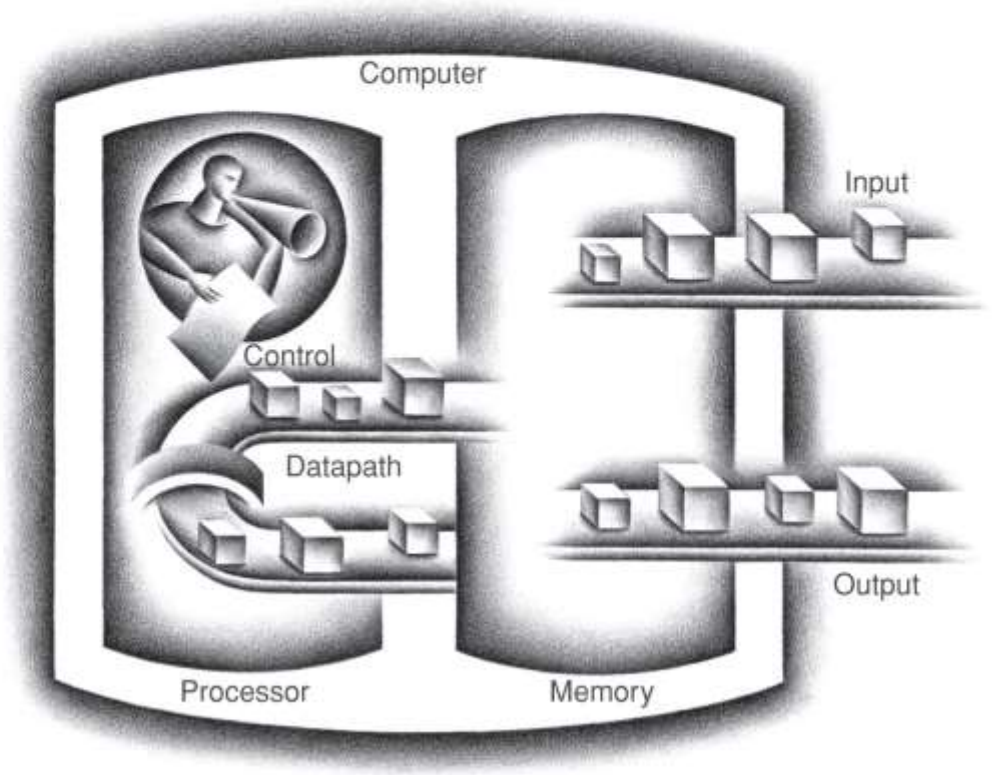
Chapter 7: I/O System

Ngo Lam Trung, Pham Ngoc Hung, Hoang Van Hiep

[with materials from *Computer Organization and Design*, MK and M.J. Irwin's presentation, PSU 2008]

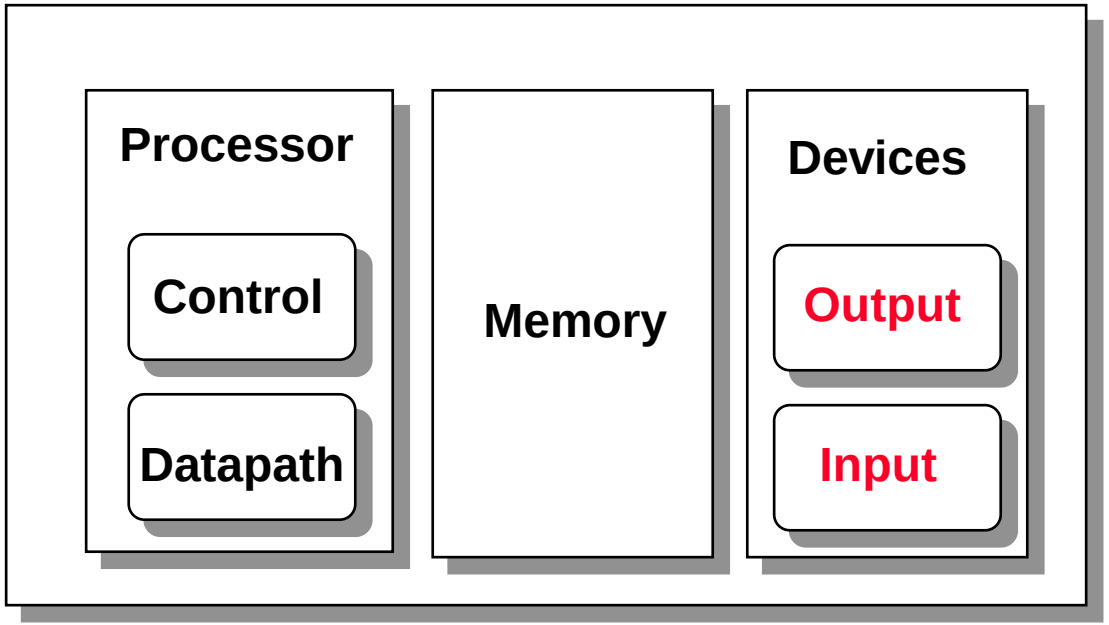
Computer Organization

- Computer needs the interface to communicate with the outside world: human, other machines, the physical environment.



Review: Major Components of a Computer

- Input + Output = I/O system
  - KVM, Network,...
  - Drives, USB devices,... *and thousands of other devices…*
  - Sensors,...
  - Controllers,...



Important metrics

- For processor and memory: performance and cost
- For I/O system: what are the most important?
  - Performance
  - Expandability
  - Dependability
  - Cost, size, weight
  - Security
  - …

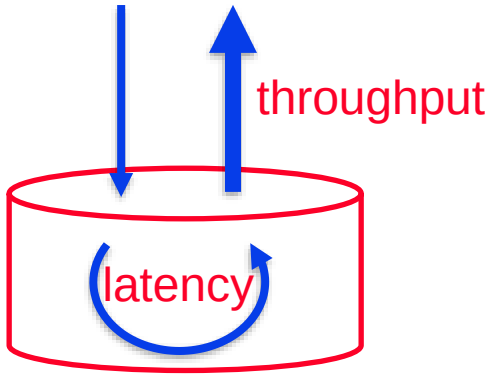
Input and Output Devices

- I/O devices are incredibly diverse with respect to
  - **Behavior** – input, output or storage
  - **Partner** – human or machine
  - **Data rate** – the peak rate at which data can be transferred between the I/O device and the main memory or processor

Device	Behavior	Partner	Data rate (Mb/s)
Keyboard	input	human	0.0001
Mouse	input	human	0.0038
Laser printer	output	human	3.2000
Magnetic disk	storage	machine	800.0000-3000.0000
Graphics display	output	human	800.0000-8000.0000
Network/LAN	input or output	machine	100.0000-10000.0000

I/O Performance Measures

- I/O bandwidth (throughput) – amount of information that can be input/output and communicated across an interconnect between the processor/memory and I/O device per unit time
  1. How much data can we move through the system in a certain time?
  2. How many I/O operations can we do per unit time?



- I/O response time (latency) – the total elapsed time to accomplish an input or output operation

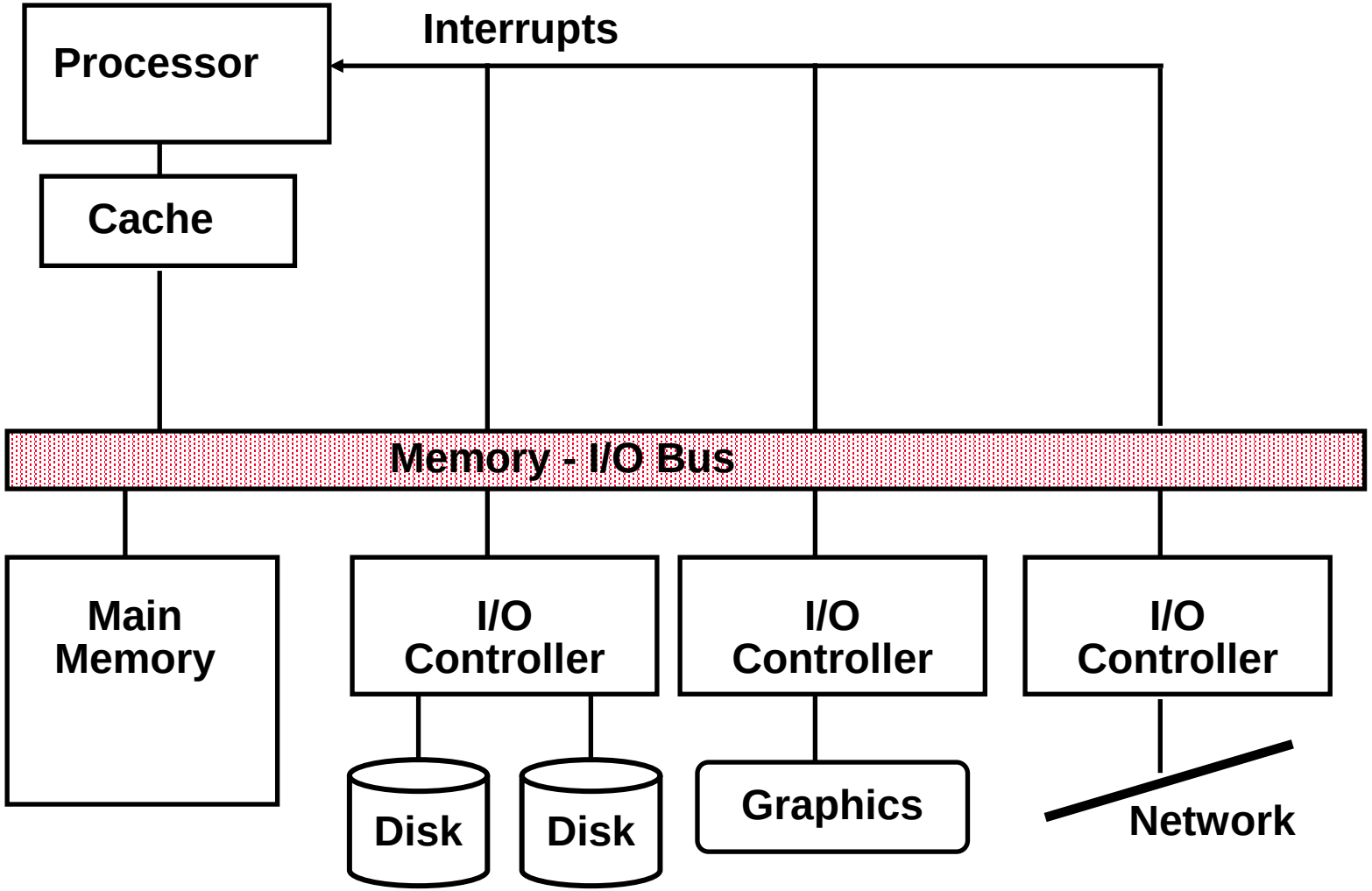
- Many applications require *both* high throughput and short response times

System Interconnection

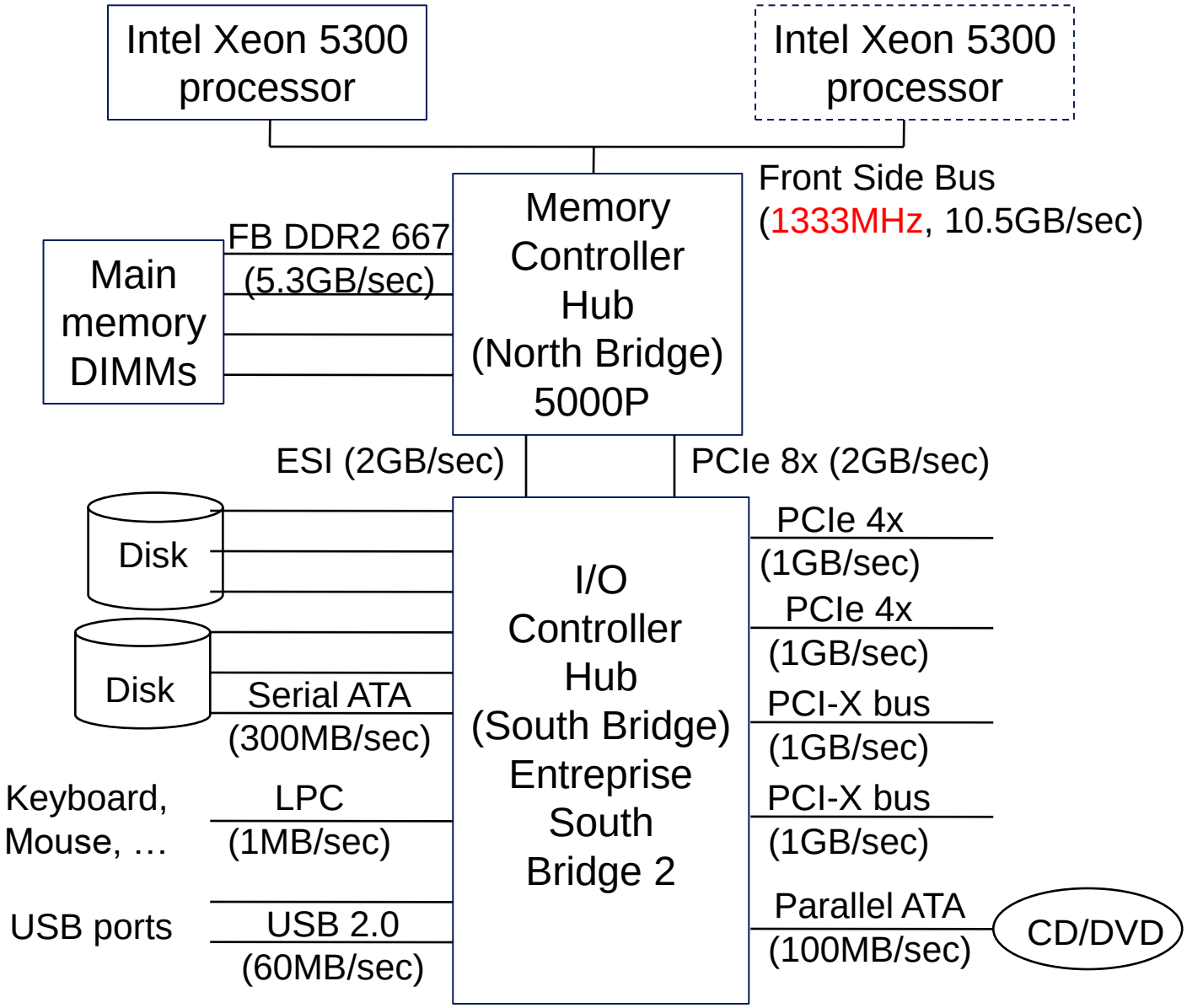
- Processor
- Memory
- I/O devices

- How to connect them physically?

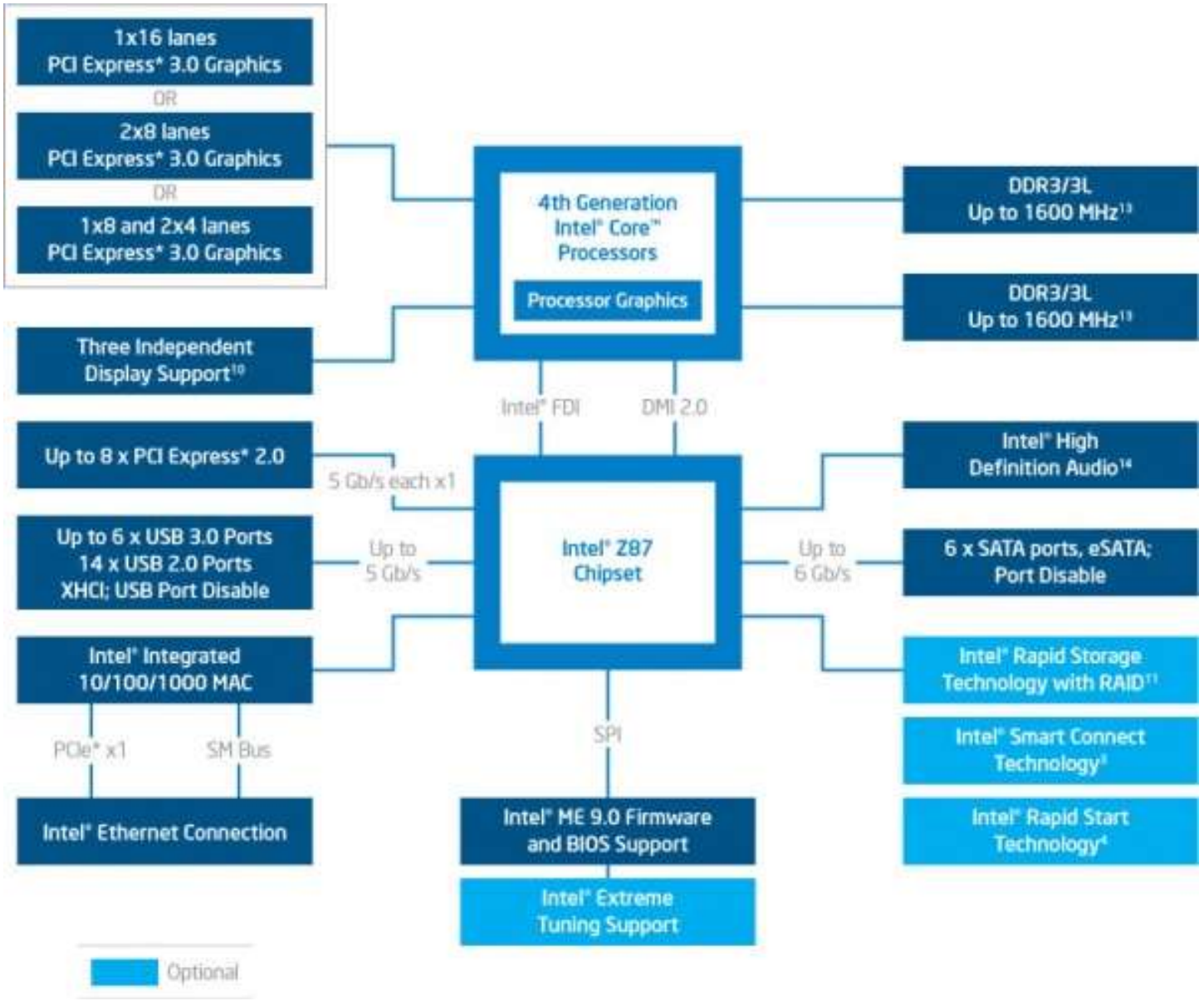
A Typical I/O System



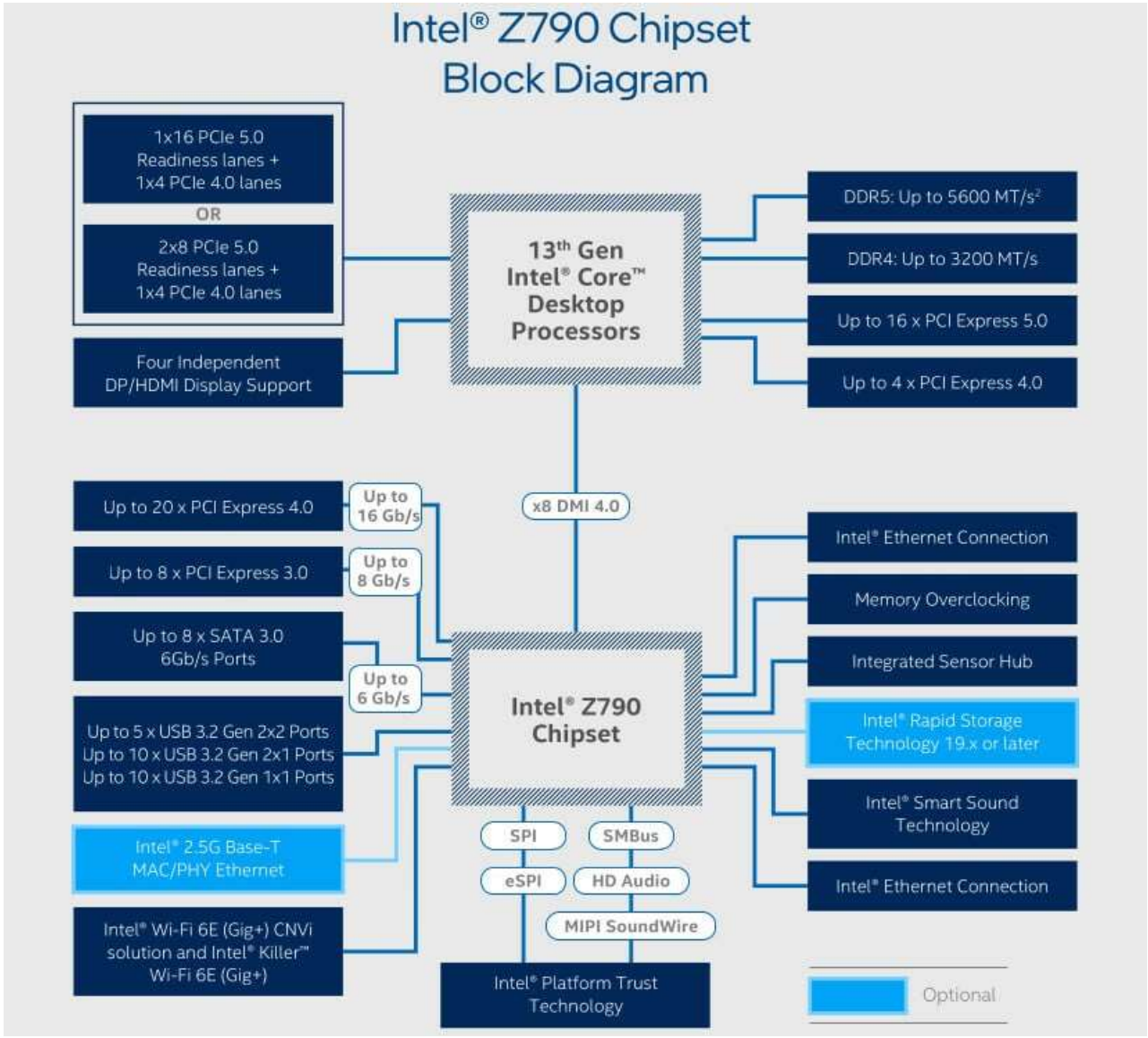
Intel Xeon 5300 I/O System (2007)



Intel Core i7 with Z87 chipset (2013)



Intel Core 13th gen with Z790 chipset (2022)



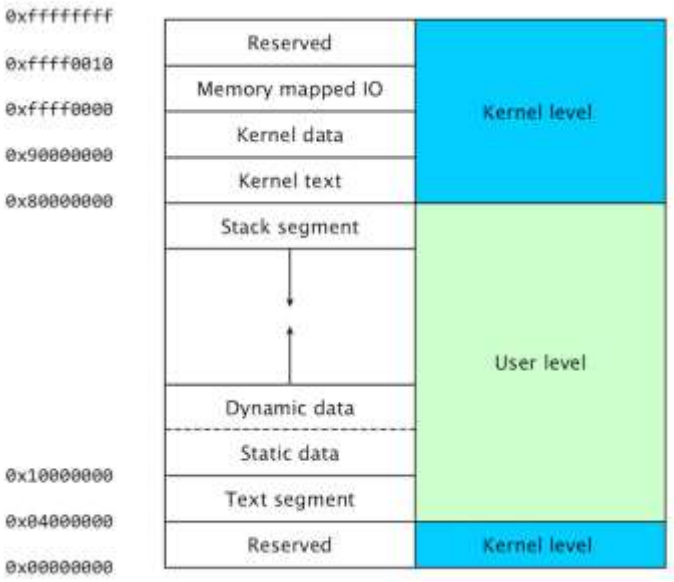
Interfacing with I/O Devices

- Physical connection is done, now how about data transfer?

- How is a user I/O request transformed into a device command and communicated to the device?
- How is data actually transferred to or from a memory location?
- What is the role of the operating system?

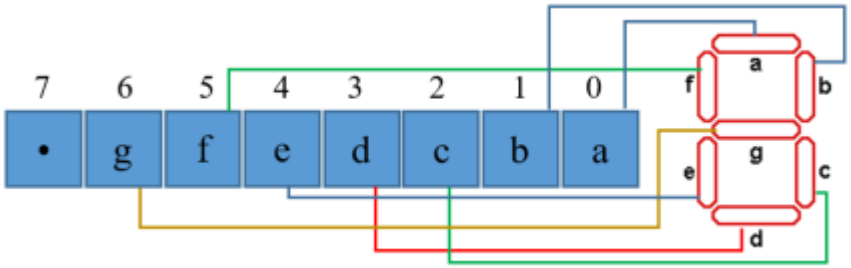
Communication of I/O Devices and Processor

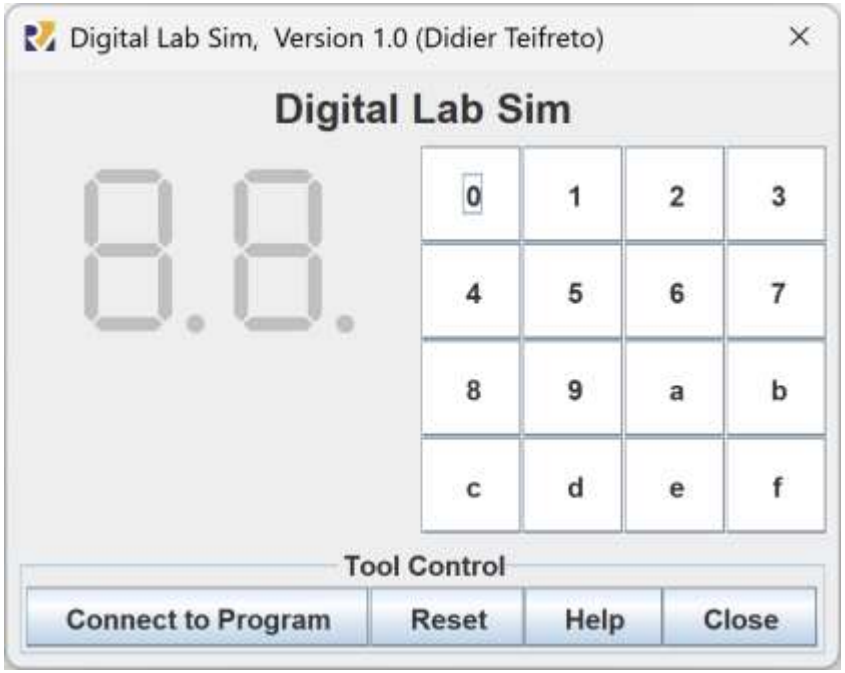
- How the processor directs (find) the I/O devices
  - Special I/O instructions
    - Must specify both the device and the command
  - Memory-mapped I/O
    - I/O devices are mapped to memory addresses
    - Read and writes to those memory addresses are interpreted as commands to the I/O devices
    - Load/stores to the I/O address space can *only* be done by the OS
- Memory map



Example: controlling 7-seg LED in RARS

- Tools→Digital Lab Sim: 2x 7-seg LEDs display
  - Byte value at address 0xFFFF0010 : command right seven segment display
  - Byte value at address 0xFFFF0011 : command left seven segment display

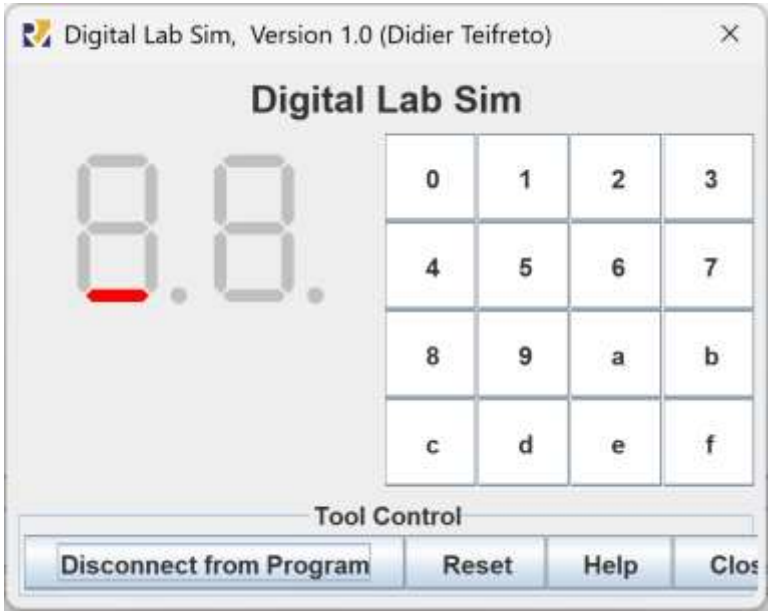




Example: controlling 7-seg LED in RARS

- Tools→Digital Lab Sim: 2x 7-seg LEDs display
  - Byte value at address 0xFFFF0010 : command right seven segment display
  - Byte value at address 0xFFFF0011 : command left seven segment display

```
li    a0,  0x8          #value
li    t0,  0xFFFF0011 #address
sb    a0,  0(t0)       #turn-on
```



Exercise

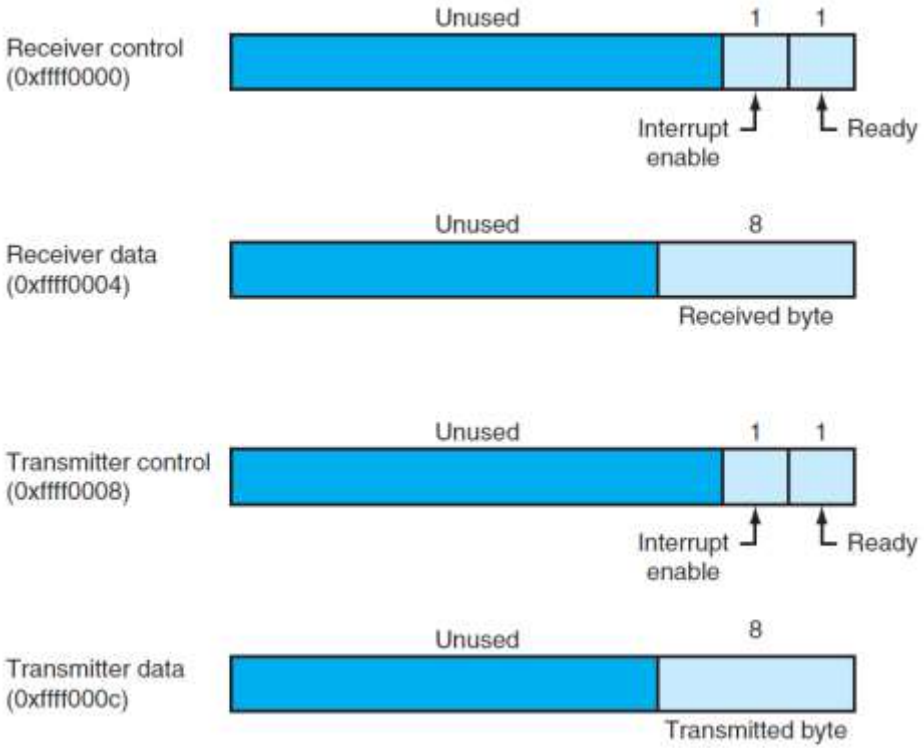
- Write program to display the value 24 to Digital Lab Sim

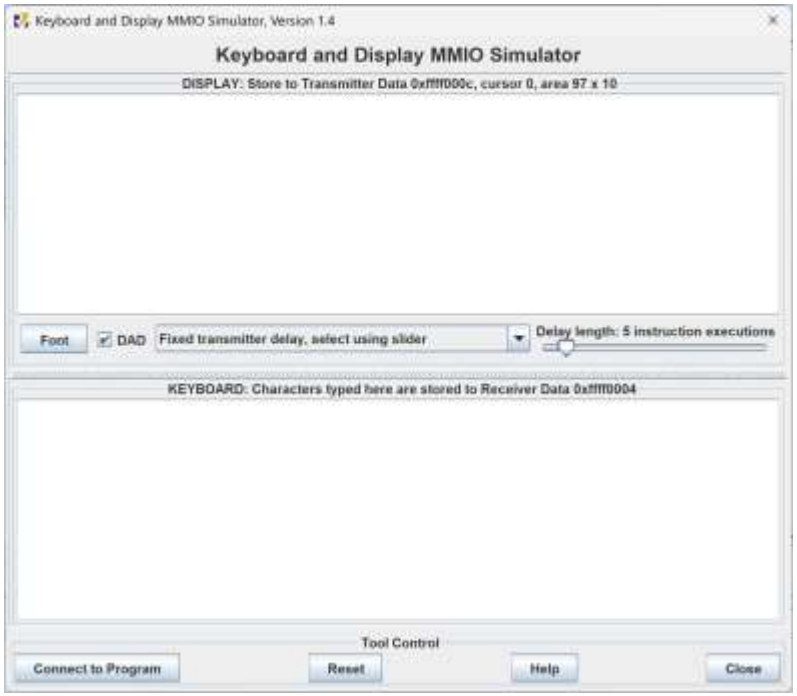
Communication of I/O Devices and Processor

- How I/O devices communicate with the processor
  - Polling
  - Interrupt driven I/O
  - Direct memory access
- Polling – the processor periodically checks the status of an I/O device to determine its need for service
  - Processor is totally in control – but does all the work
  - Can waste a lot of processor time due to speed differences

Example: polling the memory-mapped keyboard

- Terminal:
  - Input: receiver control (0xffff0000) and data (0xffff0004)
  - Output: transmitter control (0xffff0008) and data (0xffff000c)





Example: polling the memory-mapped keyboard

- Polling for data, then read when data is available

```
.eqv KEY_READY 0xFFFF0000
.eqv KEY_CODE  0xFFFF0004
.text
    li   s0,  KEY_CODE
    li   s1,  KEY_READY
WaitForKey:
    lw   t1, 0(s1) # check data available
    beq  t1, zero, WaitForKey
ReadKey:
    lw   t0, 0(s0)
    li   a7, 11 #print char
    mv a0, t0
    ecall
    j WaitForKey
```

Exercise

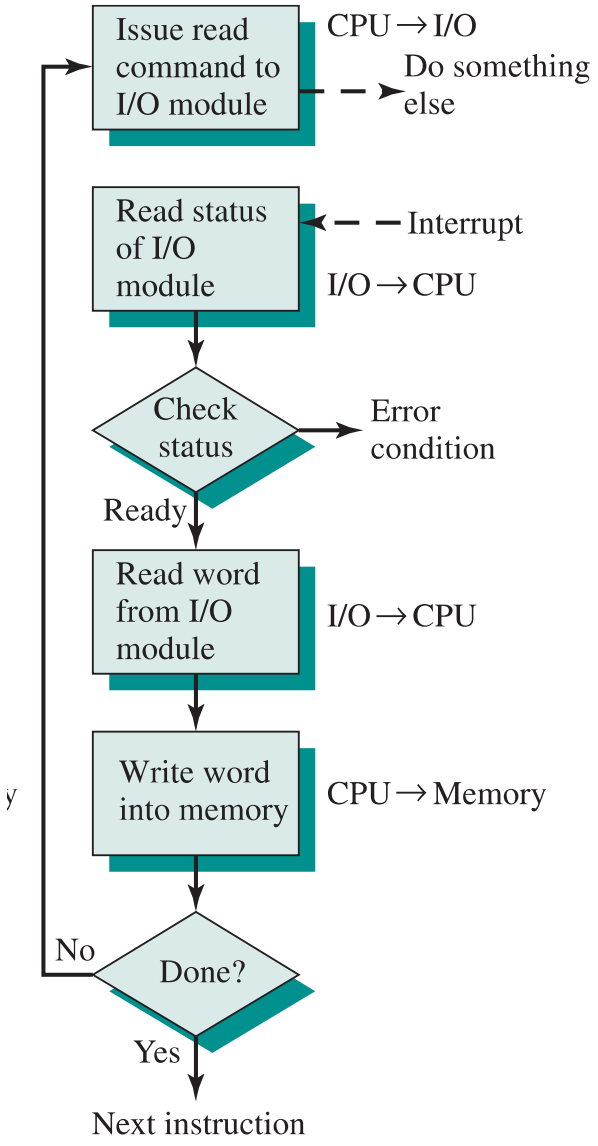
- Write a program to continuously read data from the terminal, encode the data by shifting it 3 position in the ASCII table, then write the encoded data to the terminal.
- Remember to check for terminal input and output ready before read/write.

```
.eqv KEY_CODE   0xFFFF0004
.eqv KEY_READY  0xFFFF0000
.eqv DISPLAY_CODE   0xFFFF000C
.eqv DISPLAY_READY  0xFFFF0008
.text
    li  a0, KEY_CODE
    li  a1, KEY_READY
    li  s0, DISPLAY_CODE
    li  s1, DISPLAY_READY
loop:
WaitForKey:
    lw      t1, 0(a1)               # t1 = [a1] = KEY_READY
    beq     t1, zero, WaitForKey    # Polling
ReadKey:
    lw      t0, 0(a0)               # t0 = [a0] = KEY_CODE

WaitForDis:
    lw      t2, 0(s1)               # t2 = [s1] = DISPLAY_READY
    beq     t2, zero, WaitForDis    # Polling
Encrypt:
    addi    t0, t0, 1               # change input key
ShowKey:
    sw      t0, 0(s0)               # show key
    j       loop
```

Interrupt driven I/O

- The I/O device issues an interrupt to indicate that it needs attention.
- The processor detects and “serves” the interrupt by executing a handler (aka. Interrupt service routine).



Interrupt Driven I/O

- Advantages of using interrupts
  - Relieves the processor from having to continuously poll for an I/O event;
  - User program progress is only suspended during the actual transfer of I/O data to/from user memory space
- Disadvantage – special hardware is needed to
  - Indicate the I/O device causing the interrupt and to save the necessary information prior to servicing the interrupt and to resume normal processing after servicing the interrupt

RISC-V Interrupt

- Control and Status Registers (CSRs): indicate the state of the CPU and allow software to control the behavior of the CPU.

- ustatus (status register)
- ucause (interrupt cause)
- utvec (trap vector)
- uie (interrupt enable)
- uip (interrupt pending)
- uepc (exception program counter)

Registers | Floating Point | Control and Status

Name	Number
ustatus	0
fflags	1
frm	2
fcsr	3
uie	4
utvec	5
uscratch	64
uepc	65
ucause	66
utval	67
uip	68
cycle	3072
time	3073
instret	3074
cycleh	3200
timeh	3201
instreth	3202

Interrupt handling

1. Declare the interrupt handling routine (Interrupt Service Routine - ISR)
2. Load the **interrupt handling routine** address into the **utvec** register.
3. Depending on the program, set the interrupt source in the **uie** register.
4. Enable global interrupts, set the uie bit of the **ustatus** register.
5. (For RARS simulator) Set up the simulation tool to enable interrupts (Keypad, Timer Tool, ...)

Interrupt handling

- When an interrupt occurred: RISC-V jumps to interrupt service routine
- Inside ISR
  - Classify the source of interrupt, depending on the interrupt source, perform the corresponding processing.
  - Exit from ISR with instruction **`uret`** (exception return). This basically restores PC with value in EPC.

Example: detect a keypad button pressed (1/3)

```
.eqv IN_ADDRESS_HEXA_KEYBOARD       0xFFFF0012
.data
    message: .asciz "Someone's presed a button.\n"
# MAIN Procedure
.text
main:
    # Load the interrupt service routine address to the UTVEC register
    la      t0, handler
    csrrs   zero, utvec, t0

    # Set the UEIE (User External Interrupt Enable) bit in UIE register
    li      t1, 0x100
    csrrs   zero, uie, t1       # uie - ueie bit (bit 8)

    # Set the UIE (User Interrupt Enable) bit in USTATUS register
    csrrsi  zero, ustatus, 0x1  # ustatus - enable uie (bit 0)
    # Enable the interrupt of keypad of Digital Lab Sim
    li      t1, IN_ADDRESS_HEXA_KEYBOARD
    li      t3, 0x80  # bit 7 = 1 to enable interrupt
    sb      t3, 0(t1)
```

Example: detect a keypad button pressed (2/3)

```
    # ------------------------------------------------------
    # Infinite main loop to demo the effective of interrupt
    # ------------------------------------------------------
loop:
    nop
    # Delay 10ms
    li      a7, 32
    li      a0, 10
    ecall
    nop
    j       loop
end_main:
```

Example: detect a keypad button pressed (3/3)

```
# Interrupt service routine
handler:
    # ebreak # Can pause the execution to observe registers
    # Saves the context
    addi    sp, sp, -8
    sw      a0, 0(sp)
    sw      a7, 4(sp)

    # Handles the interrupt: Shows message in Run I/O
    li      a7, 4
    la      a0, message
    ecall

    # Restores the context
    lw      a7, 4(sp)
    lw      a0, 0(sp)
    addi    sp, sp, 8

    # Back to the main procedure
    uret
```

Direct Memory Access (DMA)

- For high-bandwidth devices (like disks) interrupt-driven I/O would consume a *lot* of processor cycles
- With DMA, the DMA controller has the ability to transfer large blocks of data directly to/from the memory without involving the processor
  1. The processor initiates the DMA transfer by supplying the I/O device address, the operation to be performed, the memory address destination/source, the number of bytes to transfer
  2. The DMA controller manages the entire transfer (possibly thousand of bytes in length), arbitrating for the bus
  3. When the DMA transfer is complete, the DMA controller interrupts the processor to let it know that the transfer is complete
- There may be multiple DMA devices in one system
  - Processor and DMA controllers contend for bus cycles and for memory

Summary

- Characteristics of I/O system and devices
- I/O performance measures
- I/O system organization
- Methods for I/O operation and control
  - Polling
  - Interrupt
  - DMA