Computer Architecture CS 355

Busses & I/O System

Text:
Computer Organization & Design, Patterson & Hennessy
Chapter 6.5-6.6

Objectives:
During this class the student shall learn to:

- Describe the two basic transactions that occur over a bus.
- Describe the basic use of the data, address, and control lines for a bus.
- Define the difference in operation between synchronous and asynchronous busses.
- Define basic definitions: memory-mapped I/O, polling, interrupts, parallel, serial, DMA, hot-pluggable, daisy-chained.
- Relate how interrupts affect processing in an operating system
- Define interrupt vector, device driver, interrupt service routine, device controller.
- Contrast the efficiency of polling, interrupts, and DMA.
- Describe the 9 steps a processor takes to process interrupts (i.e., memorize).

Time Allocation:
Class time will be allocated as follows:

Bus	1 hour
I/O & Interrupts	1 hour
Exercise	1 hour
TOTAL:	3 hours

Bus Interfaces

Bus: Shared communication link, which uses a set of wires to connect multiple subsystems

- Can create a bottleneck since all commands/data pass through the bus

Bus transaction:
Types of Transactions:

- Read from memory or device
- Write to memory or device

Operation includes:

- Sending an address
- Sending or receiving data

Bus contains transactions or separate lines for:
C**ontrol (lines)**: Provides transaction command/response. For example:

- ReadRequest: Proc → Device: Indicates data lines contain address to read
- DataReady: Proc ←Device: Indicates that the data lines contain data to write
- Ack: Acknowledges a DataReady or ReadRequest command

D**ata (lines)**: Transmit data
A**ddress (lines)**: Transmit address of memory or device to interface with
Handshaking protocol: Sequence of transmissions completes the transaction.

- Standards define the protocol

Communications Schemes

Synchronous: Includes a clock (= a metronome) to time input/output

- Protocol interface defined as a schedule
- Requires every device to operate at that clock rate
- Requires short bus length to avoid clock skew
- Used for Processor-Memory Buses

Asynchronous: No clock is provided

- Requires a handshaking protocol to coordinate transmission
- Advantages: Variable speed, flexible bus length
- Used with I/O devices

Bandwidth:

- **Serial**: Transmits one bit at a time
- **Parallel**: Transmits multiple bits at a time
- Impacts speed: clock rate, number of pins (bits), optional address pins, etc.
- Currently high-speed serial point-to-point switched interconnections are popular

Robustness:

- **Hot pluggable**: Can plug in/remove nodes while bus is operating
- **Parity checking**: Includes an extra bit for error detection
- Detect and isolate malfunctioning units

Arbitration scheme:
Required when multiple nodes connected to a bus

- **Daisy chained**: Extensions connect different nodes in single line. Nodes share request & grant signal and are linked according to priority.
- Self-selection: Addresses: 0001, 0011, 0111, 1111: First 1 bit transmitted wins.

Types of buses include:

- **Processor-Memory Bus**: Transfers data between the processor and memory
  Usually synchronous
  PC: Part of North Bridge
- **I/O Bus**: Transfers data between devices and processor.
  Includes: FireWire, USB, SCSI, PCI
  Usually asynchronous
  PC: Part of South Bridge

	Internal/External	**Data width**	**Peak Bandwidth**
Firewire	External – serial bus	4	50-100 MB/sec
USB 3.0	External - Devices	2	0.2 MB, 1.5 MB, 60 MB/sec, 10 Gb/sec
PCI Express (Periph. Component Interconnect)	Internal – computer expansion bus	2 per lane	250 MB/sec (=1x) 1, 2, 4, 8, 15, 32 x
Serial ATA	Internal - Disk	4	300 MB/sec
SCSI (Small Comp. Sys. Interface)	External – electrical ribbon or optical connectors	4 (8/16)	300 MB/sec

Memory-Mapped I/O: Addresses are allocated to specific devices. For example:

- 0-ffff0000: Memory
- 0xffff0000 (32 bits): Control Register for Keyboard
    - Bit 0: Device Ready: Data is ready to be read
    - Bit 1: Interrupt Enable: Device indicates if interrupts are enabled

  0xffff0004 (32 bits): Keyboard data (to be read)
    - Bits 0-7: One ASCII byte read from keyboard
- ffff0008 (32 bits): Display Control Register
    - Bit 0: Device Ready: Data has been written
    - Bit 1: Interrupt Enable: Device indicates if interrupts are enabled

  0xffff0004 (32 bits): Data to be displayed (or written)
    - Bits 0-7: One ASCII byte written to display

How it works

- When Data byte is read or written to, it clears the Device Ready
- Processor has an interrupt mask which defines which devices it is willing to accept interrupts from

Alternative Scheme:

- Device I/O Commands: Special assembly language instructions exist to write to device

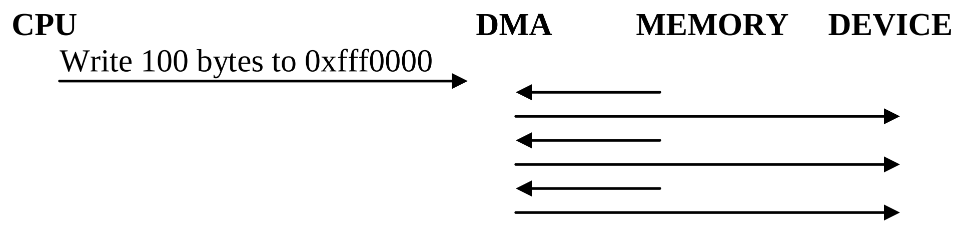
Polling: The processor regularly polls a device to determine if has completed its I/O operation
    While (Device_Ready == 0) ;

Interrupts: The device raises an interrupt control line to inform the processor when the I/O operation is complete. The processor acknowledges via an interrupt ack control line.

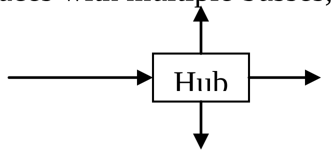
Devices

DMA: Direct Memory Access: A device, which given an address and a length, will move the associated memory to or from a device one byte/word at a time

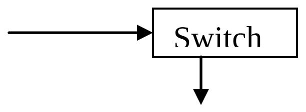
- Prevents processor from having to handle interrupts on a word or byte basis



Hub: Device interfaces with multiple busses, repeats what it receives in all directions



Switch: Device interfaces with multiple busses, repeats what it receives in the direction of the destination



I/O Hardware – Operating System Interface

Devices include:
- Storage devices: disk, floppy disk
- Communications devices: keyboard, display, serial port

Two types of devices:
Block-oriented: Transfers occur a block at a time (block = N bytes)
- E.g. disks, tapes.

Stream-oriented: Transfers occur one byte at a time. Includes:
- line-at-a-time: Processed only when End of Line is received
    - E.g. terminal, network communications port
- byte-at-a-time: Each keystroke is significant.
    - E.g. mouse, sensors.

Computer-Device Interface

Components of a Computer System
- Application Programming Interface (API): Common application interface
- **Device Driver**: S/W in O.S. memory gives orders to the device controller
- **Interrupt Vector**: Array of pointers to **interrupt service routines**, indexed by interrupting device.
    - In lower addresses of memory
    - Alternatively a **Cause register** provides the source of the interrupt
    - An **Interrupt Mask** is a set of bits defining the interrupt priorities that could be handled

Bus Interface: How Computer processor communicates with device

Components of Device
- **Device Controller**: Intelligence that drives device & communicates with CPU.
- Device (H/W)

Device Driver: Logic to control an external device.
- Includes logic to initialize device
- Initiates transmit or receive operation by placing command in command register of device controller
- Includes **interrupt service routine**: Processes the interrupt when the operation is complete.

Reconfigurable Device Drivers
- Device & driver from third party (not OS distributer)
- Eliminates the need to recompile O.S. for each new driver
- Open Systems allow plug-in device drivers
    - Defined API
    - Allows device driver to allocate buffer space & manipulate kernel tables.

Device Controller: The intelligence in the device which interfaces with the computer

- Controller runs I/O Program and sets Status bits to indicate the status of the last requested operation:
  - Busy: Device is still executing the command
  - Done: Operation is completed
  - Error: Error code if operation failed

Performing I/O

Example: The device driver has a number of characters to send to a terminal.
To Begin Transmission:

- The device driver requests a UART (Universal Asynchronous Receiver Transmitter) to transmit a byte. At 9600 bits per second, the byte will finish transmitting in 1 ms (0.00104 second).
- While the byte is transmitting the processor can wait OR the processor can do other work.

Three methods of receiving status from Device Controller:

1. If **polling** is used the device driver will busy-wait (test repeatedly) to see if the UART is done.
2. If **interrupts** are used the processor performs other work until the UART indicates (via interrupt) that the operation (transmit byte) is complete.
3. If **DMA** is used the DMA controller sends the entire packet and interrupts when packet is sent.

- When the processor determines that the byte is transmitted (or received) the device driver performs an I/O command to transfer (or receive) the next byte.
  - Bytes are received/sent from a buffer, which must remain in memory.
  - When buffer is empty, the interrupt must schedule a process to initialize new buffer for transmit/receive and reissue communications command.

Types of Interrupts
High Priority: Cannot be inhibited

- Hardware Failure: E.g. Memory parity error or power failure
- Program:
  - Arithmetic overflow
  - Divide by zero
  - Execute illegal instruction

Lower Priority: Inhibitable

- Timer: Interrupts the processor every n time units.
- I/O: Signals completion of an operation or an error condition.
- Software Trap: Command in system call interface changes mode from user to kernel mode

Definitions:
Steps by device driver to initiate an I/O operation include:

1. Wait for I/O device to become free
2. Prepare to request an I/O device to do an operation: Set up registers, buffers.

3. Perform the I/O command to initiate the operation.
4. When the external device is completed, the device interrupts the processor.

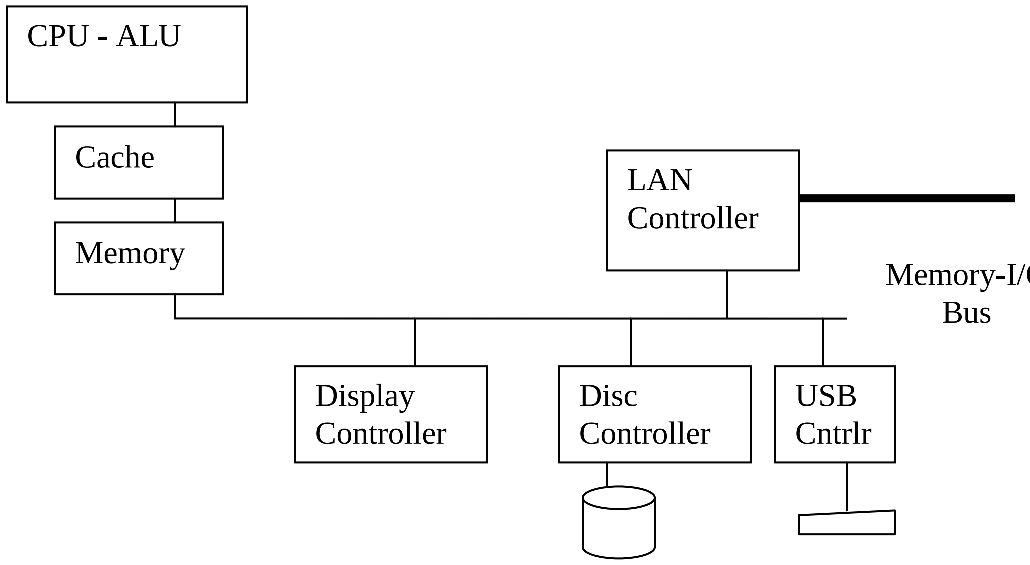
Interrupt Processing includes:

1. Device controller issues an interrupt signal to the processor.
2. Processor finishes execution of current instruction before checking for interrupt.
3. Processor sends an acknowledgment signal to the device that issued the interrupt.
4. Processor saves info on currently executing process (e.g. address of next instruction) on stack.
5. Processor jumps to interrupt service routine.
6. Interrupt service routine saves off all registers (it will be using) onto stack.
7. Interrupt service routine handles the interrupt: e.g. put received data in a buffer or get next byte from buffer to send; may start another I/O operation.
8. Interrupt service routine restores registers that were modified from stack.
9. Restore registers from stack and return to interrupted program.

Draw Interrupts:

Show the nine steps of processing an interrupt on the following graph. Assume input was entered from the keyboard.
Interrupt Processing includes:

1. Device controller issues an interrupt signal to the processor.
2. Processor finishes execution of current instruction before checking for interrupt.
3. Processor sends an acknowledgment signal to the device that issued the interrupt.
4. Processor saves info on currently executing process (e.g. address of next instruction) on stack.
5. Processor jumps to interrupt service routine.
6. Interrupt service routine saves off all registers (it will be using) onto stack.
7. Interrupt service routine handles the interrupt: e.g. put received data in a buffer or get next byte from buffer to send; may start another I/O operation.
8. Interrupt service routine restores registers that were modified from stack.
9. Restore registers from stack and return to interrupted program.


CPU - ALU
Cache
Memory
LAN Controller
Memory-I/O Bus
Display Controller
Disc Controller
USB Cntrlr