Computer Systems Lecture 19

Exceptions

An exceptions are exceptional events that interrupt normal program flow and require the attention of the CPU outside of the running program.

External (“interrupts”)

* Not caused by program execution, could be a physical system change like plugging in a peripheral ect..
* E.g. I/O interrupt like a network packet arrived

Internal(“traps”)

* Caused by program execution
* E.g. syscall or TLB miss or ALU overflow ect…

Intentional Exceptions

Use exception mechanism to request some OS functions, eg.g I/O (print to screen), memory allocation ect…

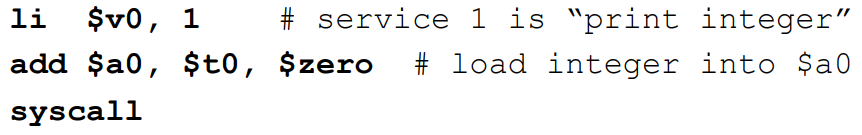
MIPS: user program uses syscall instruction

* The cause register ($v0) is set with a special value to identify the syscall exception
* The OS exception handler is invoked when the syscall instruction executes

Parameters are passed to the OS through agreed upon registers (usually $a0, $a1,…)

Syscall Example

The following will print the integer in register $t0 to the screen



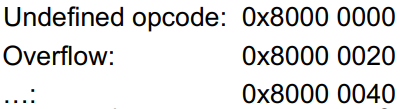
Calling syscall is essentially a function call, there is a lookup table that the operating system essentially goes to when syscall occurs which lists the addresses of all the syscall functions. The operating system uses the table to find the correct address for the desired syscall (based on $v0) and then goes to it.

Exception Mechanism

* Step 1: Save the address of the current instruction
  + This is saved in to a special register called the exception program counter (EPC), note that the system must return to the interrupted instruction (not PC+4) this is because (depending on the exception) we may need to return to the instruction that caused the exception, for example in the case of a TLB miss for a load instruction.
* Step 2: Transfer control to the OS at a known address (i.e. the exception handler PC)
* Step 3: Handle the exception
  + Deal with the cause of the exception
  + All registers must be preserved, similar to a procedure call
* Step 4: Return to user program execution
  + Handler restores user program’s registers and jumps back using EPC
  + Relies on a special instruction: eret (exception return)

Finding the Exception Handler

* Approach 1:
  + Jump to a predefined address
  + Use the Cause register to then branch to the right handler (e.g. print int, read string, exit program ect…)
  + Works well for syscall – cause register is explicitly set
* Approach 2:
  + Directly jump to a specific handler depending on the exception (vectored interrupt)
  + The interrupt signal comes from the “outside” via CPU pins e.g:



Handling the Exception

* Determine action required
  + By inspecting the Cause register or by virtue of being at the right handler (for say an undefined opcode)
* If restartable:
  + Take corrective action, then use EPC to return to program
* Otherwise:
  + Terminate program and report error using EPC, cause, etc…
* For a critical time while the interrupt is being handled, other interrupts should not happen
  + Otherwise the EPC, Cause will be overwritten
  + This is forced by masking interrupts by setting the excpetion level (EXL) bit in the status register