Chapter 9

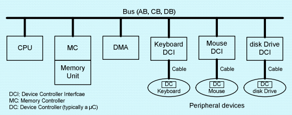
COMPUTER ARCHITECTURE: INTERCONNECTION

In this chapter

- History of microcomputer architectures
- Device controller and interface
- Device communication

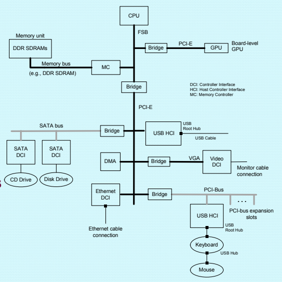
Microcomputer Architecture (1)

1. Single bus architecture
   - One bus to interconnect all system components
   - No distinctions was made between slow and fast components
   - Today, typically the architecture of small embedded systems
     - E.g., microcontrollers



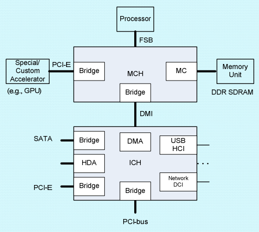
Microcomputer Architecture (2)

2. Multi-bus architecture
   - Separate buses improved performance
   - Improved CPU-memory communication
   - Bridge interconnects two different buses
     - Converts protocol used by one bus to that used by another bus
   - Multi-bus allowed also using standard buses
     - Reduced cost



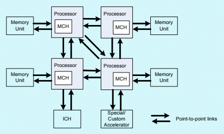
Microcomputer Architecture (3)

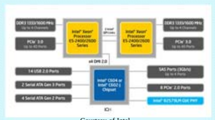
3. Integrated architecture
   - Simplified system organization
     - Motherboards with fewer but more complex components
     - Reduced cost
- Still UMA Architecture
  - Non-scalable
  - MCH became bottleneck as number of CPUs (cores) increased



Microcomputer Architecture (4)

4. Scalable interconnection
   - Creates NUMA architecture
     - Reduces average memory latency
     - Reduces program execution time
   - Requires point-to-point interconnection among processors
     - QuickPath (Intel)
     - Hypertransport (AMD)
   - Also interconnects to ICH for device communication



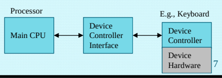


Courtesy of Intel

Device Communication

- Need two controllers
  - Each an embedded system (recall from Ch1)
    - Contains simple CPU, firmware, etc.

1. Device Controller
   - Controls device hardware
     - Keyboard matrix to read depressed keys
     - Mouse hardware to determine displacement data
     - Disk drive hardware
     - Etc.
2. Device Controller Interface
   - Facilitates communication between a Device Controller and memory and/or main CPU (processor)



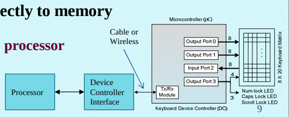
Older vs. Modern Systems

- Older Systems
  - One Device Controller Interface per device
  - Required device installation
    - Also system reboot after installation
  - Required separate cable and connection port per device
    - Supported only limited number of device connections
- Modern systems
  - Provides “plug and play” installation
    - Automatic detection and driver installation
    - Rebooting not required
  - Uses universal controller interface
    - One controller interface can stablish communication from/to many devices at once
      - E.g., USB Host Controller Interface can potentially can support 127 devices simultaneously

What are I/O ports?

- A set of fixed (physical) hardware access points
- Two main applications

1. Device Controller access points
   - Used to control device hardware
     - E.g., keyboard matrix
   - Used to configure Device Controller
     - E.g., Key repetition rate
2. Device Controller Interface access points
   - Used to set device communication mechanism
     - E.g., main CPU will poll the device or device will interrupt main CPU
   - For transferring device data directly to memory or processor
     - E.g., depressed key value send to processor



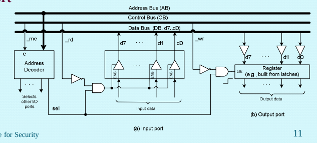
I/O Port Addressing

- Two addressing mechanisms

1. Port-Mapped I/O (also called Isolated I/O)
   - Port addresses separate from memory addresses
   - Requires separate instructions to access I/O ports
     - E.g. IN and OUT instructions on Intel processors
   - Not common with RISC processors
2. Memory mapped I/O
   - Memory address space partitioned into memory and I/O port address regions
   - Memory access instructions (e.g., LD and ST) also used to access I/O ports
   - Supported by all processors

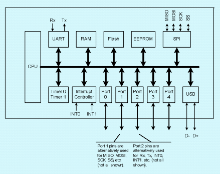
Simple I/O Ports

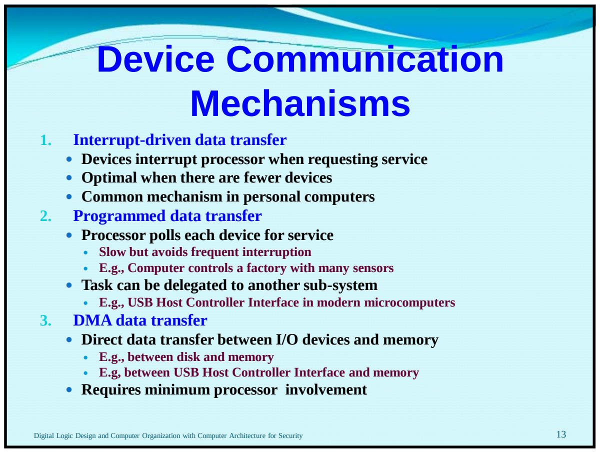
- Input ports used for inputs
  - Require tri-state buffers to output to data bus
- Output ports used for outputs
  - Require level-sensitive or edge triggered registers to load from data bus
- Access mechanism similar to memory
  - Except a port address can identify two physical ports, input and output ports
    - Port address is decoded to select I/O port
    - Read signal selects input port
    - Write signal selects output port



Device Controller and Device Controller Interface as Embedded Systems

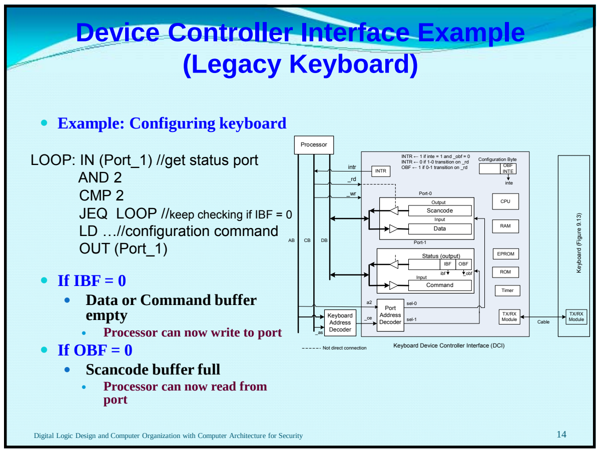
- Microcontrollers as simple embedded systems
  - CPU, RAM, EEPROM, I/O ports, data transceiver modules, etc. within a single chip
  - Applications
    - Simple Device Controller
- SoC as complex embedded system
  - Applications
    - Complex Device Controller
    - Universal controller interface
      - E.g., USB Host Controller Interface


Rx
Tx
MISO
MOSI
SCK
SS
UART
RAM
Flash
EEPROM
SPI
CPU
Timer 0 Timer 1
Interrupt Controller
Port 0
Port 1
Port 2
Port 3
Port 4
USB
INT0
INT1
D-
D+
Port 1 pins are alternatively used for MISO, MOSI, SCK, SS, etc. (not all shown).
Port 2 pins are alternatively used for Rx, Tx, INT0, INT1, etc. (not all shown).



Device Communication Mechanisms

1. Interrupt-driven data transfer
   - Devices interrupt processor when requesting service
   - Optimal when there are fewer devices
   - Common mechanism in personal computers
2. Programmed data transfer
   - Processor polls each device for service
     - Slow but avoids frequent interruption
     - E.g., Computer controls a factory with many sensors
   - Task can be delegated to another sub-system
     - E.g., USB Host Controller Interface in modern microcomputers
3. DMA data transfer
   - Direct data transfer between I/O devices and memory
     - E.g., between disk and memory
     - E.g, between USB Host Controller Interface and memory
   - Requires minimum processor involvement

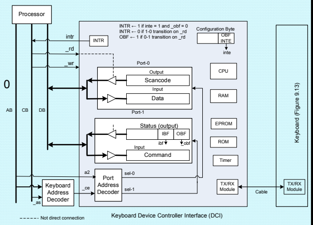


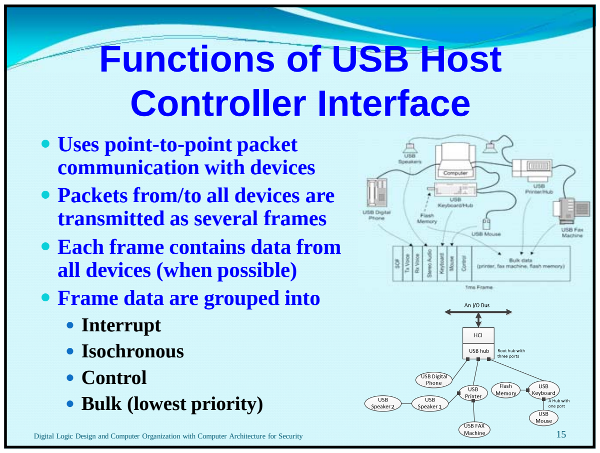
Device Controller Interface Example (Legacy Keyboard)

- Example: Configuring keyboard

```
LOOP: IN (Port_1) //get status port
      AND 2
      CMP 2
      JEQ  LOOP //keep checking if IBF = 0
      LD …//configuration command
      OUT (Port_1)
```

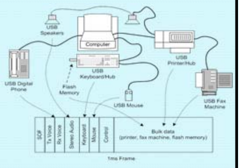
- If IBF = 0
  - Data or Command buffer empty
    - Processor can now write to port
- If OBF = 0
  - Scancode buffer full
    - Processor can now read from port

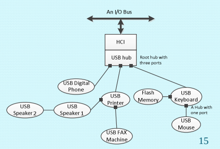


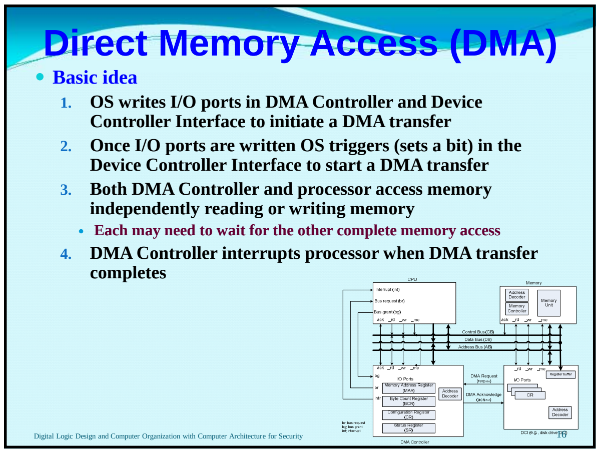


Functions of USB Host Controller Interface

- Uses point-to-point packet communication with devices
- Packets from/to all devices are transmitted as several frames
- Each frame contains data from all devices (when possible)
- Frame data are grouped into
  - Interrupt
  - Isochronous
  - Control
  - Bulk (lowest priority)

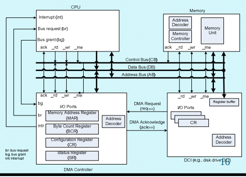


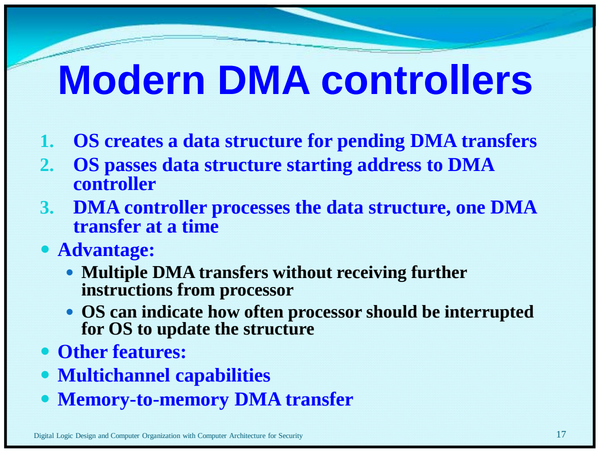




Direct Memory Access (DMA)

- Basic idea
  1. OS writes I/O ports in DMA Controller and Device Controller Interface to initiate a DMA transfer
  2. Once I/O ports are written OS triggers (sets a bit) in the Device Controller Interface to start a DMA transfer
  3. Both DMA Controller and processor access memory independently reading or writing memory
     - Each may need to wait for the other complete memory access
  4. DMA Controller interrupts processor when DMA transfer completes

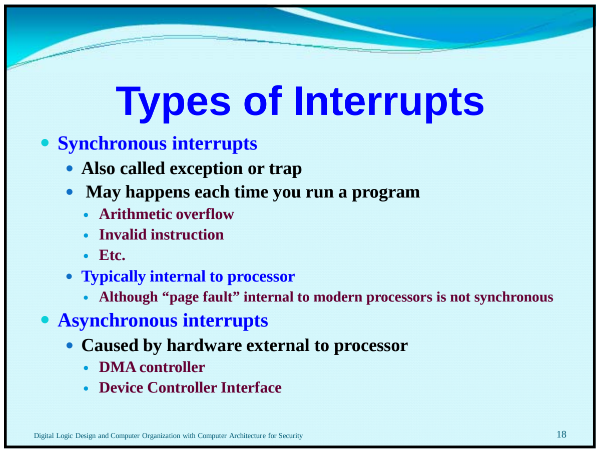




Modern DMA controllers

1. OS creates a data structure for pending DMA transfers
2. OS passes data structure starting address to DMA controller
3. DMA controller processes the data structure, one DMA transfer at a time

- Advantage:
  - Multiple DMA transfers without receiving further instructions from processor
  - OS can indicate how often processor should be interrupted for OS to update the structure
- Other features:
- Multichannel capabilities
- Memory-to-memory DMA transfer

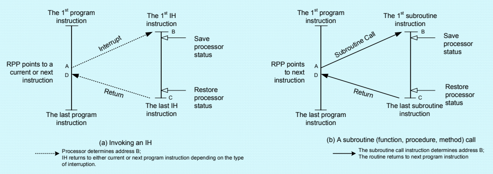


Types of Interrupts

- Synchronous interrupts
  - Also called exception or trap
  - May happens each time you run a program
    - Arithmetic overflow
    - Invalid instruction
    - Etc.
  - Typically internal to processor
    - Although “page fault” internal to modern processors is not synchronous
- Asynchronous interrupts
  - Caused by hardware external to processor
    - DMA controller
    - Device Controller Interface

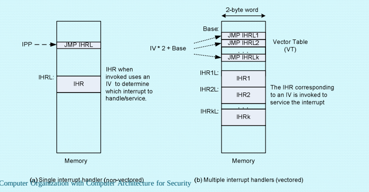
Handling Interruptions

- Subroutine call vs. Invoking Interrupt Handler (IH)
  - The subroutine address is known
  - Which IH to invoke not known, must be determined
  - CPU status is saved and restored in both cases


The 1st program instruction
The 1st IH instruction
Interrupt
Save processor status
RPP points to a current or next instruction
Return
Restore processor status
The last program instruction
The last IH instruction
(a) Invoking an IH
Processor determines address B;
IH returns to either current or next program instruction depending on the type of interruption.
The 1st subroutine instruction
Subroutine Call
RPP points to next instruction
The last subroutine instruction
(b) A subroutine (function, procedure, method) call
The subroutine call instruction determines address B;
The routine returns to next program instruction

Non-Vector vs. Vector Interrupt

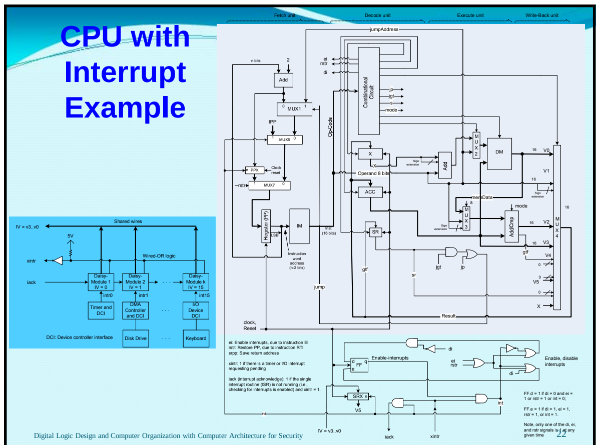
- Non-Vector Interrupts
  - Not common with modern systems
  - One IH to take care of all interrupts
    - Slow respond time
    - Interrupts are handled in some priority order
- Vector interrupts
  - OS loads IHs in memory and creates a vector table
  - IHs are invoked as interrupts occur
  - A higher priority IH interrupts a lower priority one


IPP
JMP IHRL
IHRL:
IHR
IHR when invoked uses an IV to determine which interrupt to handle/service.
Memory
(a) Single interrupt handler (non-vectored)
2-byte word
Base:
IV * 2 + Base
JMP IHRL1
JMP IHRL2
JMP IHRLk
Vector Table (VT)
IHR1L:
IHR1
IHR2L:
IHR2
IHRkL:
IHRk
The IHR corresponding to an IV is invoked to service the interrupt
Memory
(b) Multiple interrupt handlers (vectored)

Interrupt structures

- Devices are prioritized in hardware
1. Daisy chain
   - Advantage
     - Hardware scalable, can easily add more devices
   - Disadvantage
     - Low priority device may starve
     - Slow, takes time to identify the interrupting device
2. Priority encoder
   - Interrupting device quickly identified
3. Hybrid
   - Organize devices into priority classes

CPU with Interrupt Example

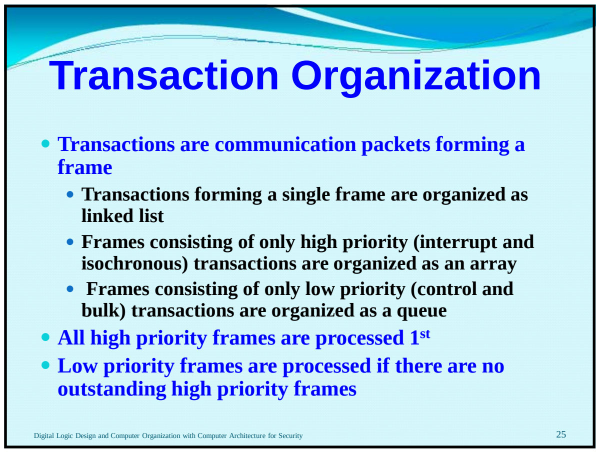


Precise Interruption

- How to determine return address
  - depends on type of interruption
  - Address of next sequential instruction or address of currently executing instruction
- CPU status upon interruption
  - depends on type of interruption
  - E.g., when Acc should not be updated upon interruption?
- Interrupts are checked in write-back stage

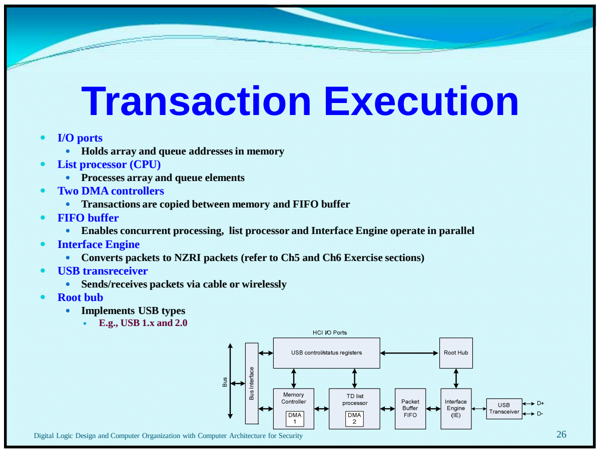
USB Host Controller Interface

- Standards
  - universal host controller interface (UHCI)
  - open host controller interface (OHCI)
  - enhanced host controller interface (EHCI)
- Types
  - USB 1.x
    - low-speed, 1.5Mb/s
    - full-speed, 12Mb/s
  - USB 2.0, high-speed, 480Mb/s
  - USB 3.0, super-speed, up to 5Gb/s
- Cable
  - Only 4 wires as Power, Ground, +D, and –D
  - NRZI (non-return-to-zero inverted) coding scheme



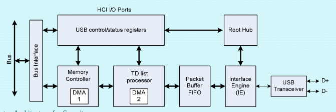
Transaction Organization

- Transactions are communication packets forming a frame
  - Transactions forming a single frame are organized as linked list
  - Frames consisting of only high priority (interrupt and isochronous) transactions are organized as an array
  - Frames consisting of only low priority (control and bulk) transactions are organized as a queue
- All high priority frames are processed 1st
- Low priority frames are processed if there are no outstanding high priority frames



Transaction Execution

- I/O ports
  - Holds array and queue addresses in memory
- List processor (CPU)
  - Processes array and queue elements
- Two DMA controllers
  - Transactions are copied between memory and FIFO buffer
- FIFO buffer
  - Enables concurrent processing, list processor and Interface Engine operate in parallel
- Interface Engine
  - Converts packets to NZRI packets (refer to Ch5 and Ch6 Exercise sections)
- USB transreceiver
  - Sends/receives packets via cable or wirelessly
- Root bub
  - Implements USB types
    - E.g., USB 1.x and 2.0


HCI I/O Ports
Bus
Bus Interface
USB control/status registers
Root Hub
Memory Controller
DMA 1
TD list processor
DMA 2
Packet Buffer FIFO
Interface Engine (IE)
USB Transceiver
D+
D-