

Interrupts

An *interrupt* (also known as an *exception* or *trap*) is an event that causes the CPU to stop executing the current program and start executing a special piece of code called an *interrupt handler* or *interrupt service routine* (ISR). The ISR typically does some work, then resumes the interrupted program.

- Similar to a procedure call, except that an interrupt:
 1. can occur between any two instructions of the program
 2. is transparent to the running program (usually)
 3. is typically *not* explicitly requested by the program
 4. calls a routine at an address determined by the type of interrupt, not by the program
 5. atomically changes some processor mode bits in the machine status register (MSR)

Interrupt Types

Microprocessors use the interrupt mechanism for lots of largely unrelated things. These things can be grouped into two categories:

1. Synchronous (instruction-related)
 - illegal instruction
 - privileged instruction
 - bus error (“machine check”)
 - divide by 0 (on most processors), floating-point errors
 - virtual memory page fault
 - system call (into operating system)
2. Asynchronous: (not instruction-related)
 - external hardware device
 - timer expiration
 - reset
 - power failure
 - on-chip debugging (on 823)

See Section 6.3.8 of the MPC823 data book.

PowerPC Interrupt Structure

As for procedure calls, the PowerPC architecture provides “bare bones” support for interrupts.

- Two special-purpose registers: save/restore registers 0 and 1 (SRR0 and SRR1)
- One instruction: return from interrupt (rfi)

Basic interrupt process:

1. stop executing current program
2. save PC of next instruction in SRR0
3. save the processor mode bits from MSR in SRR1
4. change some of the processor mode bits in MSR
5. branch to address determined by type of interrupt (ISR)

The last instruction in the ISR will be an rfi, which will:

1. restore the processor mode bits from SRR1 to MSR
2. branch to the address in SRR0

A Simple Example

Nesting Interrupts

An interrupt can happen while executing an ISR. This is called a *nested interrupt*.

- multiple interrupting devices with long-running ISRs
- debugging ISR code
- supporting virtual memory in ISRs

What must a PowerPC ISR do to support nested interrupts?

Disabling Interrupts

Sometimes you can't afford to take an interrupt:

-
- time-critical instruction sequences
- before you've initialized data needed by an ISR
- changing data structures shared with an ISR

Synchronous interrupts can be avoided by not executing instructions that might cause them:

- illegal instructions
- loads or stores to nonexistent addresses
- etc.

Asynchronous interrupts from external devices can be disabled (*masked*) using the “external interrupt enable” (EE) mode bit.

- bit 16 of the machine status register (MSR)
- if 0, external interrupt signal is ignored
- if 1, external interrupt signal causes interrupt when asserted
- EE is atomically set to 0 on *any* interrupt

Note that some interrupts (e.g. reset) are not maskable.