953214
Operating System and Computer Network

Chapter 5

Input/Output (Device Management)

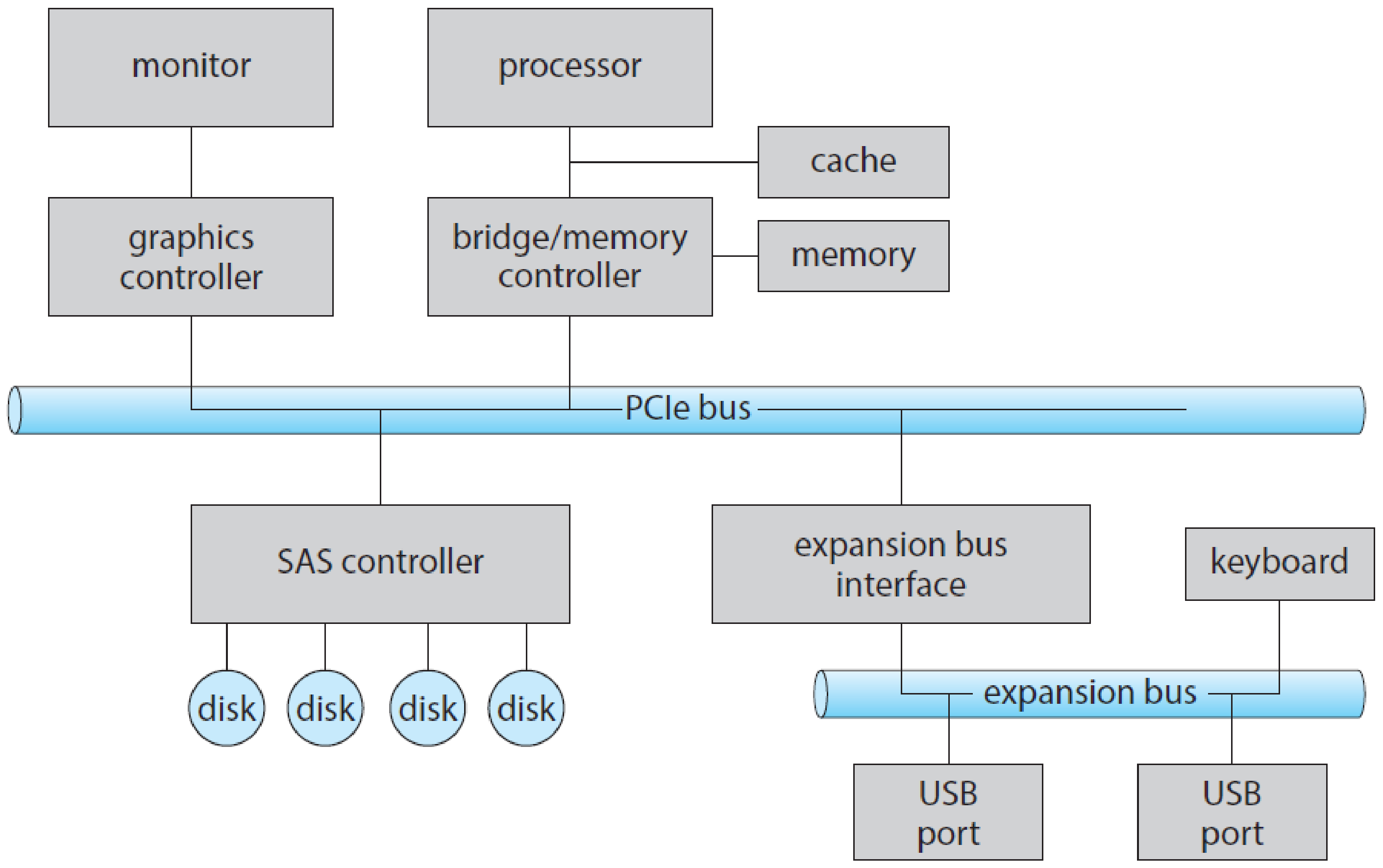
Lecturer: Dr. Phudinan Singkhamfu, Dr. Parinya Suwansrikham

CAMT
College of Arts, Media and Technology
Chiang Mai University

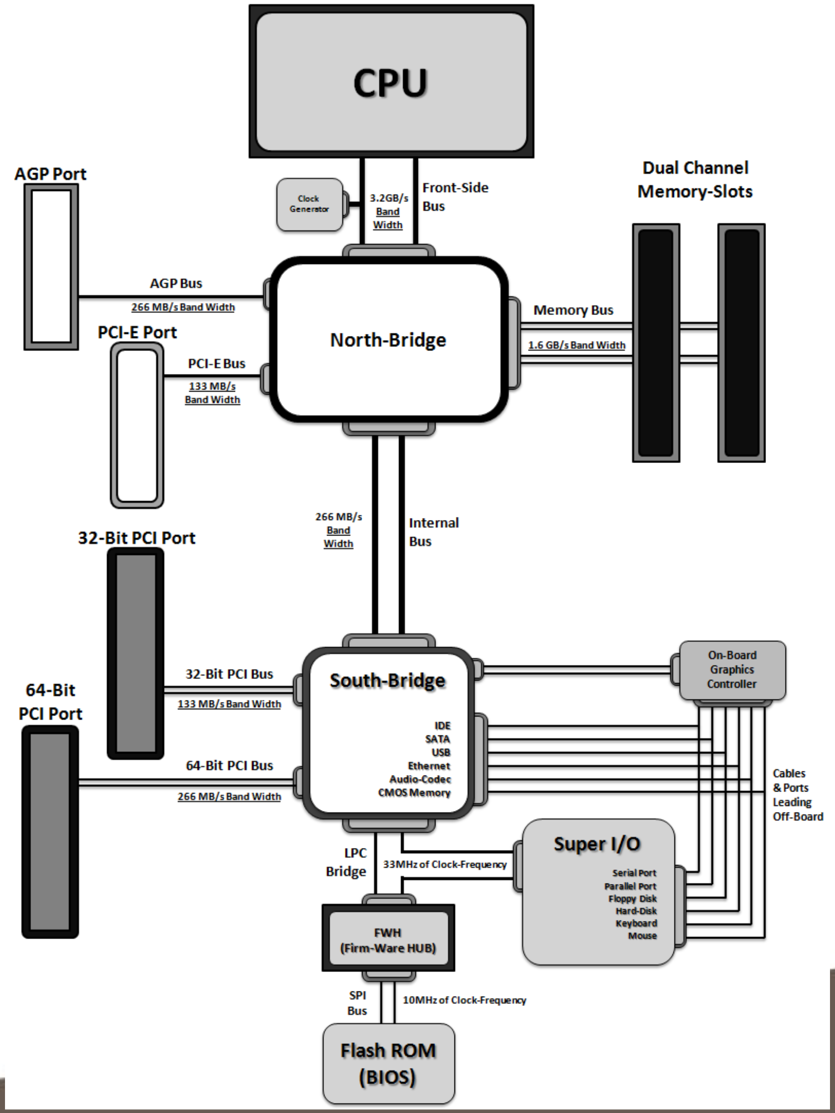
Architecture of I/O Systems

- Key components
  - **System bus**: allows the device to communicate with the CPU, typically shared by multiple devices.
  - A device **port** typically consisting of 4 registers:
    - **Status** indicates a device busy, data ready, or error condition
    - **Control**: command to perform
    - **Data-in:** data being sent from the device to the CPU
    - **Data-out:** data being sent from the CPU to the device
  - **Controller**: receives commands from the system bus, translates them into device actions, and reads/writes data onto the system bus.
  - The device itself
- Traditional devices: disk drive, printer, keyboard, modem, mouse, display
- Non-traditional devices: joystick, robot actuators, flying surfaces of an airplane, fuel injection system of a car, ...

PCI Bus Structure

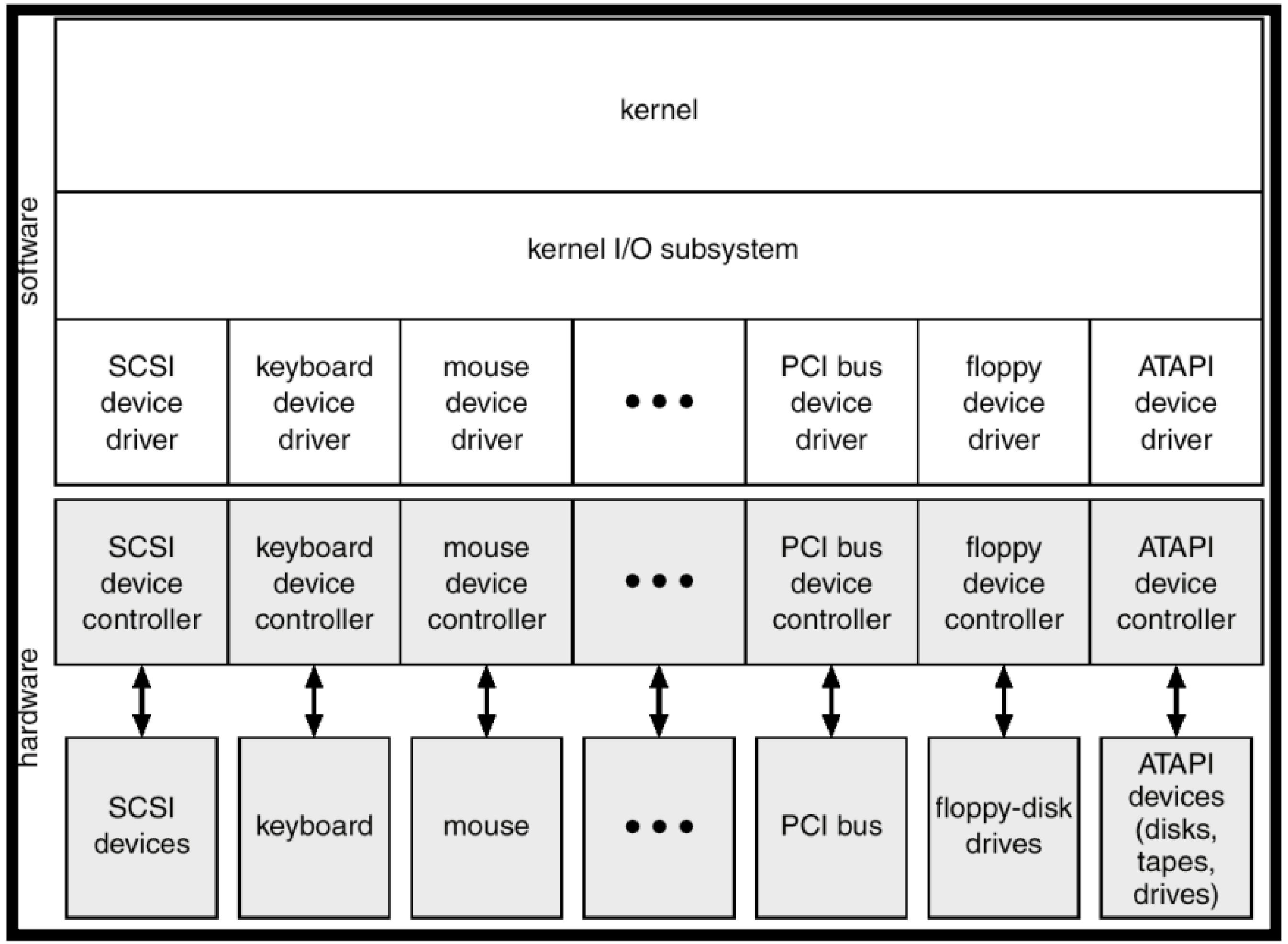


Chipset Architecture



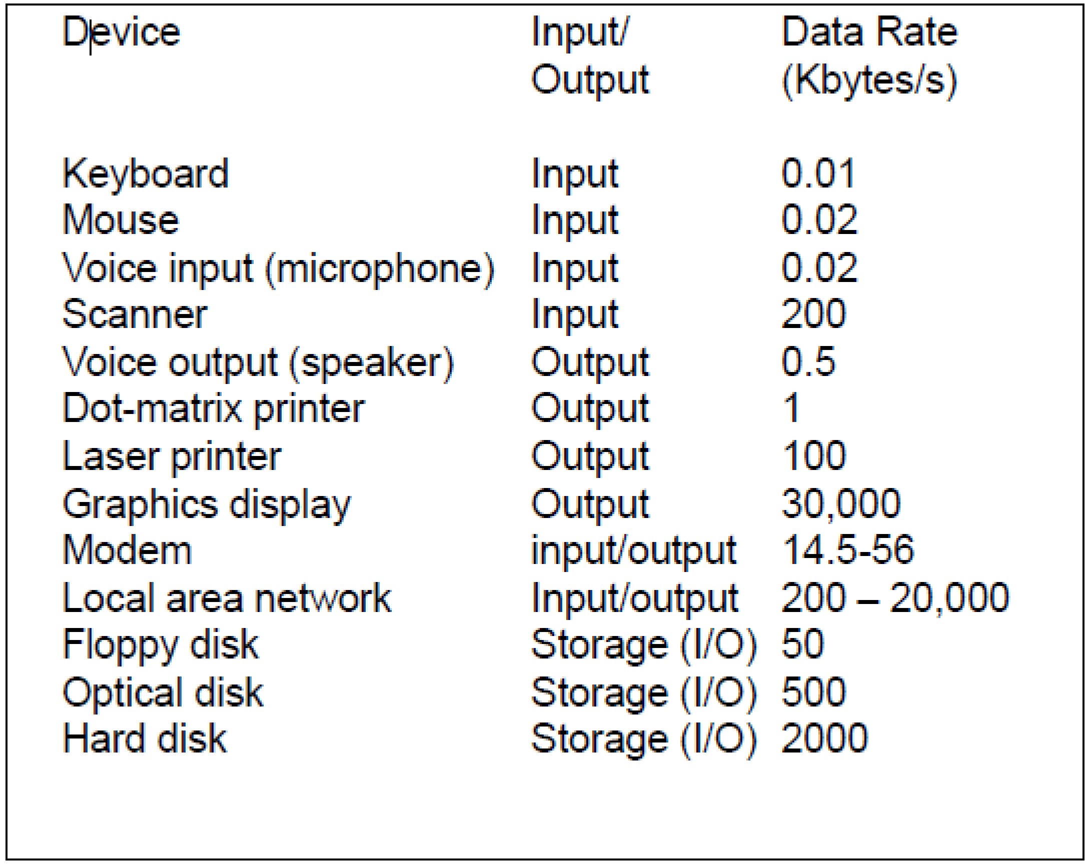
CAMT
College of Arts, Media and Technology
Chiang Mai University

Kernel I/O Subsystem



I/O Problems

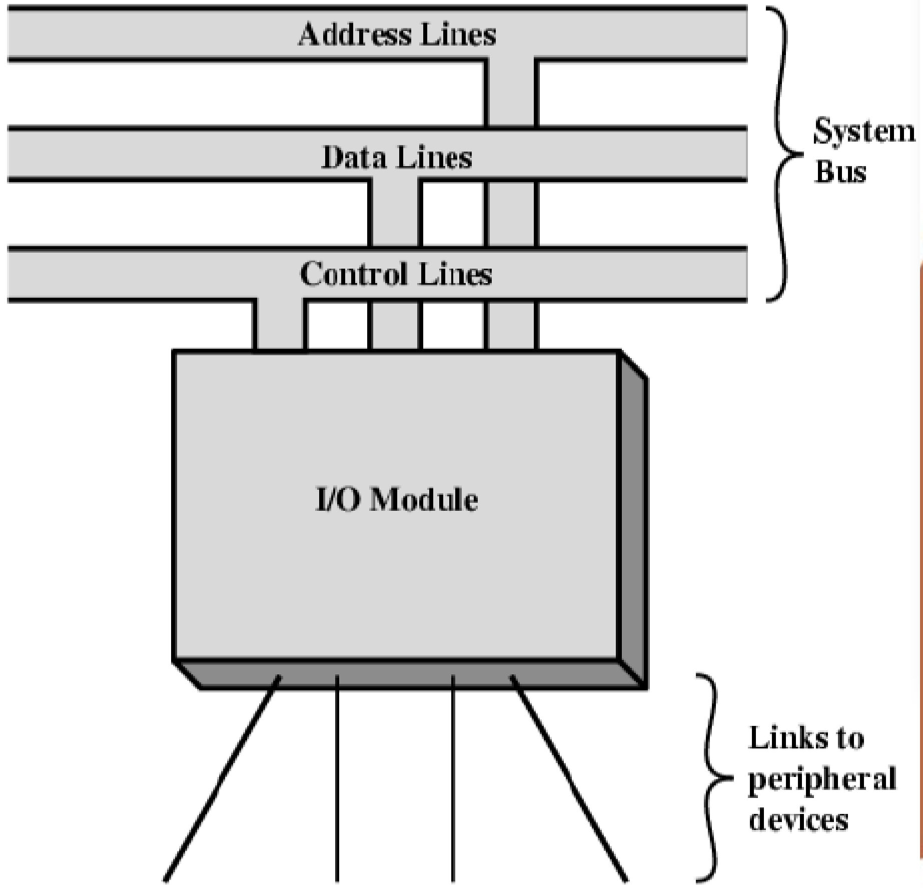
- All computer systems must have efficient means to receive input and deliver output
- Peripheral hardware is a shared resource, and not directly accessible by user programs
- Wide variety of peripherals
  - Delivering different amounts of data
  - At different speeds
  - In different formats
- All slower than CPU and RAM



Device	Input/ Output	Data Rate (Kbytes/s)
Keyboard	Input	0.01
Mouse	Input	0.02
Voice input (microphone)	Input	0.02
Scanner	Input	200
Voice output (speaker)	Output	0.5
Dot-matrix printer	Output	1
Laser printer	Output	100
Graphics display	Output	30,000
Modem	input/output	14.5-56
Local area network	Input/output	200 – 20,000
Floppy disk	Storage (I/O)	50
Optical disk	Storage (I/O)	500
Hard disk	Storage (I/O)	2000

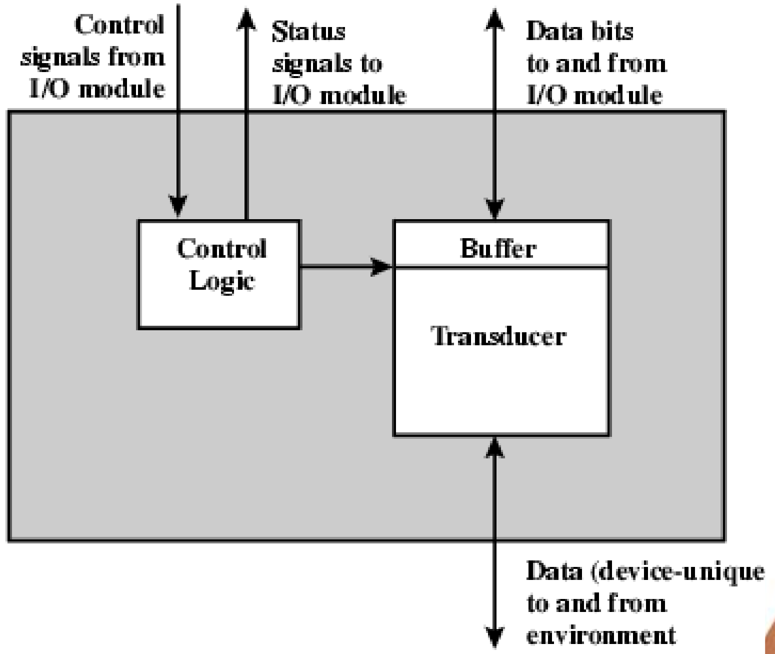
I/O Module (Controller)

- External devices are generally not connected directly to the bus structure of the computer systems
  - Wide variety of devices require different logic interfaces -impractical to expect the CPU to know how to control each device
  - Mismatch of data rates
  - Different data representation
- I/O module interfaces to CPU and Memory through the system bus and controls one or more peripheral devices
- I/O module is the third key element of a computer system



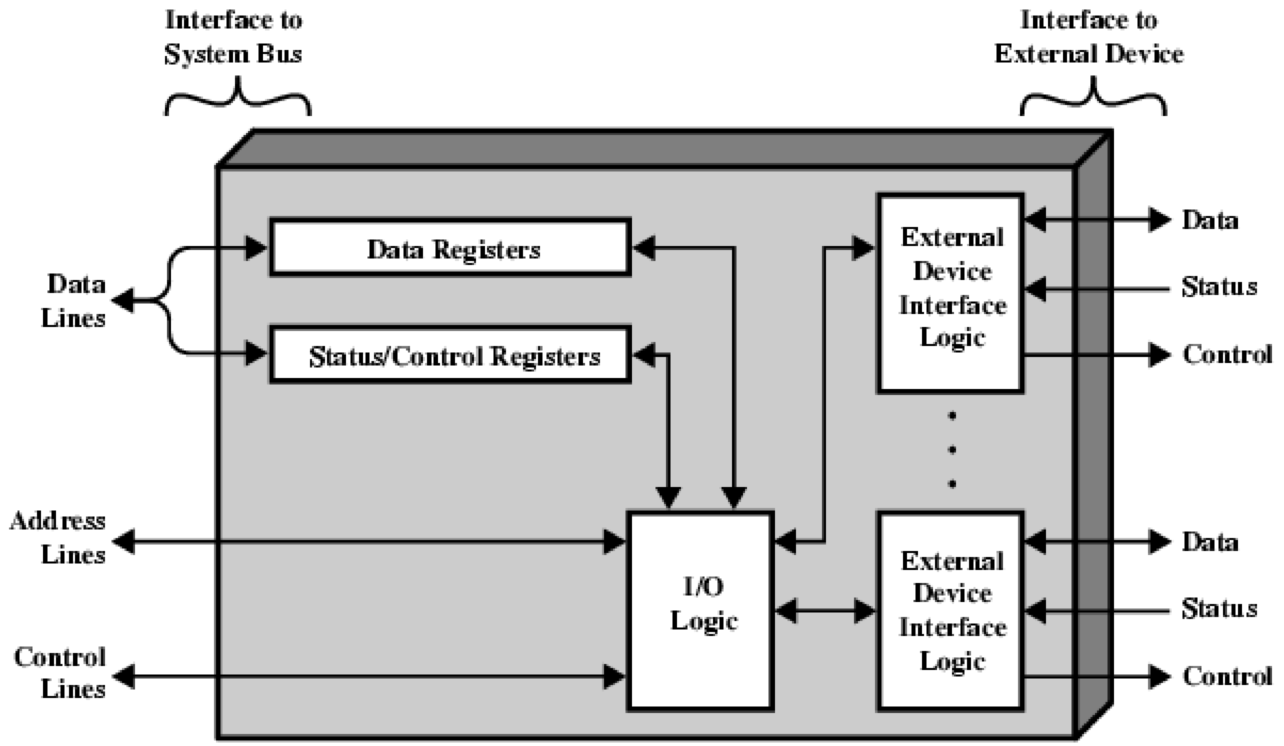
External Interface

- Interface to a peripheral from an I/O module
  - Data transmission format
  - Parallel transmission (parallel interface)
- expensive but fast, suitable for local
  - Serial transmission (serial interface)
  - cheaper but slower, suitable for long distance


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

I/O Module Tasks

- Functions
  - Control & Timing
  - CPU and Device Communication
  - Data Buffering
  - Error Detection CPU checks I/O module device status
- Decisions
  - Hide or reveal device properties to CPU
  - Support multiple or single device
  - Control device functions or leave for CPU
- Steps
  - I/O module returns status
  - If ready, CPU requests data transfer
  - I/O module gets data from device
  - I/O module transfers data to CPU



I/O Module Features

- Not just simple mechanical connectors
- Contain “intelligence” -logic for performing communication functions between peripherals and bus
- Provide standard interfaces to the CPU and the bus
- Tailored to specific I/O devices and their interfaces requirement
- Relieve CPU of the management of I/O devices
- I/O Techniques:
  - Programmed I/O (polling),
  - Interrupt driven I/O,
  - Direct memory access (DMA)

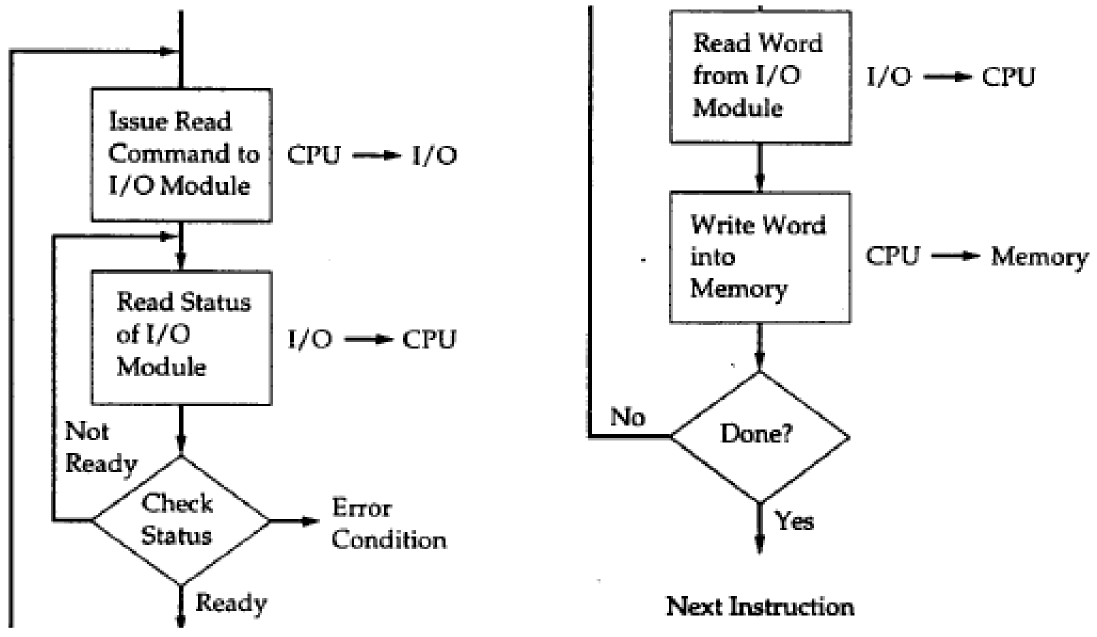
Programmed I/O (Polling)

- For each byte of I/O
  - Read busy bit from status register until 0
  - Host sets read or write bit and if write copies data into data-out register
  - Host sets command-ready bit
  - Controller sets busy bit, executes transfer
  - Controller clears busy bit, error bit, command-ready bit when transfer done
- Step 1 is busy-wait cycle to wait for I/O from device
  - Reasonable if device is fast
  - But inefficient if device slow
  - CPU switches to other tasks?
    - But if miss a cycle data overwritten/lost

Programmed I/O (Polling)

- I/O operation in which the CPU issues the I/O command to the I/O module
- CPU is in direct control of the operation
    - Sensing status, Read/write commands, Transferring data
- CPU waits until the I/O operation is completed before it can perform other tasks
- Completion indicated by a change in the status bits
- CPU must periodically poll the module to check its status bits
- The speed of the CPU and peripherals can differ by orders of magnitude, programmed I/O waste huge amount of CPU power
- Very inefficient -CPU slowed due to the speed of peripherals
- Simple

Programmed I/O (Polling)



Programmed I/O Operation

- Uses registers for data transfers, Control/command and Status
- Device status is checked to determine if an I/O operation is needed
- CPU controls the data transfer by executing a polling loop
- Simple to implement -Requires very little special software or hardware
- Useful when there are a lot of similar devices connected to one system

Addressing I/O Devices

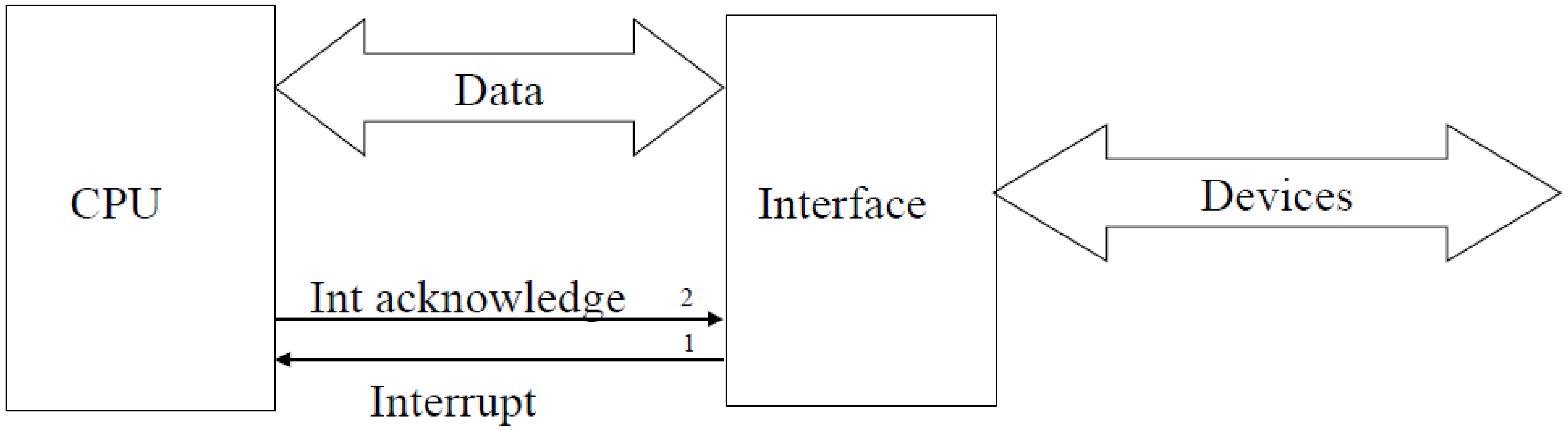
- Under programmed I/O data transfer is very like memory access (CPU viewpoint)
- Each device given unique identifier
- CPU commands contain identifier (address)
- I/O addressing techniques:
  - Memory mapped I/O: memory and I/O reside in the same “space” and use same memory instructions
  - Isolated (I/O mapped) I/O: Memory and I/O occupy different “spaces” and are accessed by unique i/o instructions

Interrupts

- Polling can happen in 3 instruction cycles
  - Read status, logical-and to extract status bit, branch if not zero
  - How to be more efficient if non-zero infrequently?
- CPU Interrupt-request line triggered by I/O device
  - Checked by processor after each instruction
- Interrupt handler receives interrupts
  - Maskable to ignore or delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
  - Context switch at start and end
  - Based on priority
  - Some nonmaskable
  - Interrupt chaining if more than one device at same interrupt number

Interrupts

- Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
  - Program -e.g. overflow, division by zero
  - Timer -Generated by internal processor timer
  - I/O -from I/O controller
  - Hardware failure -e.g. memory parity error



Servicing an Interrupt

- Servicing an Interrupt
  - When an interrupt occurs (and is accepted), the execution of the current program is suspended
- A special routine executes to service the interrupt
- Then, the interrupted program resumes
- The service routine is called an *interrupt handle err*or *interrupt service routine (ISR)*

CAMT

College of Arts, Media and Technology
Chiang Mai University

Use of Interrupts

- As an external event notifier
  - Interrupt when an “event” occurs in a device –an event that requires the CPU’s attention, e.g., a key has been hit (the ISR reads the code for the key)
- As a completion signal
  - Interrupt when a device has completed an operation –and the CPU should know about it, e.g., the output buffer for printer is empty (the CPU can send more data)
- As a means of allocating CPU time
  - Interrupt to execute another program -useful on multi-tasking systems to execute more than one program at a time, e.g., a timer is programmed to interrupt the CPU every 100 µs
- As an abnormal event indicator
  - Interrupt when an abnormal event occurs that needs immediate system attention, e.g., a heat sensor near the CPU chip generated an interrupt

Saving Registers

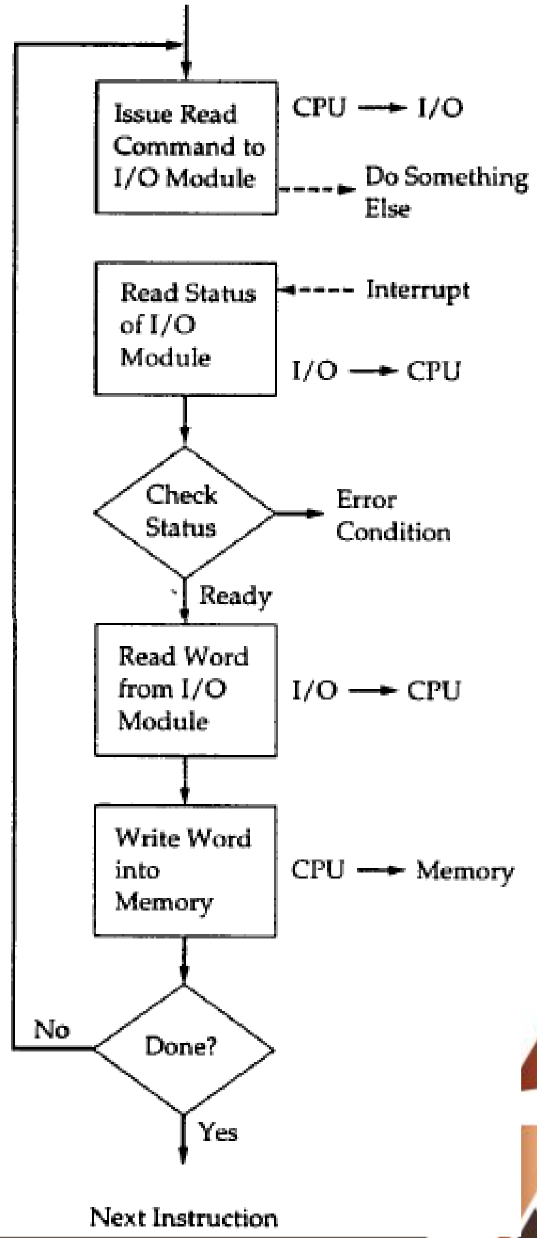
- For the interrupted program to resume, the CPU status and data registers must be saved (because they will change during the ISR)
  - They are saved before the ISR executes
  - They are restored after the ISR executes
- They are saved either
  - On the *stack* (a special area of memory to temporarily hold information), or
  - In a *process control block (PCB)*

Interrupt Driven I/O

- I/O operations are initiated by the device
- The device, or its I/O module, includes a signal to interrupt the CPU though *interrupt lines*
  - A typical CPU supports 8 to 16 interrupt inputs
- Reduce the time spent on I/O operation
- A wide variety devices use interrupt for I/O
- Basic Operation
  - CPU issues read command
  - CPU continues with its other tasks while the module performs its task
  - I/O module gets data from peripheral
  - I/O module interrupts CPU when the I/O operation is finished
  - CPU requests data by executing an interrupt service routine
  - I/O module transfers data

Interrupt Driven I/O

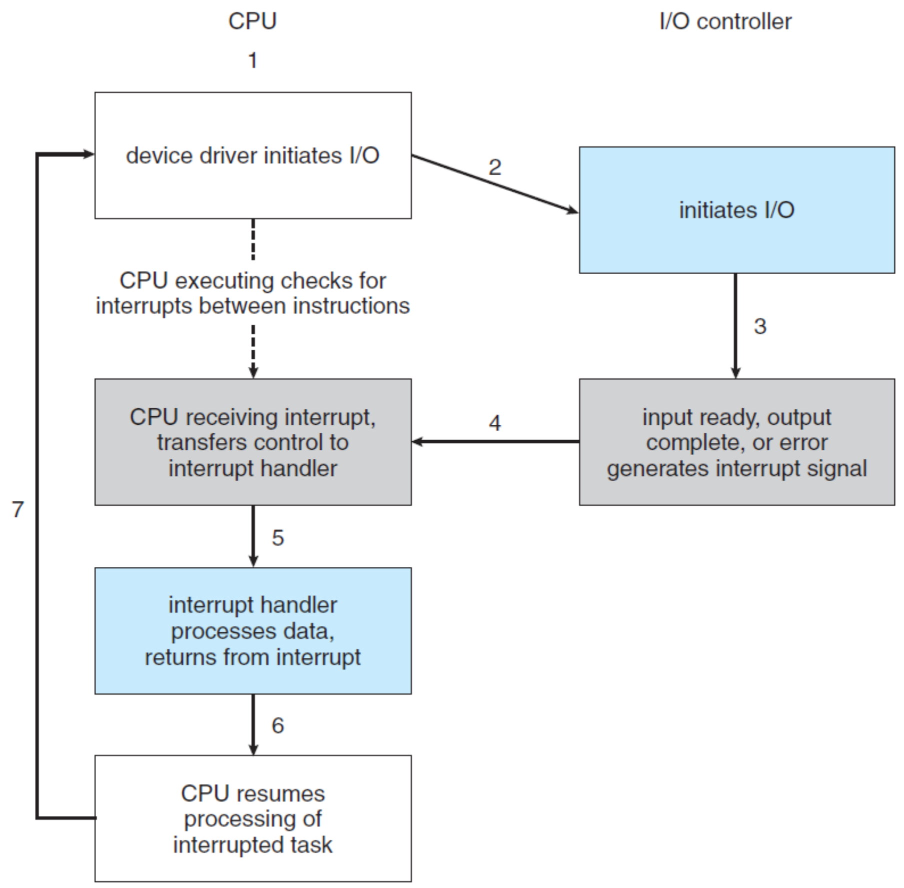
- CPU viewpoint
  - Issue read command
  - Do other work
  - Check for interrupt at end of each instruction cycle
  - If interrupted
    - Save context (registers)
    - Process interrupt -Fetch data & store
- Advantages
  - Overcomes CPU waiting
  - No repeated CPU checking of device
  - I/O module interrupts when ready



Interrupt Cycle

- Processor checks for interrupt
  - Indicated by an interrupt signal
- If no interrupt, fetch next instruction
- If interrupt pending:
  - Suspend execution of current program
  - Save context
  - Set PC to start address of interrupt handler routine
  - Process interrupt
  - Restore context and continue interrupted program

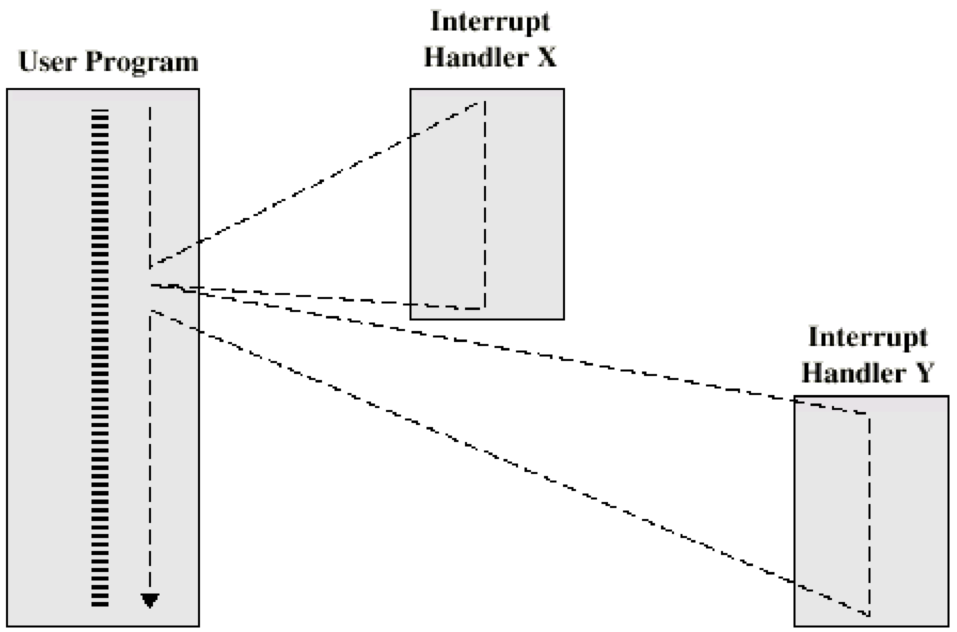
Interrupt-driven I/O Cycle

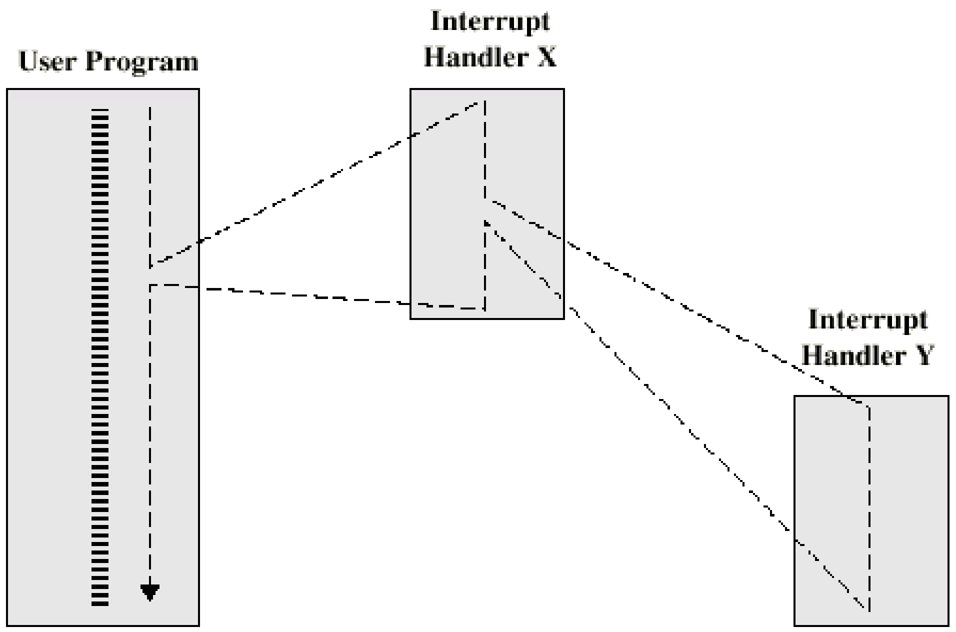

CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task
7

Multiple Interrupts

- Disable interrupts
  - Processor will ignore further interrupts whilst processing one interrupt
  - Interrupts remain pending and are checked after first interrupt has been processed
  - Interrupts handled in sequence as they occur
- Define priorities
  - Each interrupt line has a priority
  - Low priority interrupts can be interrupted by higher priority interrupts
  - When higher priority interrupt has been processed, processor returns to previous interrupt

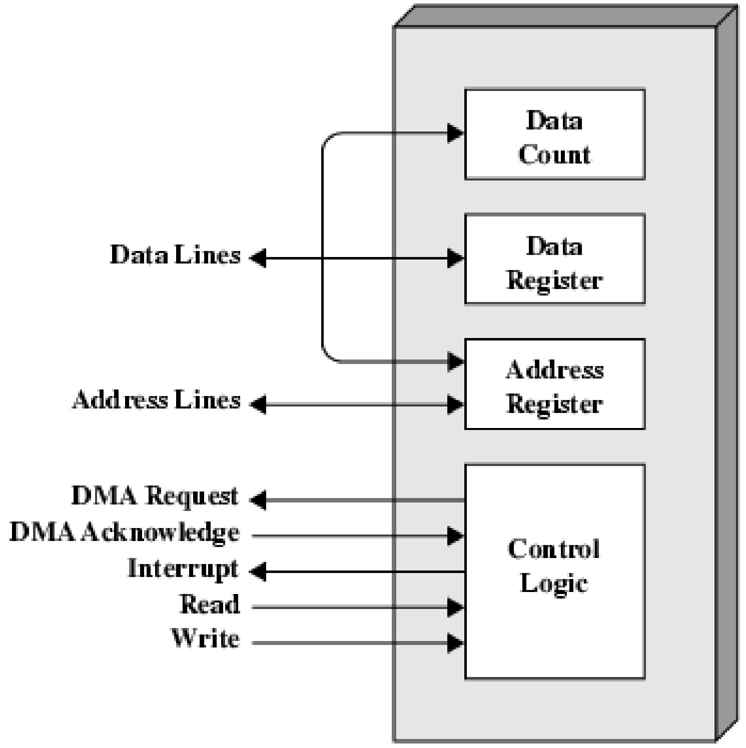
Multiple Interrupts





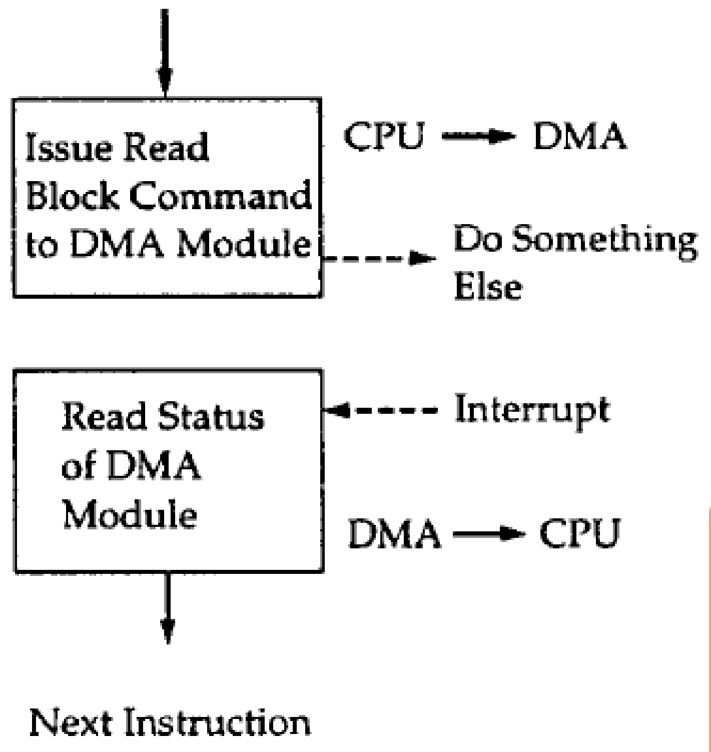
Direct Memory Access (DMA)

- Both *Polling* and *Interrupt* I/O control modes involve CPU intervention that (a) takes CPU's time, (b) slows data transfer speed
- Direct Memory Access (DMA)
  - Transfer data directly between memory and I/O devices without the invention of CPU
  - CPU initiates a DMA process (by telling a starting memory address and an I/O device) and then gives up the control right over the bus to the DMA controller
  - Fast but very complex

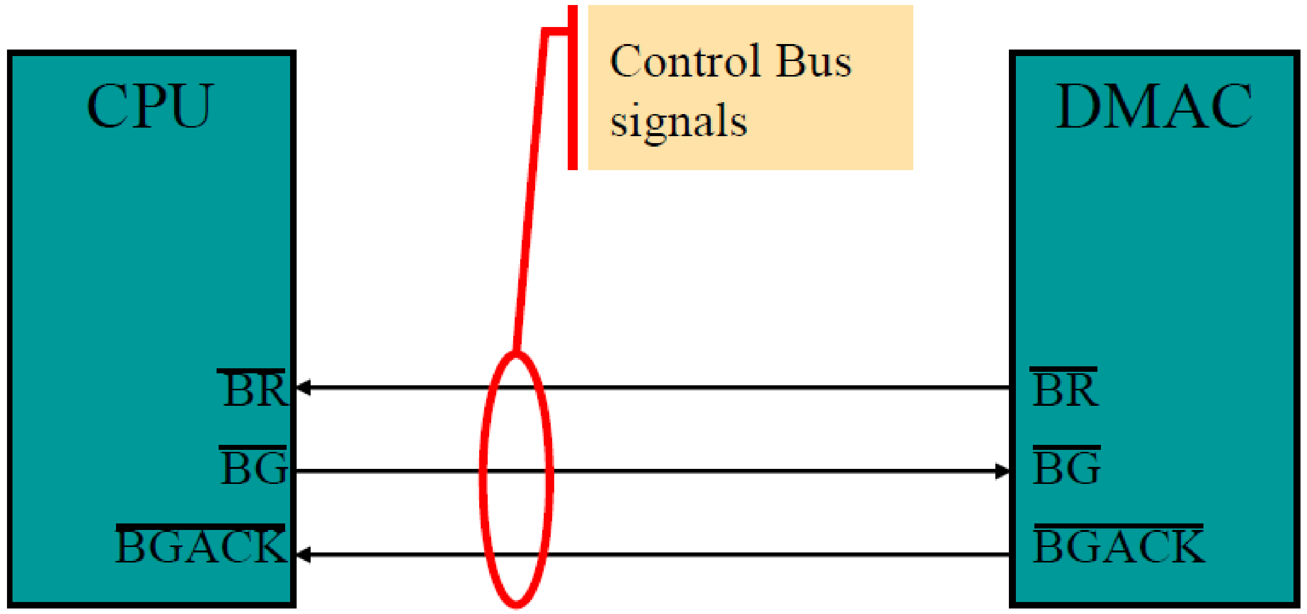


DMA Operation

- CPU tells DMA controller
  - Read/Write
  - Device address
  - Starting address of memory block for data
  - Amount of data to be transferred
- CPU carries on with other work
- DMA controller deals with transfer
- DMA controller sends interrupt when finished
- I/O module shares the bus with the CPU
  - DMA uses the bus when the CPU is not using it
  - Or it can steal cycles from the CPU by forcing it to free the bus
- Large amount of data can be transferred without severely impacting CPU performance


Issue Read Block Command to DMA Module
CPU → DMA
Do Something Else
Read Status of DMA Module
Interrupt
DMA → CPU
Next Instruction

Taking Control



- BR = Bus request (DMAC: May I take control of the system buses?)
- BG = Bus grant (CPU: Yes, here you go.)
- BGACK = BG acknowledge (DMAC: Thanks, I've got control.)

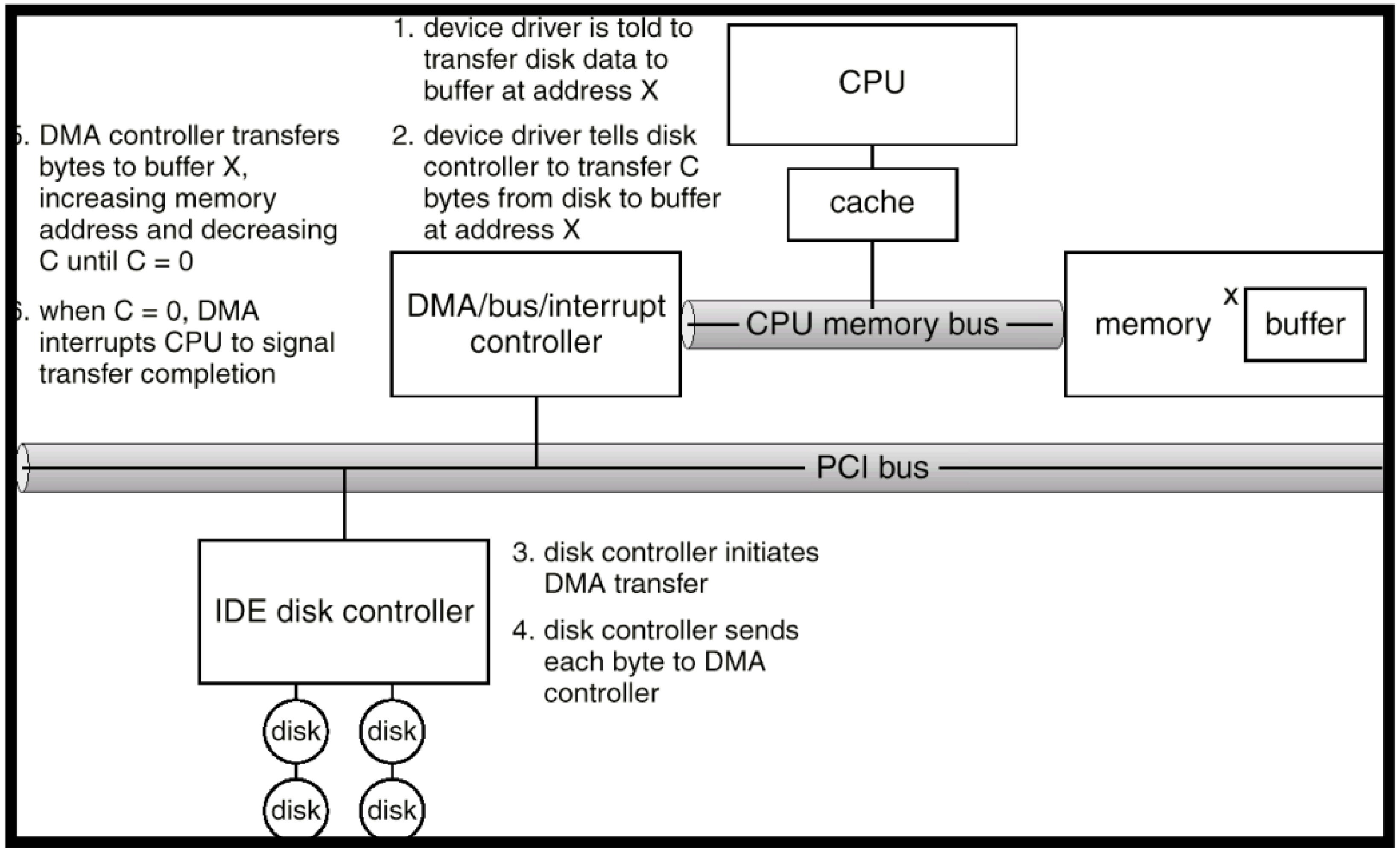
Taking Control

- DMAC issues a “bus request” signal
- CPU halts (perhaps in the middle of an instruction!) and issues a “bus grant” signal
- DMA can interrupt the processor in the middle of an instruction (before fetch, after decoding, after execution)
- DMAC issues “bus grant acknowledge” and releases BR
- DMAC has control of the system buses
- DMAC “acts like the CPU” and generates the bus signals (e.g., address, control) for one transfer to take place

DMA Transfer / Cycle Stealing

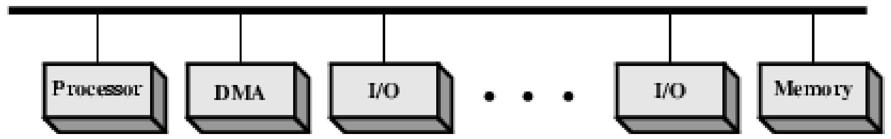
- DMA controller takes over bus for a cycle
- Transfer of one word of data
- Not an interrupt
  - CPU does not switch context
- CPU suspended just before it accesses bus
  - i.e. before an operand or data fetch or a data write
- Slows down CPU but not as much as CPU doing transfer

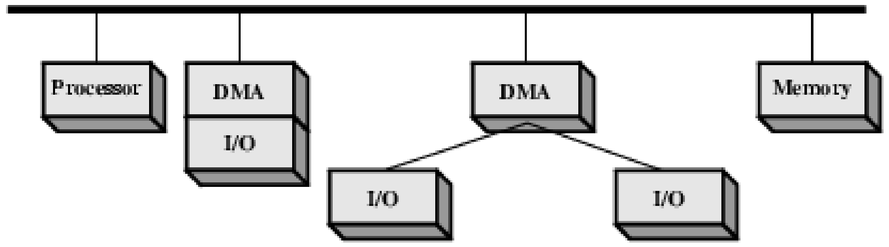
Steps in DMA Transfer


1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
. when C = 0, DMA interrupts CPU to signal transfer completion
CPU
cache
CPU memory bus
memory
x
buffer
DMA/bus/interrupt controller
PCI bus
IDE disk controller
disk
disk
disk
disk

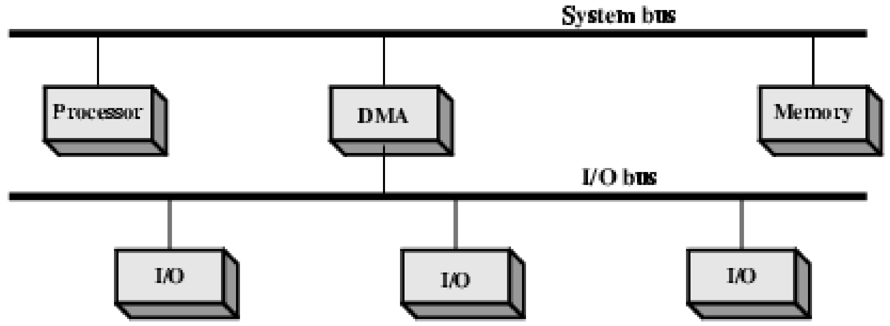
DMA Configurations

- Single Bus, Detached DMA controller
  - Each transfer uses bus twice -I/O to DMA then DMA to memory
  - CPU is suspended twice
- Single Bus, Integrated DMA controller
  - Controller may support >1 device
  - Each transfer uses bus once -DMA to memory
  - CPU is suspended once
- Separate I/O Bus
  - Bus supports all DMA enabled devices
  - Each transfer uses bus once -DMA to memory
  - CPU is suspended once





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



(c) I/O bus

I/O Bus Architecture

- Used in (pretty well all) PCs, workstations, and some mainframe computers
- We have already met the data, address, and control buses that connect a CPU to memory and I/O modules
- Collectively, these are the “CPU bus” or “system bus”
- Between the I/O modules and I/O devices, an “I/O bus” is required
- A “bus interface” connects one bus to another

I/O Channel Architecture

- I/O devices getting more sophisticated, e.g. 3D graphics cards
- Channel Architecture -an alternative I/O architecture
- I/O occurs through an “I/O processor” –the “channel subsystem”
  - Frees the CPU for other tasks
  - Has its own instruction set –“channel control words”
  - As ‘programs’, just like other CPU instructions
  - Transfers data between I/O devices and memory via DMA
- CPU instructs I/O controller to do transfer and I/O controller does entire transfer
- Improves speed
  - Takes load off CPU
  - Dedicated processor is faster

Evolution in I/O Organization

1. Processor controls device directly
2. A controller or I/O module is used separating the processor from the details of the I/O device
3. As in 2 but using interrupts, the burden of supervising the device continuously disappears
4. I/O module gains access to main memory through DMA moving data in and out memory without processor attention
5. I/O module becomes a processor -I/O specific instruction set –to be programmed by the processor in main memory
6. I/O module becomes a computer –a “channel”

Application I/O Interface

- I/O System calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers form kernel
- New devices talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks
- Devices vary in many dimensions
  - Character-stream or block
  - Sequential or random-access
  - Synchronous or asynchronous (or both)
  - Sharable or dedicated
  - Speed of operation
  - Read-write, read only, or write only

Summary

- I/O devices need interface to synchronize data transfer with CPU and memory, or convert data to appropriate format.
- I/O Control modes:
  - Programmed I/O (polling): continuous attention of the processor is required
  - Interrupts: processor launches I/O and can continue until interrupted
  - DMA: direst exchange of data between the I/O unit and the main memory
- Channels: Dedicated ‘computer’ for I/O module to handle data transfer
- Data transfer format: Parallel and Serial