td>\x1C</td>
<td>\x1D</td>
<td>\x1E</td>
<td>\x1F</td>
<td>\x20</td>
<td>\x21</td>
<td>\x22</td>
</tr>
<tr>
<td>5</td>
<td>\x23</td>
<td>\x24</td>
<td>\x25</td>
<td>\x26</td>
<td>\x27</td>
<td>\x28</td>
<td>\x29</td>
</tr>
<tr>
<td>6</td>
<td>\x2A</td>
<td>\x2B</td>
<td>\x2C</td>
<td>\x2D</td>
<td>\x2E</td>
<td>\x2F</td>
<td>\x30</td>
</tr>
</tbody>
</table>

Starting QTSpim

- With QTSpim, exceptions are handled by the simulator. On a real machine an exception handler might be permanently in ROM or might be loaded from a "boot sector" of the hard disk.
- To use the exception handler with QTSpim do the following in the Simulator Settings panel. Set the options "Allow pseudoinstructions" and "Load Exception Handler". Disable "Enable Mapped I/O".
- Now, to assemble and load your program go to File Open and select the source file. Your program is loaded along with some initialization code. The initialization code starts at address 0x00400000, the address that has until now been where your program started.
- The trap handler gets control when an executing program generates a trap. This is commonly done by dividing by zero, loading a word from a non-word-aligned address, and similar errors.
Exceptions

- Normal control flow - sequential, jumps, branches, calls, returns.
- Exception = non-programmed control transfer
  - System takes action to handle the exception
    o Records the address of the offending instruction;
    o Preserves the state of the machine;
    o Processes as necessary.
  - Returns control to user.

Two Types of Exceptions

- Interrupts
  - Caused by external events;
  - Asynchronous to program execution;
  - May be handled between instructions;
  - Simply suspend, preserve state, process as necessary, and then resume user program.
- Traps (or Exceptions)
  - Caused by internal events;
    • Exceptional conditions (overflow);
    • Invalid instruction;
    • Faults (non-resident page in memory);
    • Internal hardware error.
  - Synchronous to program execution;
  - Condition must be remedied by the handler;
  - Instruction may be retried or program may be aborted.
Additions to MIPS ISA to Support Exceptions

- **EPC** – a 32-bit register used to hold the address of the offending instruction.
- **Cause** – a register used to record the cause of the exception. To simplify the discussion, assume
  - undefined instruction = 0
  - arithmetic overflow = 1
- **Status** - interrupt mask and enable bits that determines what exceptions can occur.
- The ability to write the exception address into PC.
- May have to undo PC = PC + 4, since we want EPC to point to offending instruction (not its successor); PC = PC - 4

Big Picture: User/System Modes

- By providing two modes of execution (user/system) it is possible for the computer to somewhat manage itself
  - Operating system is a special program that runs in the privileged mode and has access to all of the resources of the computer.
  - Presents “virtual resources” to each user that are more convenient than the physical resources
    - Files vs. disk sectors;
    - Virtual memory vs. physical memory.
  - Protects each user program from others.
Exception Use in QTSpim

<table>
<thead>
<tr>
<th>Register name</th>
<th>Register number</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackAddr</td>
<td>6</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>

These seven registers are part of coprocessor 0's register set. They are accessed by the mfc0 and npc0 instructions. After an exception, register EPC contains the address of the instruction that was executing when the exception occurred. If the exception was caused by an external interrupt, then the instruction will not have started executing. All other exceptions are caused by the execution of the instruction at EPC, except when the offending instruction is in the delay slot of a branch or jump. In that case, EPC points to the branch or jump instruction and the BD bit is set in the Cause register. When that bit is set, the exception handler must look at EPC + 4 for the offending instruction. However, in either case, an exception handler properly resumes the program by returning to the instruction at EPC.

Exception Use in QTSpim

![Figure A.7.2](image)

**FIGURE A.7.2** The Cause register.

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Cause of exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IFE</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>AES</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>DRE</td>
<td>bus error on data load or store</td>
</tr>
<tr>
<td>8</td>
<td>Syn</td>
<td>special exception</td>
</tr>
<tr>
<td>9</td>
<td>Bp</td>
<td>break-point 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>TPE</td>
<td>floating point</td>
</tr>
</tbody>
</table>
Exception Use in QTSpim

Exceptions and interrupts cause a MIPS processor to jump to a piece of code, at address 80000180$_{hex}$ (in the kernel, not user address space), called an exception handler. This code examines the exception’s cause and jumps to an appropriate point in the operating system. The operating system responds to an exception either by terminating the process that caused the exception or by performing some action. A process that causes an error, such as executing an unimplemented instruction, is killed by the operating system. On the other hand, other exceptions such as page faults are requests from a process to the operating system to perform a service, such as bringing in a page from disk. The operating system

---

```
Exception Use in QTSpim (File exceptions.s)

// QTSpim S20 MIPS simulator.
// The default exception handler for QTSpim.
// Define the exception handling code. This must go first!

.kdata

.m1_: .asciiz " Exception "
.m2_: .asciiz " occurred and ignored"
.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 ""
```
Exception Use in QTSpim (File exceptions.s)

__e18__: .asciiz "[Coproc 2]"
__e19__: .asciiz ""
__e20__: .asciiz ""
__e21__: .asciiz "[MDMX]"
__e22__: .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.*/

Exception Use in QTSpim (File exceptions.s)

/* This is the exception vector address for MIPS-1 (R2000):*/
/*.ktext 0x80000080*/
/* This is the exception vector address for MIPS32:
.ktext 0x80000180*/
/* Select the appropriate one for the mode in which
QTSpim is compiled.*/
.set rodata
move $k1 $at
.set at
sw $v0 s1
sw $a0 s2
mfc0 $k0 $13
sti $t0 $k0 2
andi $t0 $t0 0xcf
/* Print information about exception.
li $v0 4
la $a0 __m1
syscall
li $v0 1
sti $t0 $k0 2
andi $t0 $t0 0xcf
syscall*/
/* ... etc*/
Summary

- Exception handling is a requirement of every computer system;
- Exceptions (or traps) are internal events;
- Interrupts are external events;
- QTSpim provides a simple exception handler.