COS10004 Computer Systems

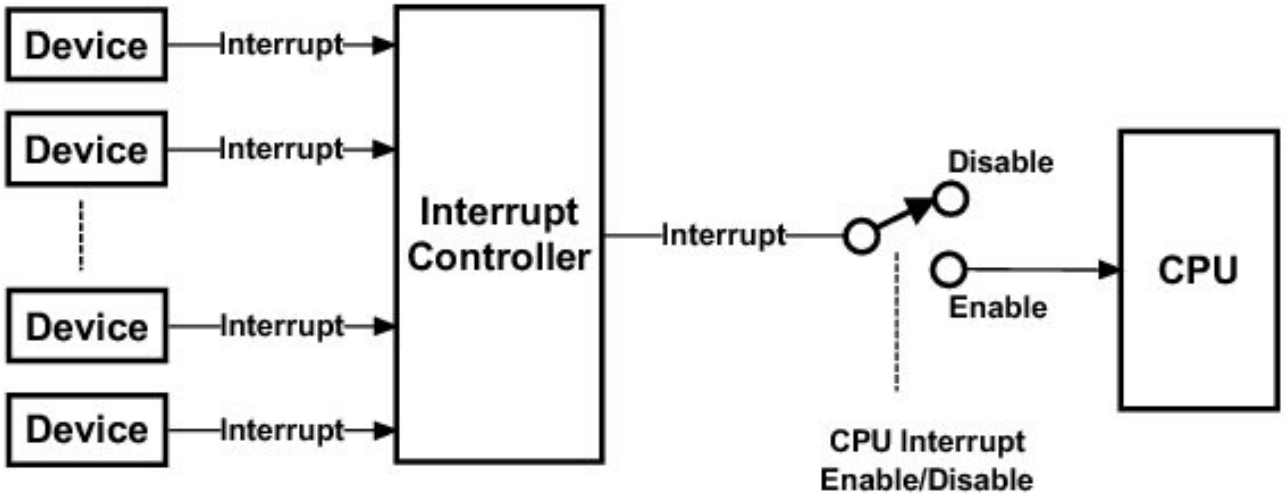
Lecture 4.5: Interrupts and Polling

CRICOS provider 00111D

Dr Chris McCarthy

CPU MULTI-TASKING - INTERRUPTS

- A response to a signal that needs attention from the software



src: https://doc.micrium.com/display/os305/Handling+CPU+Interrupts

INTERRUPTS

Stacks allow interrupt-based hardware access.

- A device (e.g. I/O) issues an electrical signal, which feeds into a priority encoder which then issues an INT signal to the CPU (depending on relative priority).
- The current work of the CPU (including the value of the IP) is ***push***ed onto the stack.
- The CPU loads the Interrupt's handler routine (code which specifies what to do when interrupted by the specific hardware), executes the code.
- The INT Handler ends with a RETurn instruction, which ***pop***s the stored IP off the stack into the IP, and processing resumes.

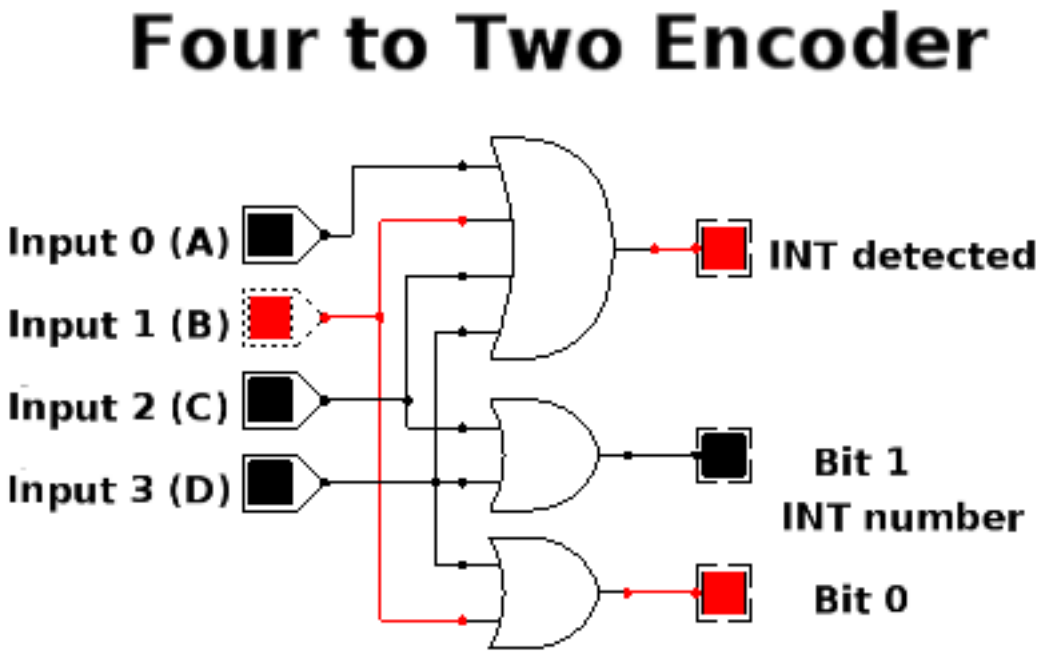
How I/O INTs work

- Each hardware event is mapped to an Interrupt number, highest priority first (lowest).
- Define an Interrupt vector table
  - an array of addresses of (pointers to) functions (int handlers) which execute depending on the interrupt level.
- Set up an Interrupt Mask – enables or disables each interrupt type.
  - e.g. mask INT 0 (CTRL+ALT+DEL) if currently processing an INT 0.
- When an event is detected, control jumps to the table, to the INT handler code.

INTERRUPTS – USE HARDWARE SIGNALS TO RUN CODE

- We need some logic to process these.
- Important interrupts (mouse move, keyboard, power down, Ctrl+Alt+Del, Ctrl+C) need a high priority.
- Unimportant (not-time-critical) things (function calls, GUI update) should have a lower priority.
- We can use an encoder to read the electrical signals from hardware devices (ranked by priority), and to generate a binary INT number used to select the correct interrupt handler code.
  - the INT number will be used as the index of the INT vector table (an array of pointers).

- Here's one that processes 4 inputs (from 4 devices) and reports the INT number in 2-bit binary.



CAN'T INT AN INT

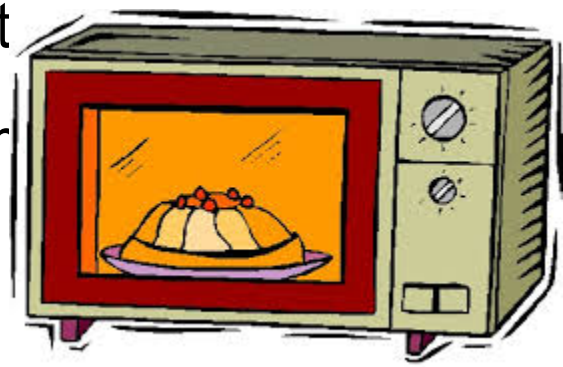
- Any routine can be interrupted, including the code loaded by an INT handler routine if the priority is higher.
- loading an INT handler is ***atomic***
  - it can't be interrupted.
  - Uses MUTEXes (XOR gates) to implement INT masks.
- prevents an interrupt from being interrupted (which would cause the computer to become unresponsive, crash or entering an unstable state).
- An INT signal which occurs while another INT is being processed times-out and is repeated.

COMMON TYPES OF INTS

- Clock – actually a count-down timer which issues an INT each time it gets to 0.
- Keyboard/Mouse
- Error – e.g. divide by zero error detected by the ALU.
- Network – a packet being received by the NIC (network interface controller).
- Exception – generated by a try/catch instruction in software.
- SysCall – INT 80 (in assembly) triggers a kernel or system command to be executed.
- Hardware – e.g. the power button generates an ATX event, CDROM ready.
- GUI events – click, drag, button up, onfocus, onload...

POLLING

- Polling is an alternative approach to interrupt
- Check state/input of each hardware device in defined sequence:
  - process any change/input as needed.
- Issues:
  - Can waste time checking hardware which is doing nothin
  - Doesn't take advantage of the stack.
  - If one device freezes, this can make the entire computer unresponsive.
- BUT! Relatively simple to implement



Do you watch it...
or wait for the ding?

Examples: CDROM INT handlers as the CD is first read (***busy wait***). Also USB device checking

SUMMARY

- Interrupts:
  - Different processes/devices need CPU attention.
  - Interrupts manage how CPU's handle these signals.
  - Stacks provide basis for storing and recalling state while an INT is handled
- Polling:
  - An alternative based on explicitly checking the state of devices/processes
  - Simple to implement but generally considered wasteful of CPU cycles.

IN THE LAB...

- Play with Stacks ☺