Computer Organization

COMP2120

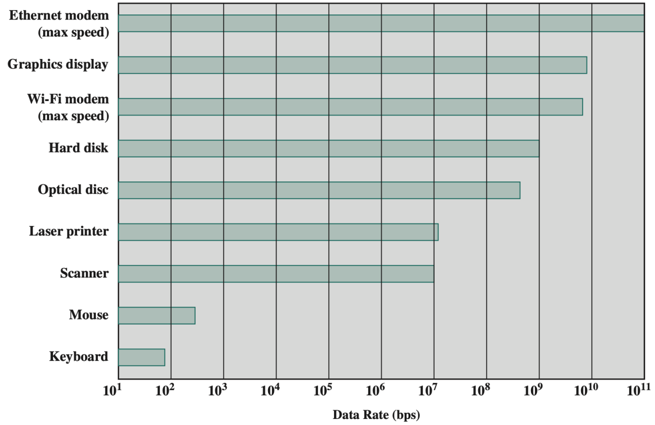
Qi Zhao

March 17, 2024

Input and Output

Input/Output

- Typical device data rate shown in the following figure.
- Device with large speed variation, cannot controlled by a single clock (synchronous communication).
- Hence asynchronous communication is employed.



Generic Model of an I/O Module

- Interface to the processor and memory via the system bus or central switch
- Interface to one or more peripheral devices by tailored data links.

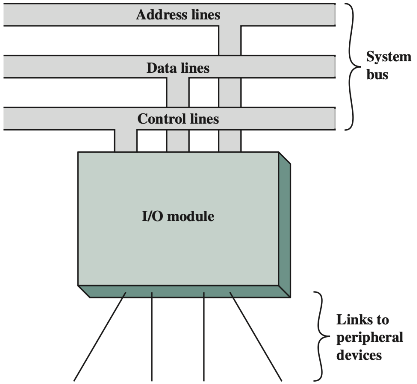
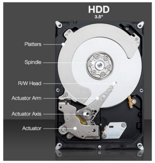


Figure 7.1 Generic Model of an I/O Module

External Devices

- Human readable: suitable for communicating with the computer user; video display terminals (VDTs), printers, mouse, keyboard

- Machine readable: suitable for communicating with equipment; magnetic disk and tape systems, sensors and actuators,



External Devices

- interface is in the form of control, data, and status signals.
- Control logic, buffer, transducer

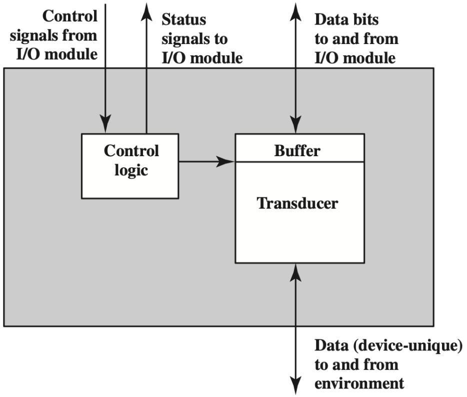

Control signals from I/O module
Status signals to I/O module
Data bits to and from I/O module
Control logic
Buffer
Transducer
Data (device-unique) to and from environment

Figure 7.2 Block Diagram of an External Device

Major requirement on an I/O Modules

Control and Timing

- Asynchronous timing.
- Example sequence:
  - processor send a request to the I/O module
  - I/O module acknowledge with device status.
  - If device ready, processor send the data by means of a command to the I/O module.
  - I/O module obtain data from device, (device access time)
  - data transferred back to the processor.

Processor/Device Communication

- command decoding — decode command sent from the bus, e.g. SEEK, READ etc.
- Data - exchange data via data bus
- Status reporting - e.g. Busy, paper out (for printer) etc.
- Address recognition - identify address of the peripheral

Major requirement on an I/O Modules

Data buffering

- store the data in buffer (because device is very slow), so that the transaction can be more efficient
- Wait for device to fill a block, and transmission is then in blocks, instead of in words).

Error detection and correction

Block Diagram of an I/O Module

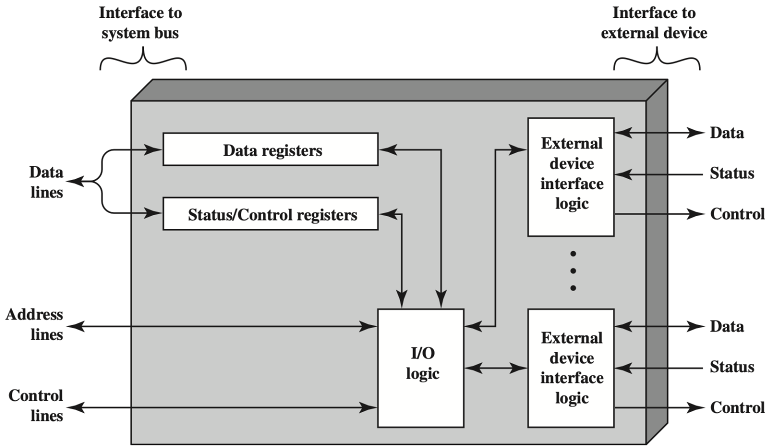


Figure 7.3 Block Diagram of an I/O Module

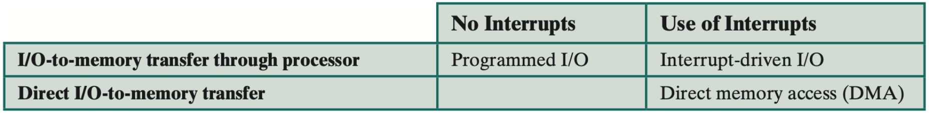
Memory-mapped I/O vs I/O Ports

- CPU controls the operation of the I/O devices by writing/reading the data registers and status/control registers. (fig 7.3)
- Status registers reflects the status of the devices, e.g. Device ready/not ready, Printer Paper Out, Data overrun etc.
- Control registers control the operation of the devices, read/write operation, paper form feed in printer etc.
- Data register (read/write) for the data.
- **Port-mapped I/O**: The registers can be in dedicated I/O ports if I/O instructions are provided in the CPU.
- **Memory-mapped I/O**: More often, these registers are put in the memory map, and I/O operations are performed using memory read/write operations — Memory-mapped I/O.

I/O Techniques

- Programmed I/O
- Interrupt-driven I/O
- Direct Memory Access (DMA)

Table 7.1 I/O Techniques



	No Interrupts	**Use of Interrupts**
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Programmed I/O

- The processor executes a program that gives it direct control of the I/O operation.
- When processor send a command, it must wait until the operation is finished, e.g. by repeatedly checking the status of the device.
- The Control and Status Register (**CSR**) is used to control the operation of the device, and report its status.
- For example — for a hard disk controller:
  bit 0 of CSR for hard disk seek,
  bit 1 for hard disk read,
  bit 2 for hard disk write
  bit 3 hard disk is ready
  bit 4 hard disk is busy
  bit 5 hard disk error ···

Programmed I/O

One example: final exam problem

(d) A simple digital clock contains a hardware timer unit and six LED displays which can display digits from 0 to 9.

The timer has a control and status register (`TCSR`) and a buffer regsiter (`TBR`) which contains time in seconds. `TCSR` is in the following format:

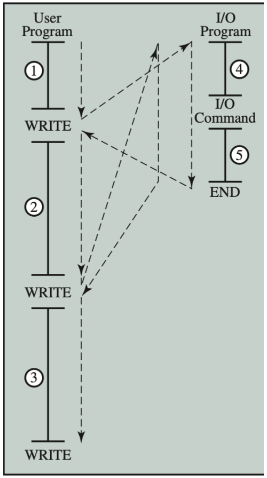
Bit 0	Ready Bit, the timer is ready.
Bit 1	Timer data ready, time is read and ready in timer buffer register. This bit will be reset to 0 once data is read.
Bit 2	Reset timer, time is set to 0.
Bit 3	Start counting.
Bit 4	Take snapshot: read the current time, timer continue counting.
Bit 5	Stop counting.

The six LED display is controlled by six LED buffer registers `LEDBR1` to `LEDBR6`. To write to the LED, just write a value between 0 and 9 to them. LED1 is the leftmost LED (LED1 and LED2 display the hour, LED3 and LED4 display the minute, and LED5 and LED6 display the second).

Write an assembly language program that use *Programmed I/O* to write six 0's to the LEDs. Then check if the timer is ready, start the timer, and continuously read the timer (snapshot) and display the current timer reading in hour, minute and second. Assume all values are valid, and no checking is required. [9]

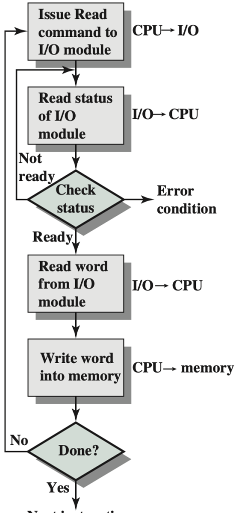
Programmed I/O

- For a read operation: CPU sends a command to the Control and Status Register (CSR) of the I/O device.
- again: CPU reads the status of the device from the CSR.
- If data is not ready, then goto again
- Read the data from the Buffer Register of the I/O device.
- Will waste CPU cycle waiting for the device, especially for slow device, such as the printer.



(a) No interrupts

Programmed I/O



(a) Programmed I/O

Interrupt-driven I/O

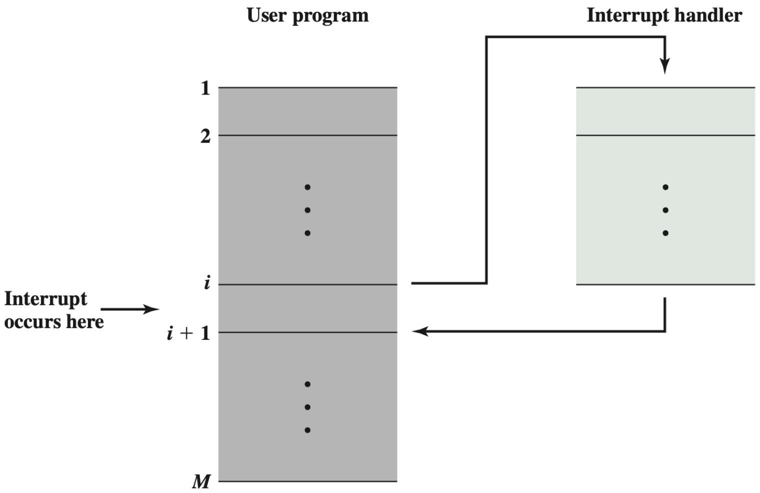
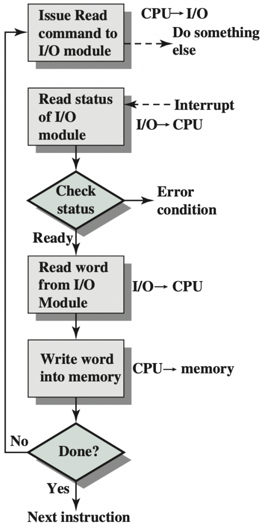


Figure 3.8 Transfer of Control via Interrupts

Interrupt-driven I/O

- Try to alleviate the problem of Programmed I/O.
- In Programmed I/O, the CPU waits for the device to finish its I/O operation. This is a waste of CPU time.
- In Interrupt-driven I/O, the processor issue an I/O command, then continue to execute instructions from other process.
- When I/O finishes, I/O module interrupts the CPU.
- The CPU will then suspend current program, execute the remaining I/O operations (as in Programmed I/O), then return to the suspended program.

Interrupt-driven I/O



(b) Interrupt-driven I/O

Interrupt Processing — Hardware side

- device issue an interrupt signal to processor
- processor finish current instruction execution
- processor check for interrupt, find one, send an interrupt acknowledge (INTA) signal to device.
- Device remove interrupt.
- Processor need to remember the current position of the program before jumping to the interrupt servicing routine. This is done by putting the PC and the flag registers (Processor Status Word PSW) to the stack.
- Processor then load PC with address of interrupt routine.

Interrupt Processing — Software side

- The Interrupt routine will save those registers that have been used by the interrupt routine into the stack (so that they can be recovered).
- The Interrupt routine perform required action, e.g. Reading data from device.
- When finished, the interrupt routine will restore the saved registers.
- The last instruction executed by the interrupt routine is a Return from Interrupt instruction (`RETI`), which will restore the PSW and PC from the stack, in the reverse order that they were saved.

Interrupt-driven I/O

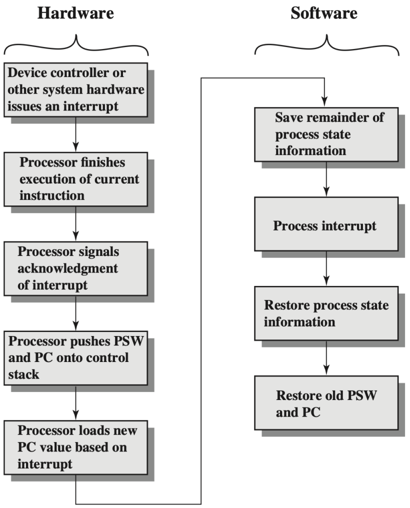
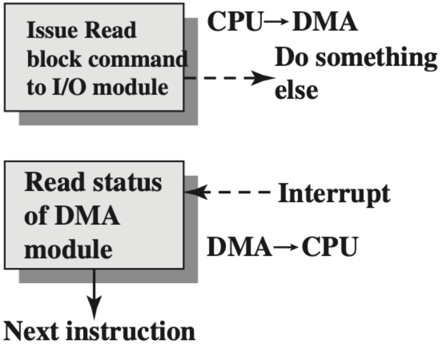

Hardware
Software
Device controller or other system hardware issues an interrupt
Processor finishes execution of current instruction
Processor signals acknowledgment of interrupt
Processor pushes PSW and PC onto control stack
Processor loads new PC value based on interrupt
Save remainder of process state information
Process interrupt
Restore process state information
Restore old PSW and PC

Figure 7.6 Simple Interrupt Processing

Direct Memory Access (DMA)


Issue Read block command to I/O module
CPU→DMA
Do something else
Read status of DMA module
Interrupt
DMA→CPU
Next instruction

(c) Direct memory access

Direct Memory Access (DMA)

- Interrupt driven I/O still involves CPU operation.
- By using intelligent device controller (i.e. the controller contains an I/O processor), we can minimize CPU intervention.
- Special purpose processor called Input-Output Processor (IOP)
- The CPU will tell the I/O processor to perform the following, for example: Read the device and place the data in memory location x to y.
- The I/O processor will directly write to the memory.
- However, CPU will use the memory, say to read instruction, or operand in every instruction execution.
- How to resolve this conflict (both CPU and I/O processor wants to write to the memory at the same time)?

Direct Memory Access (DMA)

- The I/O processor will steal cycles from the CPU:
- The I/O processor issues a signal to tell the CPU to disconnect itself from the buses, and let the I/O device control the memory bus.
- During this period, the CPU will see an elongated clock, and the CPU will wait for the end of the clock cycle to perform its next action. Different with Interrupt.

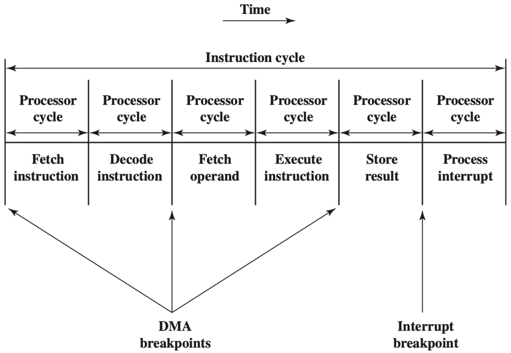
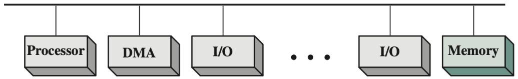


Figure 7.13 DMA and Interrupt Breakpoints during an Instruction Cycle

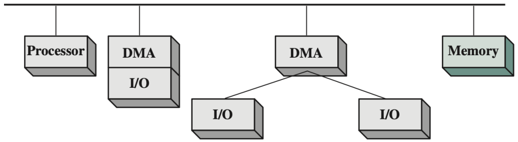
Direct Memory Access (DMA)

- The I/O processor reads/writes the memory for one cycle. When the I/O processor finishes with the current word, it removes the signal, and the clock will returns to normal.
- CPU will continue its normal operation.
- When I/O processor finished with the entire I/O operation, it interrupts the CPU, notify the end of I/O operation

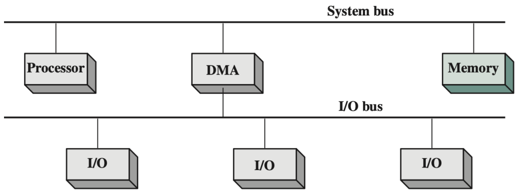
Direct Memory Access (DMA)



(a) Single-bus, detached DMA



(b) Single-bus, integrated DMA-I/O



(c) I/O bus