I/O Data Transfer

There are two key questions that determine how data is transferred to and from a non-trivial I/O device. Each question has two possible answers.

1. How does the CPU know when data is available?

   a. Polling

   b. Interrupts

2. How is data transferred into and out of the device?

   a. Programmed I/O

   b. Direct Memory Access (DMA)

So far we have implicitly assumed the simple answers to these questions (polling and programmed I/O). We will look at interrupts next, and DMA later in the term.

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) (see Table 7-1 on p. 7-8 of MPC823 data book)

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.

Managing Multiple Device Interrupts

A system with multiple devices that can interrupt the CPU must be able to determine which device to service on an interrupt. There are two related issues:

1. **Identification**: which device(s) caused the current interrupt?

2. **Prioritization**: if more than one device is simultaneously interrupting, which one gets handled first?

Depending on the system, high-priority interrupts may or may not be able to interrupt low-priority service routines.

Three standard approaches:

1. Non-vectored
2. Vectored
3. Autovectored

Non-vectored Interrupts

- Simplest hardware, least flexible
- Used in 6802, PowerPC, MIPS?
- single CPU interrupt input/vector
- ISR checks (polls) each device to see which may have caused interrupt
- prioritization?
- nesting?

Vectored Interrupts

- Used in 8080, 80x86, Z80
- again, single CPU interrupt input
- CPU has special *interrupt acknowledge* bus cycle to get vector number directly from device
  - like read, but different control signals
  - 8086: $\overline{\text{INTA}}$
- prioritization typically via daisy chain
- typically requires that devices be designed to work with specific CPU
- nesting?

Autovectored Interrupts

- Used in 68000, SPARC
- Can be built on top of vectored (80x86 w/8259A (PC)) or non-vectored interrupts (MPC823)
- multiple CPU interrupt inputs, one for each priority level (e.g., $\overline{\text{IRQ0}}$, $\overline{\text{IRQ1}}$, $\overline{\text{IRQ2}}$, etc.)
- CPU *autovectors* based on highest-priority asserted interrupt
- CPU *interrupt priority level* controls which interrupts get recognized
  - e.g., if IPL = 3 then disable $\overline{\text{IRQ3}}$, $\overline{\text{IRQ4}}$, $\overline{\text{IRQ5}}$, ...
  - on interrupt, CPU atomically raises IPL to match level request being serviced
  - generalization of enable bit

Autovectored Interrupts, cont'd

- directly provides prioritization, nesting
- more devices than priority levels?
- Intel 8259A interrupt controller chip builds autovectoring on top of 80x86 vectoring (IBM PC)
- MPC823 provides on-chip interrupt controller for pseudo-autovectored interrupts

MPC823 Interrupt Controller

Part of on-chip System Interface Unit (SIU), *not* part of PowerPC processor "core".
See Section 12.3 of data book.

- MPC823 provides eight external interrupt inputs, $\overline{\text{IRQ0}}$ (highest priority) through $\overline{\text{IRQ7}}$ (lowest priority).
- Eight additional internal interrupt priorities, Level 0 through Level 7, generated by on-chip devices
- Priorities are interleaved: $\overline{\text{IRQ0}}$, Level 0, $\overline{\text{IRQ1}}$, Level 1, ...
- Assertion of any of these 16 interrupts can potentially assert the PowerPC "external interrupt" signal
- Interrupt Vector Register (SIVEC)
  - indicates highest-priority interrupt (0-16)
  - read-only memory-mapped register gives interrupt * 4
- Interrupt Mask Register (SIMASK)
  - bits to enable/disable each interrupt
- Interrupt Pending Register (SIPEND)
  - bits indicate which interrupts are pending