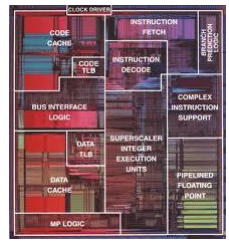
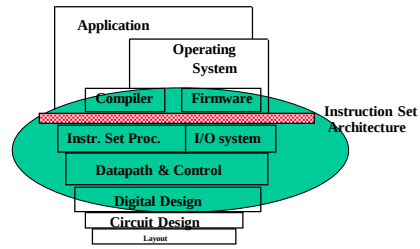
UTD

CS/SE 3340
Computer Architecture



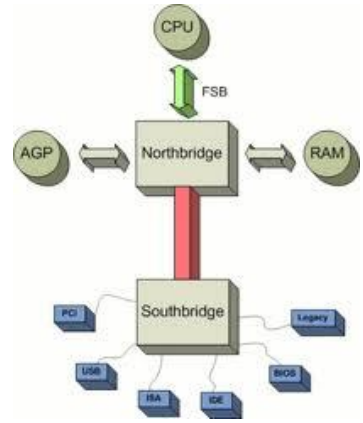


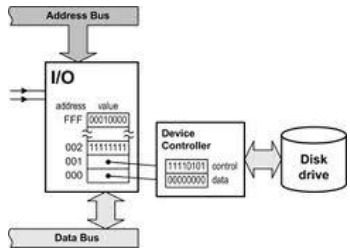
Input & Output

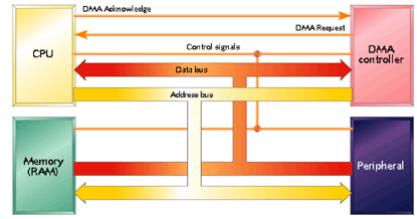
Adapted from slides by Profs. D. Patterson and J. Hennessey

Questions

- What/Why of Input/Output in computer systems?
- How to connect I/O devices physically?
- How does S/W i/f with I/O devices?
- What are I/O registers (How to abstract I/O devices)?
- What are the three important questions for I/O and How to answer them?







3

Why Input & Output?

- So far we have learnt how the processor process data in memory
  - Huge array of bytes
- But *how* does the processor process information from **outside world**?
  - Get data from a keyboard or a sensor
  - Display results of processing on screen (monitor) or paper (printer)?
  - Communicate with other system via networks?
- The processor needs to work with input & output (I/O) devices

4

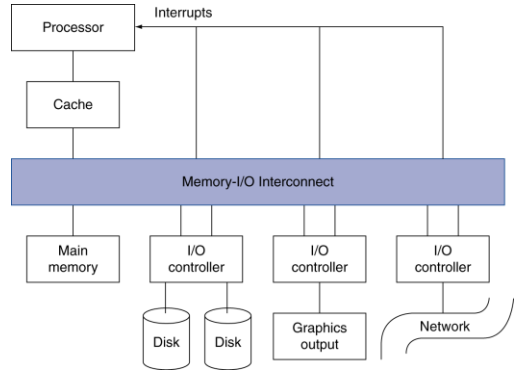
From I/O Channels to Device Drivers

- In the beginning I/O channels provides means for the CPU to work with physical I/O devices
  - I/O hardware understands commands from CPU
  - The CPU has special instructions to pass commands to I/O hardware (e.g. *Channel Control Words, CCW,* in IBM mainframes)
- Modern Operating Systems unifies I/O operations under *device drivers*
  - Layer of ***abstraction*** that hides I/O device specific details

5

I/O Devices

- I/O devices can be characterized by
  - Function: *input, output, storage*
  - Interface with the CPU: *serial, parallel*
  - Unit of data transfer: *character, block*
- I/O interconnect allows for data exchange among components



6

Interconnecting Components

- Need interconnections between
  - CPU, memory, I/O controllers
- *Bus*: shared communication channel
  - Parallel set of wires for data and synchronization of data transfer
  - Can become a bottleneck
- Performance limited by physical factors
  - Wire length, number of connections
- More recent alternative: high-speed serial connections with switches
  - Like networks

7

Bus Types

- Processor-Memory buses
  - Short, high speed
  - Design is matched to memory organization
- I/O buses
  - Longer, allowing multiple connections
  - Specified by standards for interoperability
  - Connect to processor-memory bus through a bridge
  - e.g. Universal Serial Bus (USB)

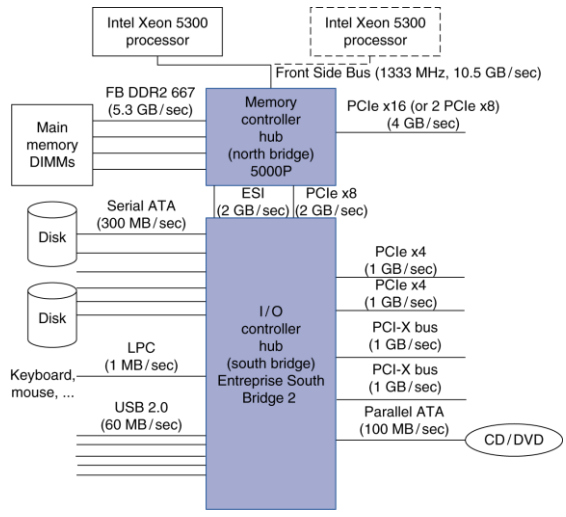
8

I/O Bus Examples

	Firewire	USB 2.0, USB 3.0	PCI Express	Serial ATA	Serial Attached SCSI
Intended use	External	External	Internal	Internal	External
Devices per channel	63	127	1	1	4
Data width	4	2	2/lane	4	4
Peak bandwidth	50MB/s or 100MB/s	0.2MB/s, 1.5MB/s, or 60MB/s 600MB/s	250MB/s/lane 1×, 2×, 4×, 8×, 16×, 32×	300MB/s	300MB/s
Hot pluggable	Yes	Yes	Depends	Yes	Yes
Max length	4.5m	5m	0.5m	1m	8m
Standard	IEEE 1394	USB Implementers Forum	PCI-SIG	SATA-IO	INCITS TC T10

9

Typical x86 PC I/O System



10

I/O Management

- In modern computer systems I/O operations are mediated by the OS
- I/O operations
  - Multiple programs share I/O resources
    - Need protection and scheduling
  - I/O causes asynchronous interrupts
    - Same mechanism as exceptions
  - I/O programming is fiddly
    - OS provides abstractions to programs

11

I/O Registers

- I/O devices are managed by I/O controller hardware
  - Transfers data to/from device
  - Abstracted as three sets of registers
- Command registers
  - Cause device to do something
- Status registers
  - Indicate what the device is doing and occurrence of errors
- Data registers
  - Write: transfer data to a device
  - Read: transfer data from a device

12

Where To Locate I/O Registers?

- *Memory mapped I/O*
  - Registers are addressed in same space as memory
  - Address decoder distinguishes between them
  - OS uses address translation mechanism to make them only accessible to kernel
  - Example: MIPS
- *I/O instructions and I/O ports*
  - Separate instructions to access I/O ports
  - Can only be executed in kernel mode
  - Example: x86 (IN and OUT instructions on I/O segment)

13

When to Perform I/O Operations ? Easy Way - Polling

- Periodically check I/O status register
  - If device ready, do operation
  - If error, take action
- Common in small or low-performance real-time embedded systems
  - Predictable timing
  - Low hardware cost
- In other systems, wastes CPU time

14

When to Perform I/O Operations? Better Way - Interrupts

- When a device is ready or error occurs
  - Controller *interrupts* CPU
- *Interrupt* is like an exception
  - But not synchronized to instruction execution
  - Can invoke handler between instructions
  - Cause information often identifies the interrupting device
- Priority interrupts
  - Devices needing more urgent attention get higher priority
  - A higher priority I/O device can interrupt handler for a lower priority I/O device

15

I/O Data Transfer – How?

- *Programmed I/O*
  - CPU transfers data between memory and I/O data registers
  - Time consuming for high-speed devices
- *Direct Memory Access (DMA)*
  - OS provides starting address in memory
  - I/O controller transfers to/from memory autonomously
  - Controller interrupts on completion or error

16

MIPS I/O

- MIPS supports *interrupts* and *DMA*
- MIPS CPU communicates with I/O devices using *memory-mapped* input/output (I/O)
- With memory mapped I/O, the I/O devices appear to be primary memory locations
  - There is no need for specific (additional) instructions to communicate with I/O devices
  - No need for I/O ports either
- What existing instructions can be used?
  - Load & Store

17

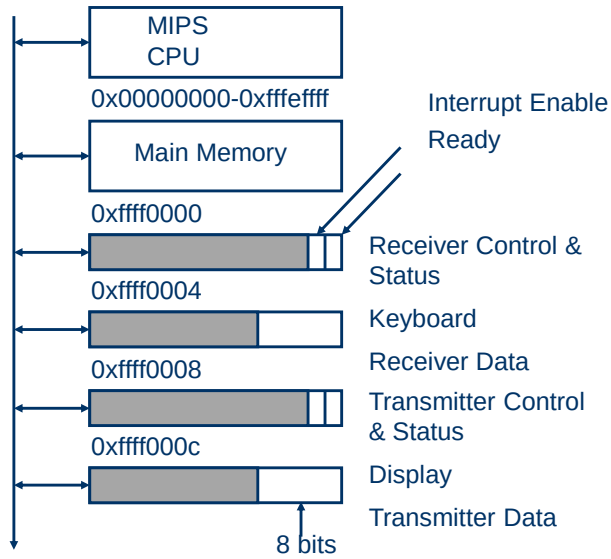
MIPS- Memory-Mapped I/O

- The MIPS I/O address space **`0xffff0000`** to **`0xffffffff`** is reserved for memory-mapped I/O
  - How large is the I/O address space?
- Any address in this space can be mapped to I/O registers (command, status, data) of an I/O device
- I/O is performed using the regular *load* and *store* instructions on these addresses
  - e.g.

```
li   $t0, 0xffff0004
lw   $s1, ($t0)
```

18

MIPS I/O Example



Keyboard Controller – C Language

Keyboard Control	0xFFFF0000	2 bits
Keyboard Data	0xFFFF0004	8 bits
Display Control	0xFFFF0008	2 bits
Display Data	0xFFFF000C	8 bits

```
char  read_ch (void) {
      volatile unsigned int  *recv;
      recv  =  0xFFFF0000;
      while ((recv[0] & 1) == 0) /* - */;
      return  recv[1] & 0x00FF;
      }
```

Keyboard Controller – MIPS Assembly

```
        .text
        .globl   read_ch
read_ch:
        subu     $sp, $sp, 4 # "push" stack
        sw       $t0, 0($sp) # save temporaries
        li       $t0, 0xffff0000 # RCR address
recv_rda:
        lw       $v0, 0($t0) # read status from RCR
        andi     $v0, $v0, 0x0001 #  check if Ready
        beq      $v0, $zero, recv_rda #  wait until
  Ready is "one"
        lw       $v0, 4($t0) #  read data from RDR
        lw       $t0, 0($sp) #  temporaries
        addiu    $sp, $sp, 4 #  "pop" stack
        jr       $ra
```

21

Display Controller – C Language

Keyboard Control	0xFFFF0000	2 bits
Keyboard Data	0xFFFF0004	8 bits
Display Control	0xFFFF0008	2 bits
Display Data	0xFFFF000C	8 bits

```
void write_ch (char c){
     volatile unsigned int  *txmit;

     txmit = 0xFFFF0008;  /* base */
     while ((txmit[0] & 1) == 0) /* - */;
       txmit[1] = c;
    }
```

22

Display Controller – MIPS Assembly

```
   .text
   .globl  write_ch      #  ARG0 = $a0 = character
write_ch:
   subu    $sp, $sp, 8   #  "push" stack
   sw      $t1, 4($sp)   #  save temporaries
   sw      $t0, 0($sp)   #  save temporaries
   li      $t0, 0xffff0008 # TCR address
write_tbe:
   lw      $t1, 0($t0)   # transmitter status port
   andi    $t1, $t1, 0x0001 # TDR is empty?
   beq     $t1, $zero, write_tbe # is READY "zero"
   sw      $a0, 4($t0)   # write data-byte to TDR
   lw      $t1, 4($sp)   # temporaries
   lw      $t0, 0($sp)   #  temporaries
   addiu   $sp, $sp, 8   # "pop" stack
   jr      $ra
```


Summary

- The CPU processes data from outside world via input and output operations
- Device drivers are abstraction that hides I/O device specifics
- *Where*: memory mapped or I/O ports
- *When*: CPU can either poll for data, or be interrupted when data are ready
- *How*: programmed I/O or DMA based
- Memory-mapped I/O eliminates the need for I/O specific instructions and I/O ports