I/O Basics

• Each I/O device communicates with the computer through a set of I/O registers (ports) which include data, control, and status bits. For example, consider a keyboard:

  Input Port
  8 bits

  INPR

  Input Flag
  1 bit

  FGI

  Output Port
  8 bits

  OUTR

  Output Flag
  1 bit

  FGO

• When a key is hit, ASCII code for that key is stored in INPR and FGI is set to 1.

• As long as FGI is set to 1, no other key is accepted.

• The computer keeps checking FGI, whenever it sees that it is set to 1, it copies INPR into memory/register and resets FGI.

• This protocol is called Programmed-control I/O since the computer keeps checking for the next input.

• Output will be handled in a similar way:

• The disadvantage of Programmed-control I/O is that the computer idles a long time before keyboard can provide the next character.

Interrupt driven I/O

• A better protocol is to have the computer and IO device work independently. Whenever the key is hit, its ASCII value is stored in INPR and FGI is set.

• FGI is then sent to the computer as an interrupt signal.

• At that time, when the current instruction completes, the computer interrupts the current program, saves the current state, and goes to an interrupt service routine to read in INPR into memory/register and reset FGI.

• FGI and FGO are used as interrupt signals.

• In interrupt driven I/O, we would like to have the capability to disable/enable interrupts
Use another Flag, such as \textit{IEN} (interrupt enable), which can allow the computer to ignore/process the interrupt signals.

- This is useful for ? (Answer)

\section*{I/O Instructions}

- Do we need \textit{special I/O} instructions?
- Not if we use \textit{Memory-Mapped I/O}:
  - I/O ports may be mapped into memory (Memory Mapped I/O); i.e., I/O ports use a portion of the memory (and the same address space as memory).
  - In that case, one can use \textit{load/store instructions} to transfer ports to registers, check the flags, and set or reset them (bit manipulation instructions are useful).
- However, if I/O ports use their \textit{own} address space:
  - We will need special I/O instructions to identify the port number as part of the instruction; For example:

\begin{verbatim}
inp     reg, port ; register:=port
out     port, reg ; port:=register
ski     port     ; skip on input flag
sko     port     ; skip on output flag
ion     ; interrupt on (IEN:=1)
ioff    ; interrupt off (IEN:=0)
\end{verbatim}

\section*{Interrupt Service Routine}

- Interrupt Service Routine is a program (subroutine) written by the \textit{system developer} which is used to carry out operations needed to handle an interrupt. For example, in our case:

1. Disable interrupts.
2. Check to see which interrupt flag is set.
3. Transfer between register and port accordingly.
4. Reset the flag.
5. Enable interrupt (\textit{ion}).
6. Go back to the interrupted program.
- This routine can be stored anywhere in the memory. However, its \textit{beginning address} must be
saved at a known location.

- For example: Address 1 holds the beginning address of the interrupt service routine.

- Also, before we go to an Interrupt Service Routine, we must save the point of return at a known location so we can go back to the original program; e.g.: Address 0.

- We check for an interrupt at the end of each cycle. If there is an interrupt, we'll go to the interrupt service routine in the next cycle.

**Vectored Interrupts**

- Typically there are multiple devices that can interrupt the CPU for service.

- We need mechanisms to:
  - Arbitrate among multiple simultaneous interrupt service requests
    - Round robin
    - Priority based
  - Isolate the interrupts for one-at-a-time service
  - Go to the correct interrupt service routine for each interrupt.
**IO Channel**

- Interrupt-driven IO relieves the CPU from waiting for every IO event.
- But the CPU can still be bugged down if it is used in transferring IO data.
  - Typically *blocks* of bytes.
- Thus, specialized processors, called **IO channels** are used that are capable of controlling an I/O block transfer between an IO device and the computer’s memory independent of the main processor.
- IO channels (also called IO processors or IO controllers) operate either from fixed programs in their ROM or from programs downloaded by the OS in their RAM.
  - Example: **DMA** (Direct Memory Access) controller transfers a block of information between memory and an IO device.
DMA

- Consider printing a 60-line by 80-character page
- **With no DMA:**
  - CPU will be interrupted 4800 times, once for each character printed.
- **With DMA:**
  - OS sets up an I/O buffer and CPU writes the characters into the buffer.
  - DMA is commanded (includes the beginning address of the block and its size) to print the buffer.
  - DMA will take items from the block one-at-a-time and performs everything requested.
  - Once the operation is complete, the DMA sends a single interrupt signal to the CPU.

I/O Communication Protocols

- Typically one I/O channel controls multiple I/O devices.
- We need a two-way communication between the channel and the I/O devices.
  - The channel needs to send the command/data to the I/O devices.
  - The I/O devices need to send the data/status information to the channel whenever they are ready.

Channel to I/O Device Communication

- Channel sends the address of the device on the bus.
- All devices compare their addresses against this address.
  - Optionally, The device which has matched its address places its own address on the bus again.
    - First, it is an acknowledgement signal to the channel;
    - Second, it is a check of validity of the address.
- The channel then places the I/O command/data on the bus received by the correct I/O
device.

- The command/data is queued at the I/O device and is processed whenever the device is ready.

**I/O Devices to Channel Communication**

- The I/O devices-to-channel communication is more complicated, since now several devices may require *simultaneous* access to the channel.
  - Need arbitration among multiple devices (bus master?)
  - Need priority scheme to handle requests one-at-a-time.
- There are 3 methods for providing I/O devices-to-channel communication

**Daisy Chaining**

- Two schemes
  - **Centralized control** (priority scheme)

- The I/O devices activate the request line for bus access.
- If the bus is not busy (indicated by no signal on busy line), the channel sends a Grant signal to the first I/O device (closest to the channel).
  - If the device is not the one that requested the access, it propagates the Grant signal to the next device.
  - If the device is the one that requested an access, it then sends a busy signal on the busy line and begins access to the bus.
- Only a device that holds the Grant signal can access the bus.
- When the device is finished, it resets the busy line.
• The channel honors the requests only if the bus is not busy.

• Obviously, devices closest to the channel have a higher priority and block access requests by lower priority devices.

• Decentralized control (Round-robin Scheme)

The I/O devices send their request.

The channel activates the Grant line.

The first I/O device which requested access accepts the Grant signal and has control over the bus.

Only the devices that have received the grant signal can have access to the bus.

When a device is finished with an access, it checks to see if the request line is activated or not.

If it is activated, the current device sends the Grant signal to the next I/O device (Round-Robin) and the process continues.

Otherwise, the Grant signal is deactivated.

Polling

• The channel interrogates (polls) the devices to find out which one requested access:
• Any device requesting access places a signal on request line.

• If the busy signal is off, the channel begins polling the devices to see which one is requesting access.
  □ It does this by sequentially sending a count from 1 to \( n \) on \( \log_2 n \) lines to the devices.

• Whenever a requesting device matches the count against its own number (address), it activates the busy line.

• The channel stops the count (polling) and the device has access over the bus.

• When access is over, the busy line is deactivated and the channel can either continue the count from the last device (Round-Robin) or start from the beginning (priority).

**Independent Requests**

• Each device has its own Request-Grant lines:
  □ Again, a device sends in its request, the channel responds by granting access
  □ Only the device that holds the grant signal can access the bus
  □ When a device finishes access, it lowers its request signal.
  □ The channel can use either a Priority scheme or Round-Robin scheme to grant the access.
I/O, Cache, and Virtual Memory

- Because of caches and virtual memory, potentially there may be 3 copies a data:
  - In the cache
  - In the main memory
  - On the disk
- This brings about the possibility of stale data.
- Either the OS or the hardware need to ensure that CPU and I/O channels operate on the most current data values.
- Option 1: CPU and I/O Channels share the cache:
  - There are no stale data problem between CPU and I/O operations; as both access the latest version of data in the cache.
- But, CPU performance will be degraded substantially since I/O will replace cache blocks with data that may not be used by the CPU in the near future; i.e., locality of effect will be lost.
- Also, access to the cache needs to be arbitrated between the CPU and I/O channels.

- Option 2: CPU and I/O Channels access main memory directly:

- In the second option 2 stale-data problem exist:
  - The I/O system sees stale data on output because memory is not up-to-date.
    - A write-through cache solves this problem since the cache and memory are always coherent.
    - A write-back cache requires the hardware or OS to check the output block addresses to make sure they are not in the cache (time-consuming).
  - The CPU sees stale data in the cache on input after the I/O system has updated memory.
    - Either the hardware or OS need to ensure that the input data area can’t possibly be in the cache.
• With a virtual memory system, there is a question of whether an I/O channel or DMA should transfer using virtual addresses or physical addresses.

• Use of physical addresses for transfers has a couple of major problems:
  - Transferring a buffer larger than a page can cause serious problems since pages of the buffer are probably not in sequential physical memory locations.
  - While the DMA is accessing one page of a buffer, other pages may be replaced or relocated by the virtual memory.

• Virtual DMA: DMA uses virtual addresses that are mapped to physical addresses during a DMA operation.
  - Buffer needs to be sequential in the virtual memory, but the pages can be scattered in physical memory.
  - Address translation registers (similar to a page table) are used to contain the physical page address corresponding to each virtual page (registers are updated by the OS).