Lecture 17: Interrupt tools

Interrupts-
External examples: Timers, keyboard, DAQ cards, video cards, serial and parallel ports, digital logic pulses.
Internal examples: (processor generated) illegal code exceptions, task scheduling algorithms.
NMI – nonmaskable interrupts: internal or external; avoid when possible, especially on UARTS or timers and on hard R-T systems. NMI must be re-entrant. Note: don't use NMI with edge triggered interrupt (vulnerable to noise); also ISRs typically cannot be re-entrant during 1st few lines of execution.

Needed interrupt components-
Interrupt vector: connects all interrupts of the system. (INTEL 80x86 contains 256 entries, but most PCs have only 16 interrupts available.) This vector is an array of pointers to the interrupt service routines that will be triggered when the corresponding interrupt occurs. The vector also contains a bit for each interrupt line that signals whether there is an interrupt pending on that line, i.e., a peripheral device has raised the interrupt, and is waiting to be serviced.

non-vectored interrupt processing: an interrupt occurs, control is transferred to one single routine that decides what to do with the interrupt. This approach is also used in operating systems that allow multiple devices to share the same interrupt.

Computer system hardware:
-can automatically prevent other interrupts from being serviced as long as the ISR of the first interrupt is still running,
-can assign a (static) priority to each interrupt, such that the ISR of a lower-priority interrupt is pre-empted when a higher-priority interrupt is triggered,
-can queue interrupts, such that none of them gets lost.

Interrupt controller: external hardware that shields the operating system from the electronic details of the interrupt lines and makes the interrupt vector look like what it should look like in software: a simple vector. It can also provide just one interrupt input to the CPU.

```
      Interrupt   ->    Priority    ->    interrupt   ->    control     ->  CPU
       Register         register           vector            logic

                        mask/status
                         register

Address Bus ____________________________________________________________________
Data Bus    ____________________________________________________________________
Control Bus ____________________________________________________________________

                                                            RAM   ROM   I/O
```

Lecture 14: Interrupt tools (cont)

DMA controller –
H/W solution for efficient transfer of data to/from CPU/memory.
Frees CPU from pulling on external devices that need servicing.
Initiates and controls bus access between external devices and memory.
Does low level buffer filling/emptying for all devices on the data bus.
Controlled by CPU via interrupts and sends messages to CPU via interrupts.
Connected to control/data/address bus.
Address bus keeps tab of the address and counter of the data to be processed.
Control bus signals which device has access to data bus which transports data.
Useful for vertical integration, for example with extra external ROM/RAM.

Interrupt software-
Interrupt Service Routine (ISR): software routine called when an interrupt occurs on the interrupt line for which the ISR has been registered in the interrupt vector. Typically, this registration takes place through a system call to the operating system, but can be done directly by a (sufficiently privileged) program too. In VxWorks, all routines connect to interrupts using intConnect. Note: The operating system cannot intervene in the launching of the ISR, because everything is done by the CPU hardware. The context of the currently running task is saved on the stack of that task: its address is in one of the CPU registers, and it is the only stack that the CPU has immediate and automatic access to. Each task must get enough stack space to cope with ISR overhead; for nested interrupt applications, this space can become very large. And all ISRs use the same interrupt stack segment of memory.

Software interrupt tools of the real-time kernel:
-Interrupt enable/disable: Each processor has atomic operations to enable or disable ("mask") the interrupts.
-Prioritized interrupt disable: technique disables interrupts except those above a specified priority level. If the processor allows interrupt priorities, most opportunities/problems that are known in the task show up in the interrupt handling too.
-Interrupt nesting: If the processor and/or operating system allows interrupt nesting, ISR servicing of one interrupt can itself be pre-empted by another interrupt. Interrupt nesting increases code complexity, because ISRs must use re-entrant code only, i.e. any number of concurrent processes (including itself) can call them at the same time. Hence, ISRs should be written in such a way that they are robust against being pre-empted at any time. Some ISR re-entrant techniques that are used: global and static variables ***that are protected by mutual exclusion semaphores*** and used outside the ISR; private variable copies of global variables; and data that is provided only by function input arguments.

Software interrupts: not caused by a (asynchronous) hardware event, but by a specific (synchronous) machine language operation code, such as, for example a trap (an unexpected software condition or error, such as a divide by zero, or an undefined instruction.) Linux accepts multiple interrupt handlers on the same interrupt number. Software interrupt and trap numbers are defined by the hardware, i.e., they have entries in the interrupt vector. As with a hardware interrupt, a software interrupt or a trap causes the processor to save the current state, and to jump to the ISR routine specified in the interrupt vector; useful for example for debugging – an ISR can be triggered by traps in the code generated by the debug compiler.