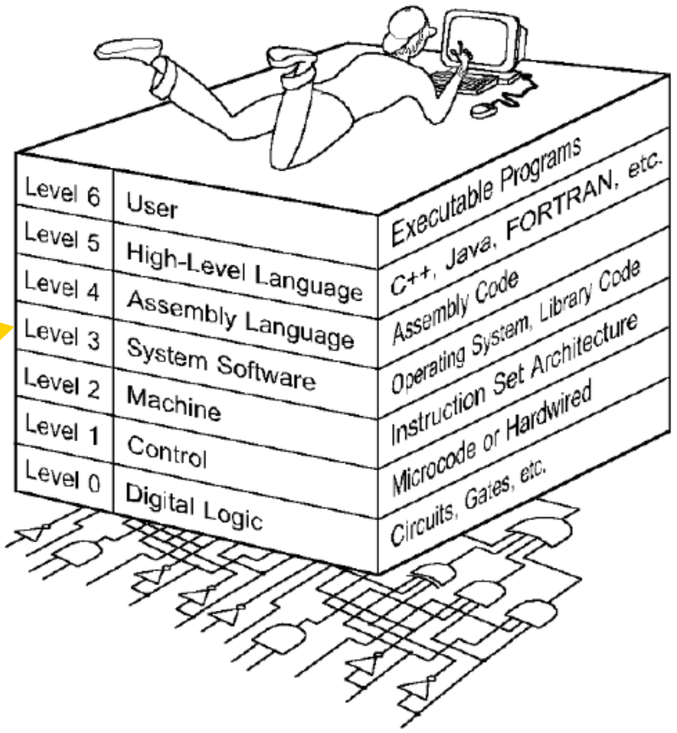
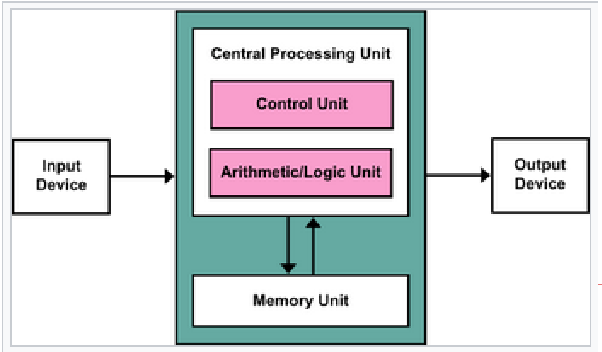


Chapter 7–Input/Output and Storage Systems

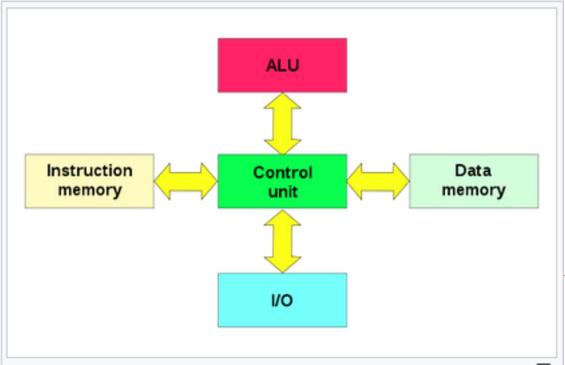
Provides inter-**link** between **CPU** (containing CU, ALU, Registers), **Memory**, **Disk Drives**, and other **I/Os**.



A von Neumann architecture scheme

Memory access through faster data+address buses.

Disks access is through slow I/O buses.



Harvard architecture

7.1 Introduction

- Data storage and retrieval is one of the primary functions of computer systems.
  - One could easily make the argument that computers are more useful to us as data storage and retrieval devices **than** they are as **computational machines**.
- All computers have I/O devices connected to them, and to achieve good performance I/O should be kept to a minimum!
- In studying I/O, we seek to understand the different types of I/O devices as well as how they work.

7.2 I/O and Performance

- **Sluggish** I/O throughput can have a **ripple effect**, **dragging** down overall system performance.
  - This is **especially** true **when virtual memory** is involved.
- The fastest processor in the world is of little use if it spends most of its time **waiting for data**.
- If we really **understand** what's happening in a computer system, we can make the best possible **use of its resources**.

7.3 Amdahl's Law

- The overall performance of a system is a result of the interaction of all of its components.
- **System performance** is most effectively improved when the performance of the most heavily **used components is improved**.
- This idea is quantified by Amdahl's Law:

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

where S is the overall speedup; f is the fraction of work performed by a faster component; and k is the speedup of the faster component.

7.3 Amdahl’s Law

- Amdahl’s Law gives us **a handy way** to estimate the **performance improvement** we can expect **when we upgrade** a system component.

Problem: On a large system, suppose we can upgrade a CPU to make it 50% faster for $8,000 or upgrade its disk drives for $7,000 to make them 150% faster.
- Processes spend 70% of their time running in the CPU and 30% of their time waiting for disk service.
- An upgrade of which component would offer the greater benefit for the lesser cost?

Example-2: Solve Example-1 with $10k CPU

Ans: Disk option is more economical with $333 per %performance increase!

Example-1:

CPU to make it 50% faster for $8,000 or upgrade its disk drives for $7,000 to make them 150% faster.

Processes spends:
70% time running in the CPU, and
30% time for disk data

7.3 Amdahl's Law

- The CPU option offers a 30% speedup:

$$f = 0.70, \quad k = 1.5 \qquad S = \frac{1}{(1 - 0.7) + 0.7/1.5} = 1.3$$

- And the disk drive option gives a 22% speedup:

$$f = 0.30, \quad k = 2.5 \qquad S = \frac{1}{(1 - 0.3) + 0.3/2.5} = 1.22$$

- Each 1% of improvement for the processor costs $266, and for the disk a 1% improvement costs $318.

Should price/performance be your only concern?

CPU is more economical!

7.3 Amdahl’s Law Example-3

On a large system, suppose we can upgrade a CPU to make it 40% faster for $6,000 or upgrade its disk drives for $5,000 to make them 130% faster. Processes spend 80% of their time running in the CPU and 20% of their time waiting for disk service. An upgrade of which component would offer the greater benefit for the lesser cost?

CPU: S=1.296	$203 per %speedup
Disk: S=1.127	$394 per %speedup

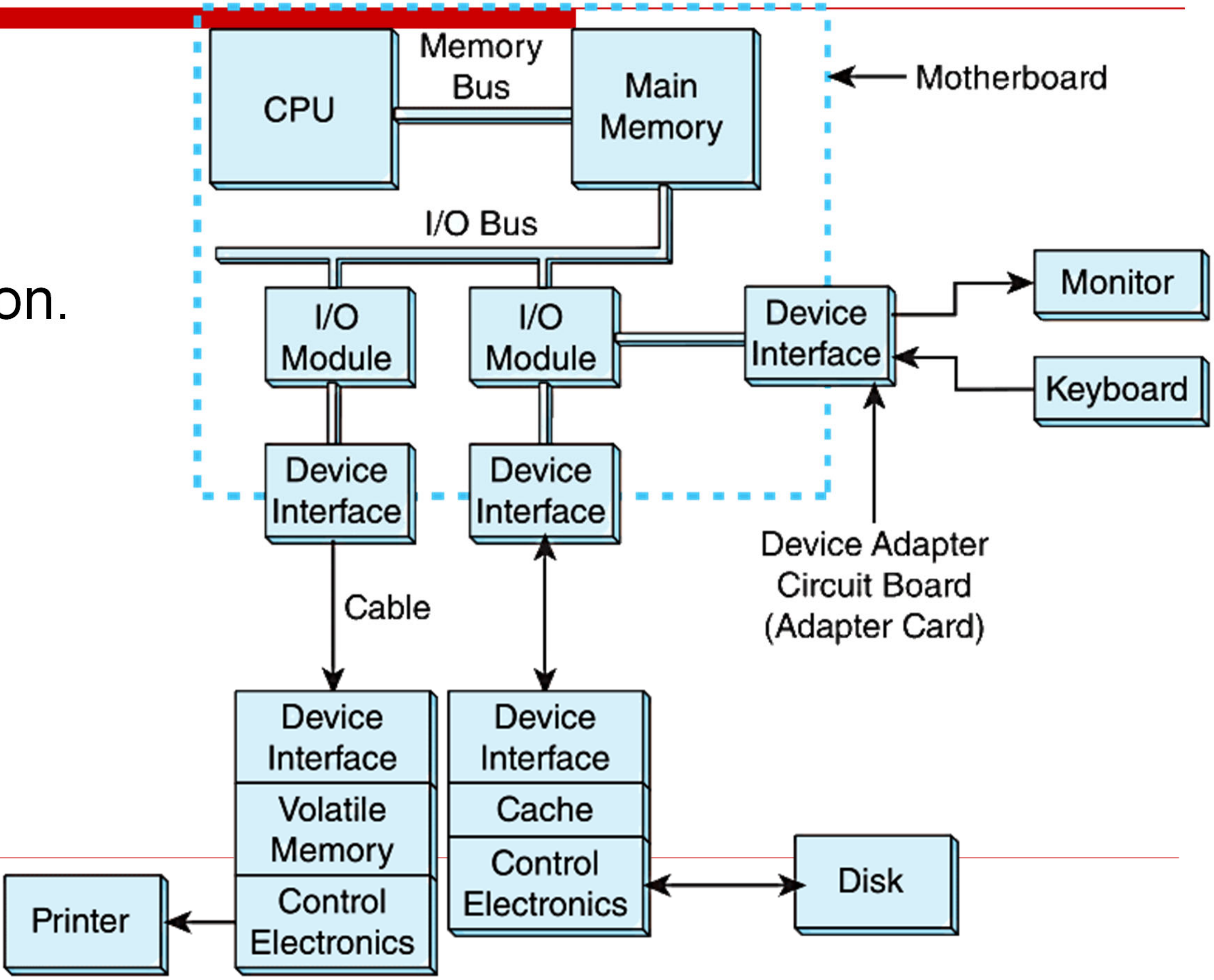
CPU upgrade is more economical!

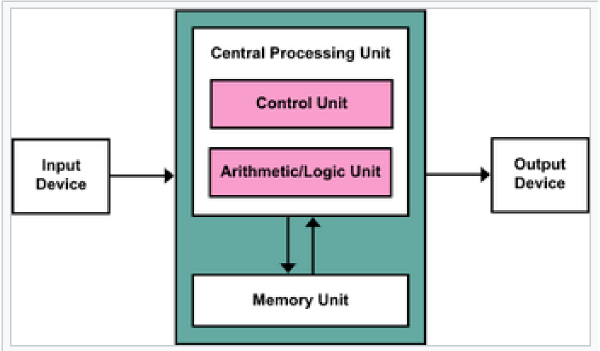
7.4 I/O Architectures

- We define input/output as a subsystem of components that moves ***coded data*** between **external** devices **and a host** system.
- I/O subsystems include:
  - Blocks of main memory that are **devoted to I/O** functions.
  - Buses that move data into and out of the system.
  - Control modules in the host and in peripheral devices
  - Interfaces to external components such as keyboards and disks.
  - Cabling or communications links between the host system and its peripherals.

7.4 I/O Architectures

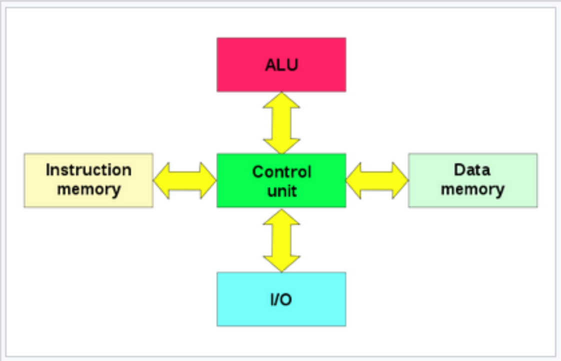
This is a model I/O configuration.





A von Neumann architecture scheme

I/O Architecture



Harvard architecture

- I/O can be controlled in five general ways.
  - *Programmed I/O* **reserves a register** for each I/O device, like InReg/OutReg (in MARIE). Each register is continually polled to detect data arrival.
  - *Interrupt-Driven I/O* allows the CPU to do other things **until** I/O is **requested**.
  - *Memory-Mapped I/O* **shares memory address** space between **I/O devices** and program memory.
  - *Direct Memory Access (DMA)* **offloads I/O processing** to a special-purpose chip that takes care of the details.
  - *Channel I/O* uses **dedicated** I/O processors.

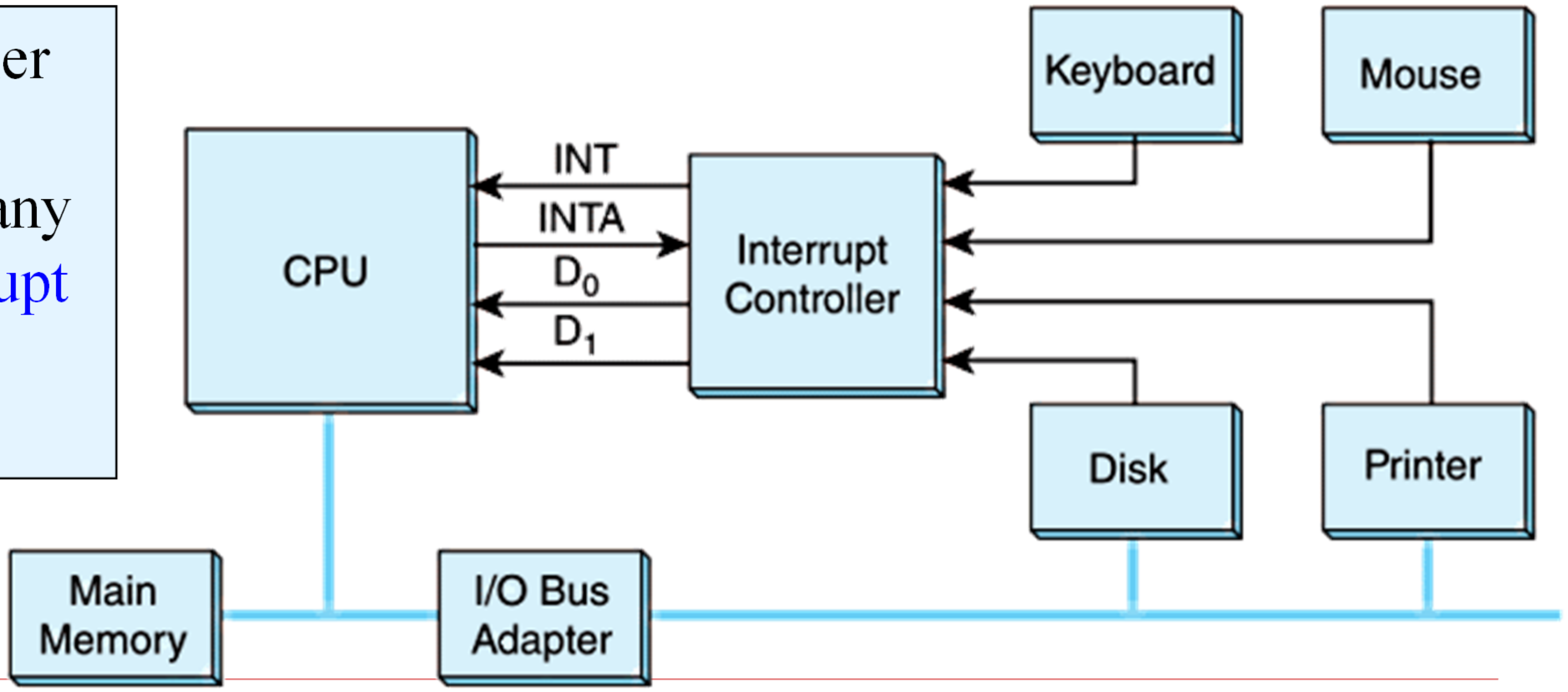
1- Programmed: poll buffer register
2- Interrupt-driven: request to CPU
3- Memory-mapped: memory+i/o shared address space
4- DMA: memory-i/o direct access through chip
5- Channeled: via dedicated processors.

7.4 I/O Architectures

This is an idealized I/O subsystem that uses interrupts.

Each device connects its interrupt line to the interrupt controller.

The controller signals the CPU when any of the interrupt lines are asserted.



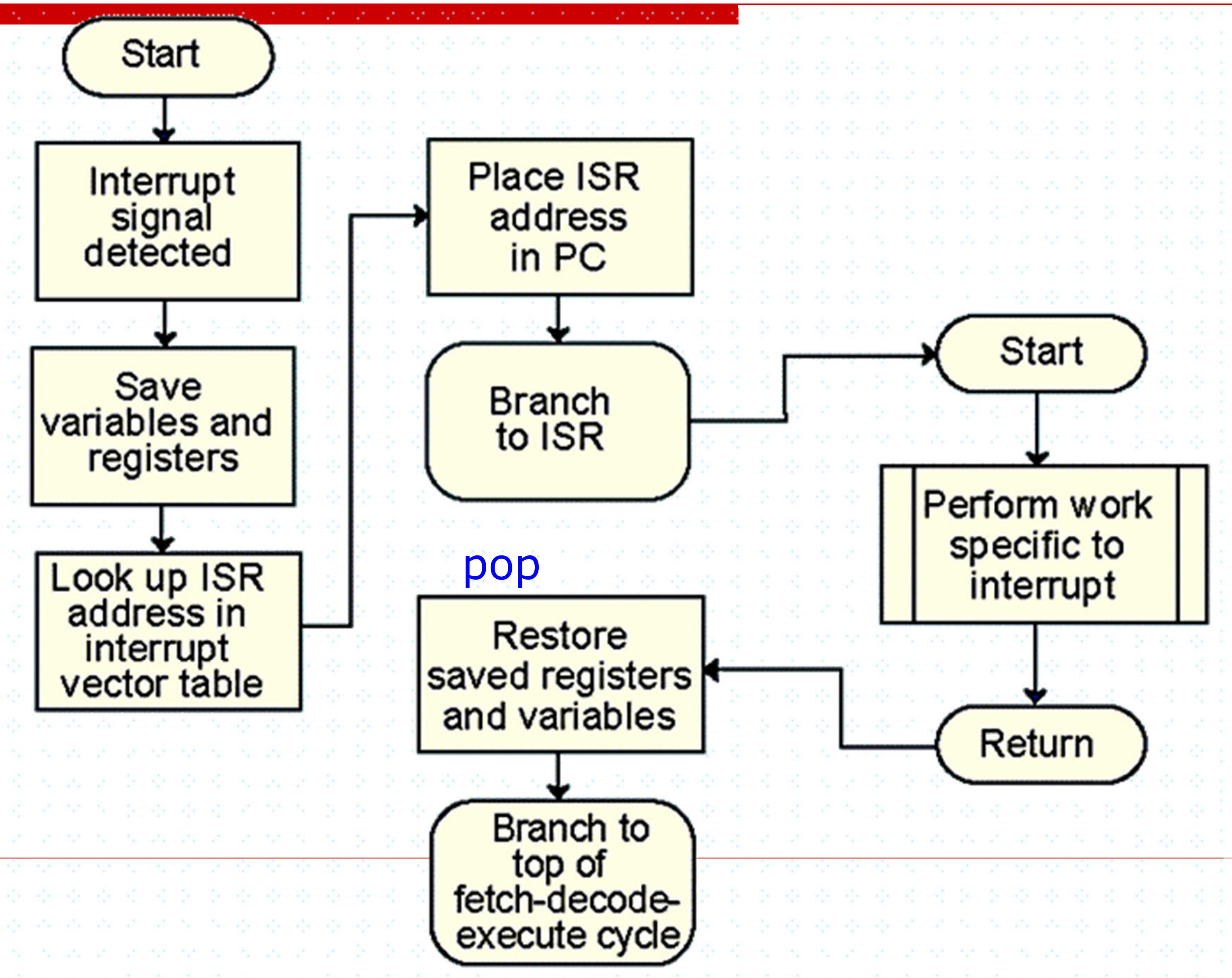
7.4 I/O Architectures

- Recall from Chapter 4 that in a system that uses interrupts, the **status** of the interrupt signal is **checked at the top of** the fetch-decode-execute **cycle**.
- The **particular code** that is **executed** whenever an interrupt occurs is **determined by** a set of addresses called *interrupt vectors (IV)* stored in **low memory**.
- The **system state is saved (on stack)** before the interrupt service routine is executed and is **restored** afterward.

We provide a flowchart on the next slide.

7.4 I/O Architectures

push
all
to
stack


Start
Interrupt signal detected
Save variables and registers
Look up ISR address in interrupt vector table
Place ISR address in PC
Branch to ISR
Start
Perform work specific to interrupt
Return
pop
Restore saved registers and variables
Branch to top of fetch-decode-execute cycle

1- Programmed: poll buffer register
2- Interrupt-driven: request to CPU
3- Memory-mapped: memory+i/o shared address space
4- DMA: memory-i/o direct access through chip
5- Channelled: via dedicated processors.

7.4 I/O Architectures

- In memory-mapped I/O devices and main memory share the same address space.
  - Each I/O device has its own **reserved block** of memory.
  - Memory-mapped I/O therefore **looks just like a memory access** from the point of view of CPU.
  - Thus the same instructions to move data to and from both I/O and memory, greatly **simplifying system design**.
- In **small systems** the low-level **details** of the data transfers are **offloaded** to the I/O controllers built into the I/O devices.

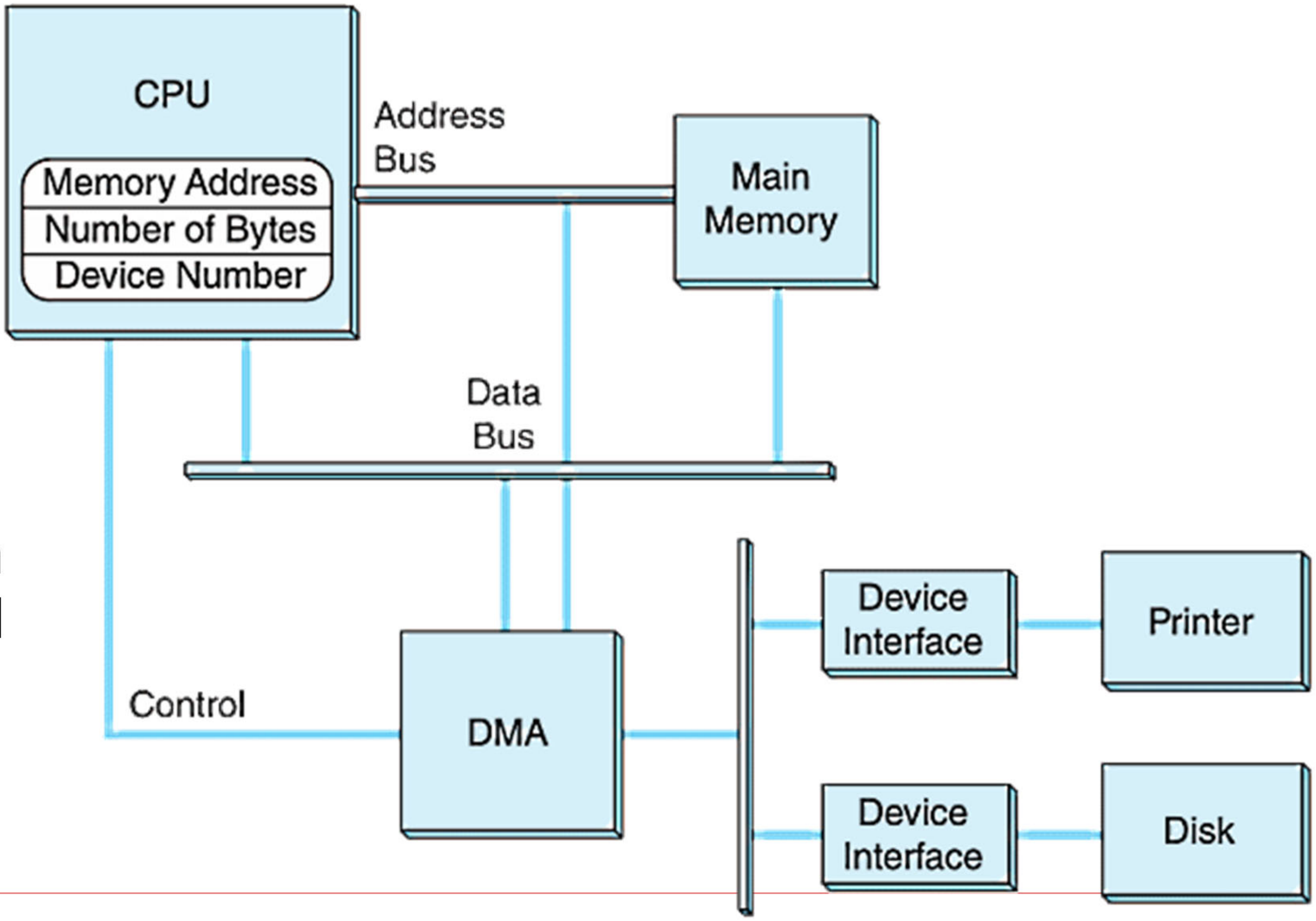
1- Programmed: poll buffer register
2- Interrupt-driven: request to CPU
3- Memory-mapped: memory+i/o shared address space
4- DMA: memory-i/o direct access through chip
5- Channelled: via dedicated processors.

7.4 I/O Architectures

This is a DMA configuration.

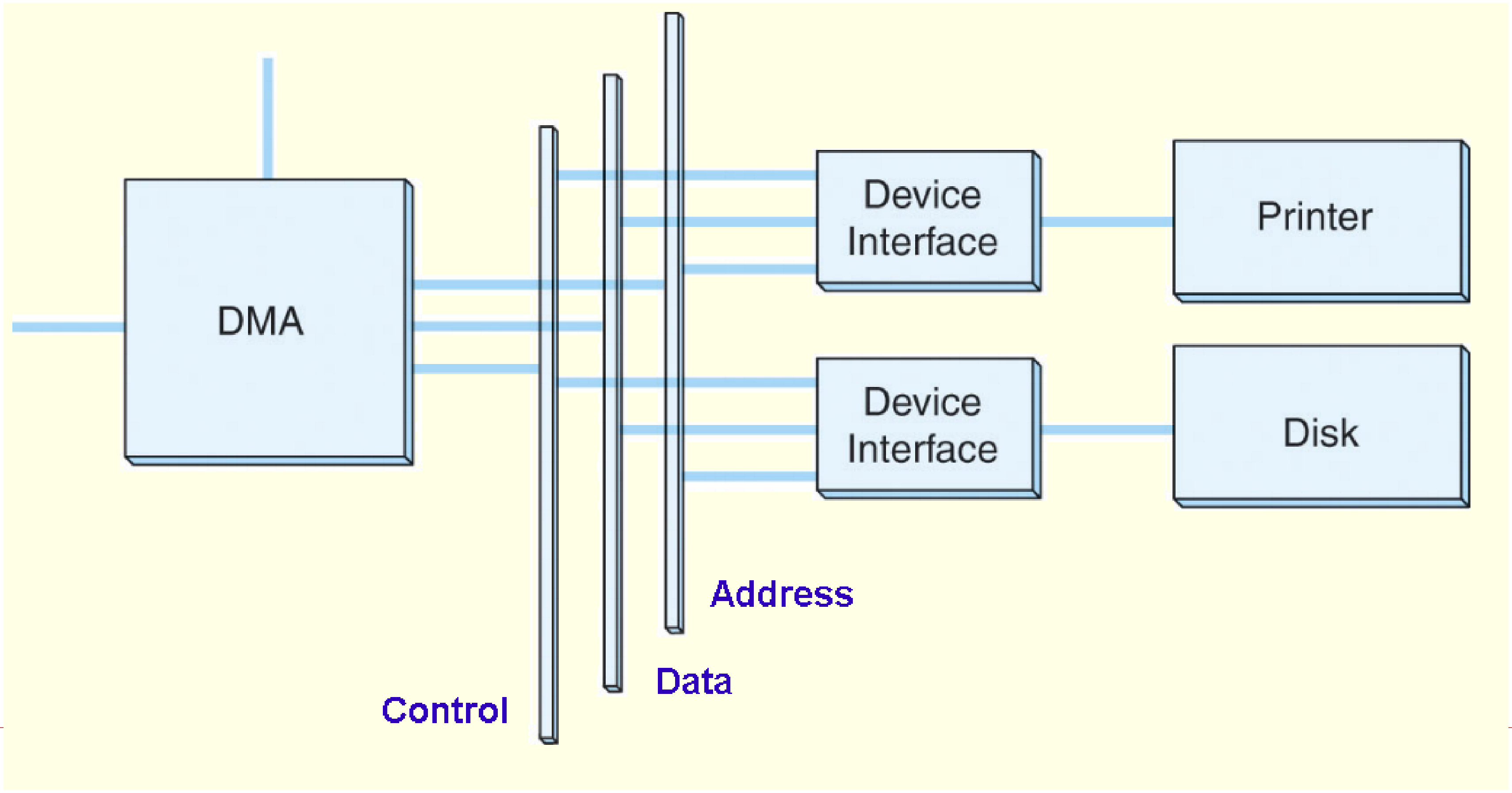
Notice that the DMA and the CPU **share the bus**.

The DMA runs at a higher **priority** and **steals memory cycles** from the CPU.



7.4 I/O Architectures

This is a generic DMA configuration showing how the DMA circuit connects to a data bus.



1- Programmed: poll buffer register
2- Interrupt-driven: request to CPU
3- Memory-mapped: memory+i/o shared address space
4- DMA: memory-i/o direct access through chip
5- Channelled: via dedicated processors.

7.4 I/O Architectures

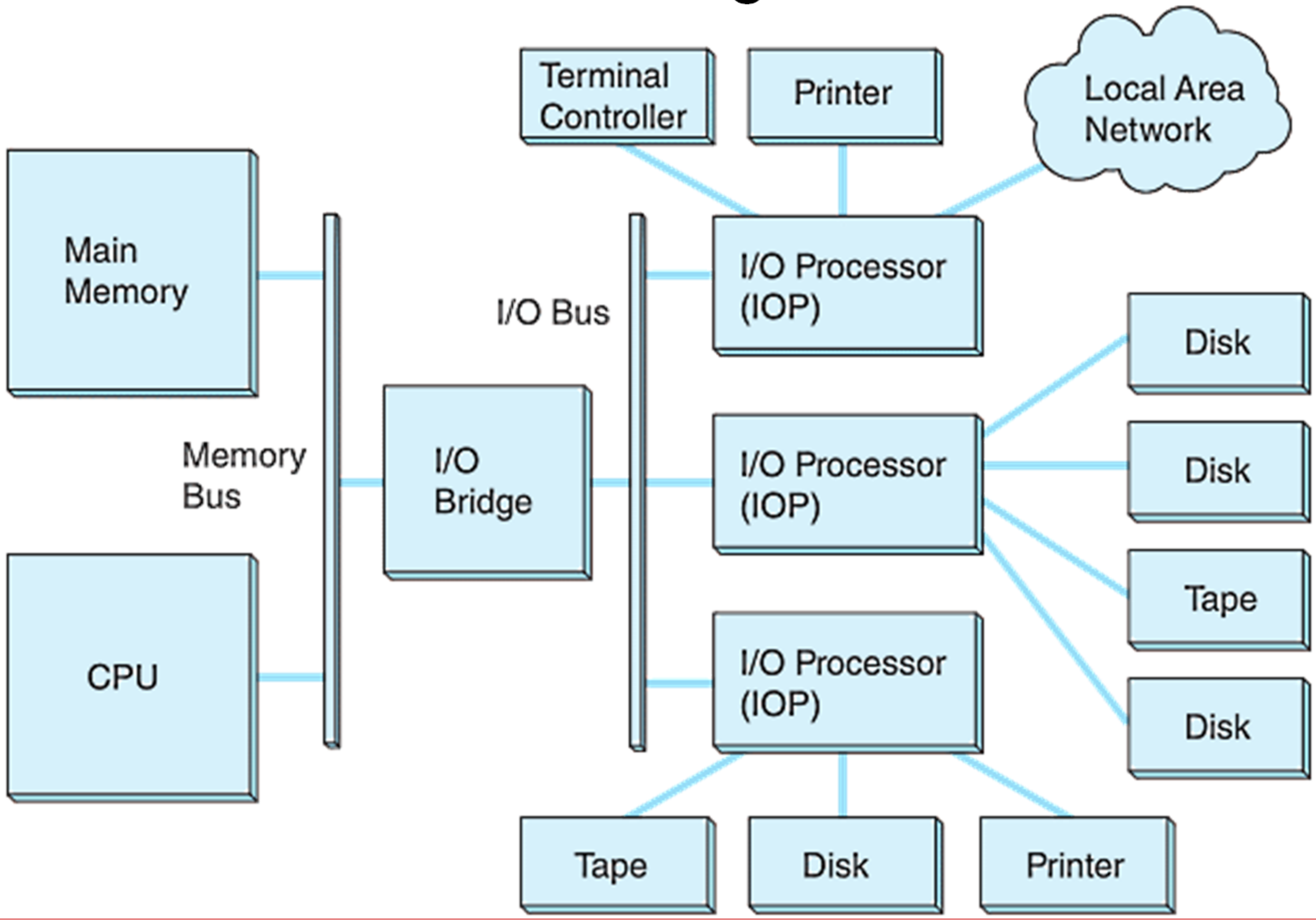
- **Very large systems employ** channel I/O.
- Channel I/O consists of one or more I/O processors (**IOP/s**) that control various channel paths.
- **Slower devices** such as terminals and printers are combined (*multiplexed*) **into a single faster channel**.
- On IBM **mainframes**, multiplexed channels are called *multiplexor channels*, the faster ones are called *selector channels*.

7.4 I/O Architectures

- Channel I/O is **distinguished from DMA by** the intelligence of the IOPs.
- The IOP negotiates **protocols**, issues device **commands**, **translates** storage coding to memory coding, and can **transfer entire files** or groups of files **independent** of the host CPU.
- The **host** has to **create the program** instructions only **for the I/O operation** and tell the IOP **where to find** them.

7.4 I/O Architectures

- This is a channel I/O configuration.



1- Programmed: poll buffer register
2- Interrupt-driven: request to CPU
3- Memory-mapped: memory+i/o shared address space
4- DMA: memory-i/o direct access through chip
5- Channelled: via dedicated processors.

7.4 I/O Architectures: **another way**

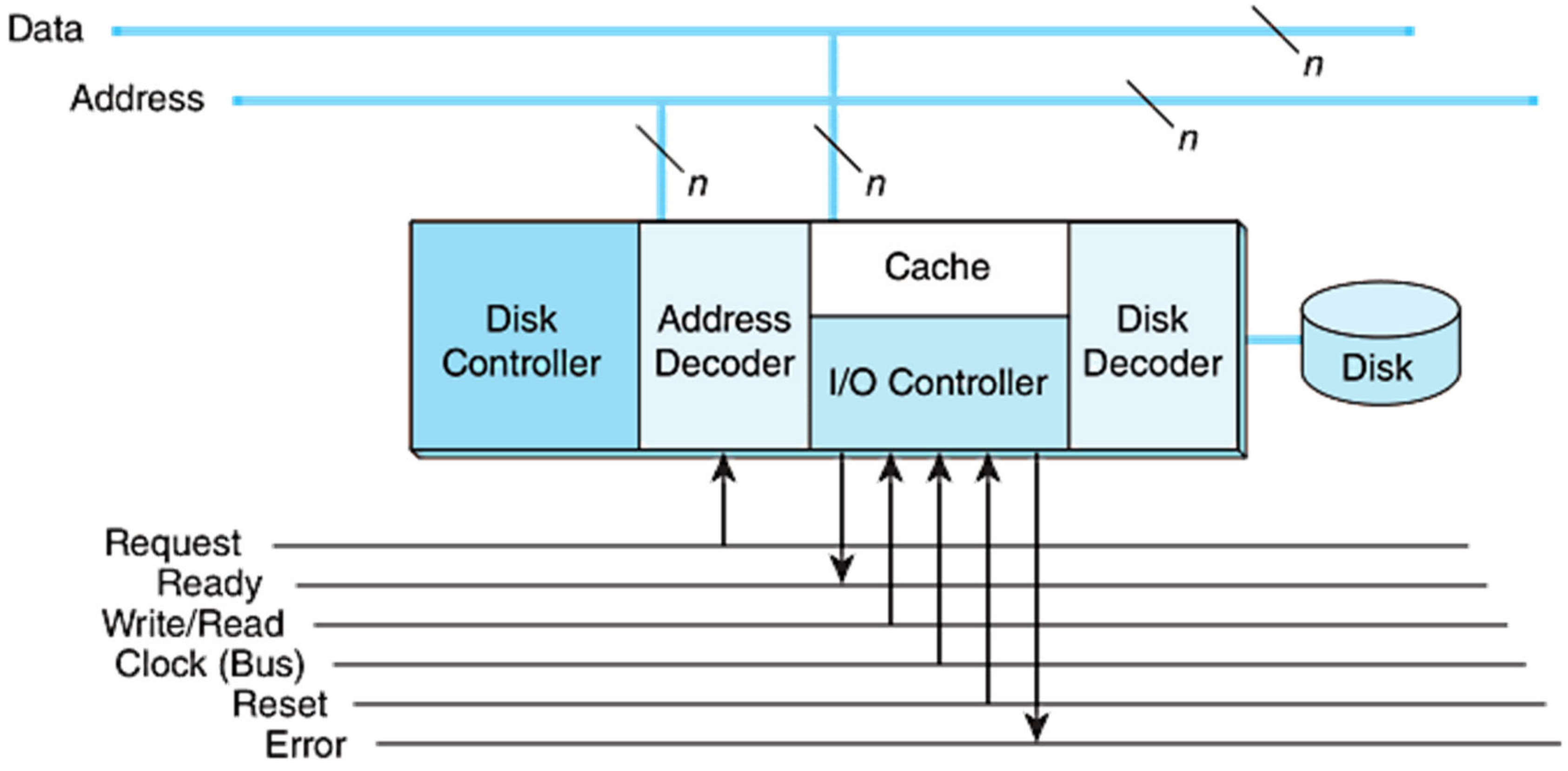
- Character I/O devices **process one byte** (or character) **at a time**.
  - Examples include modems, keyboards, and mice.
  - Keyboards are usually connected through an interrupt-driven I/O system.
- Block I/O devices **handle bytes in groups**.
  - Most mass **storage** devices (**disk** and tape) are block I/O devices.
  - Block I/O systems are most efficiently connected through DMA or channel I/O.

7.4 I/O Architectures

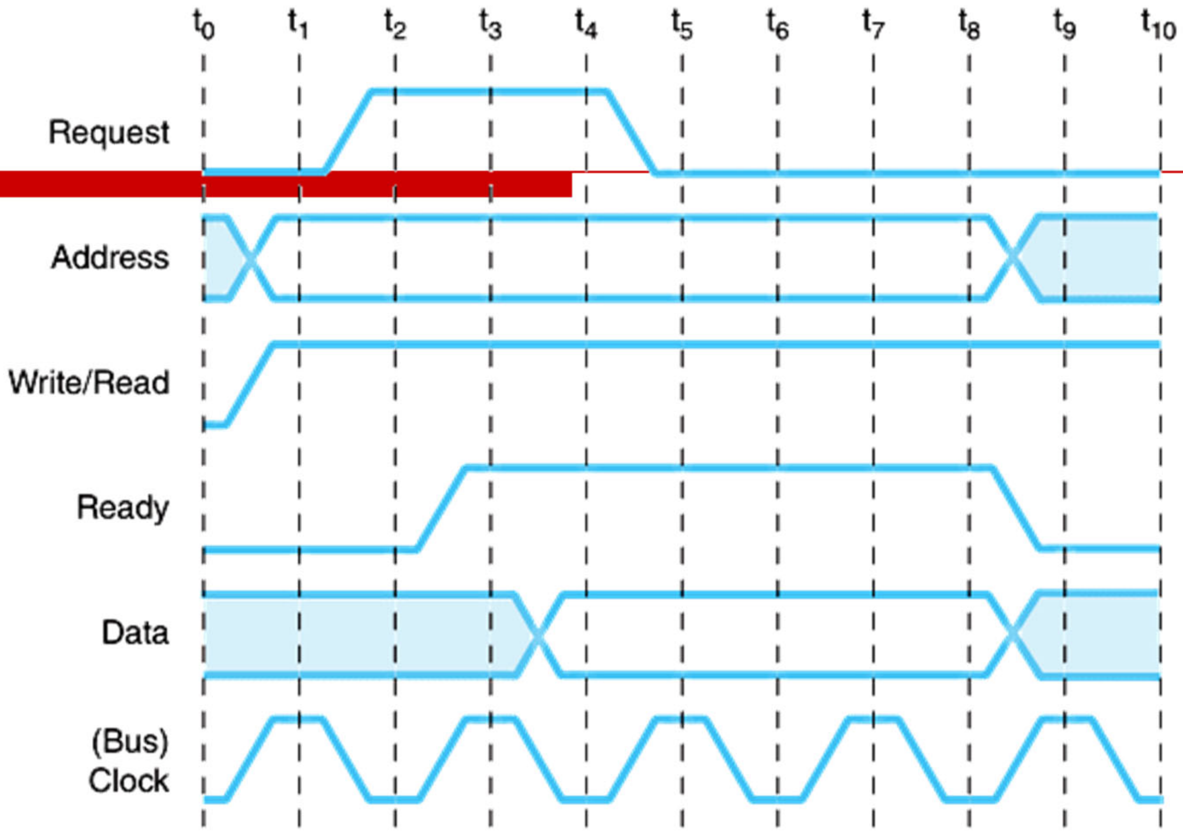
- I/O buses, unlike memory buses, **operate asynchronously**. Requests for **bus access** must be **arbitrated** among the devices involved.
- **Bus control lines** activate the devices when they are needed, **raise signals when errors** have occurred, and **reset devices** when necessary.
- The number of **data lines** is the *width* of the bus.
- A bus clock **coordinates** activities and **provides bit cell boundaries**.

7.4 I/O Architectures

This is how a bus connects to a disk drive.



Timing diagrams, such as this one, define bus operation in detail.

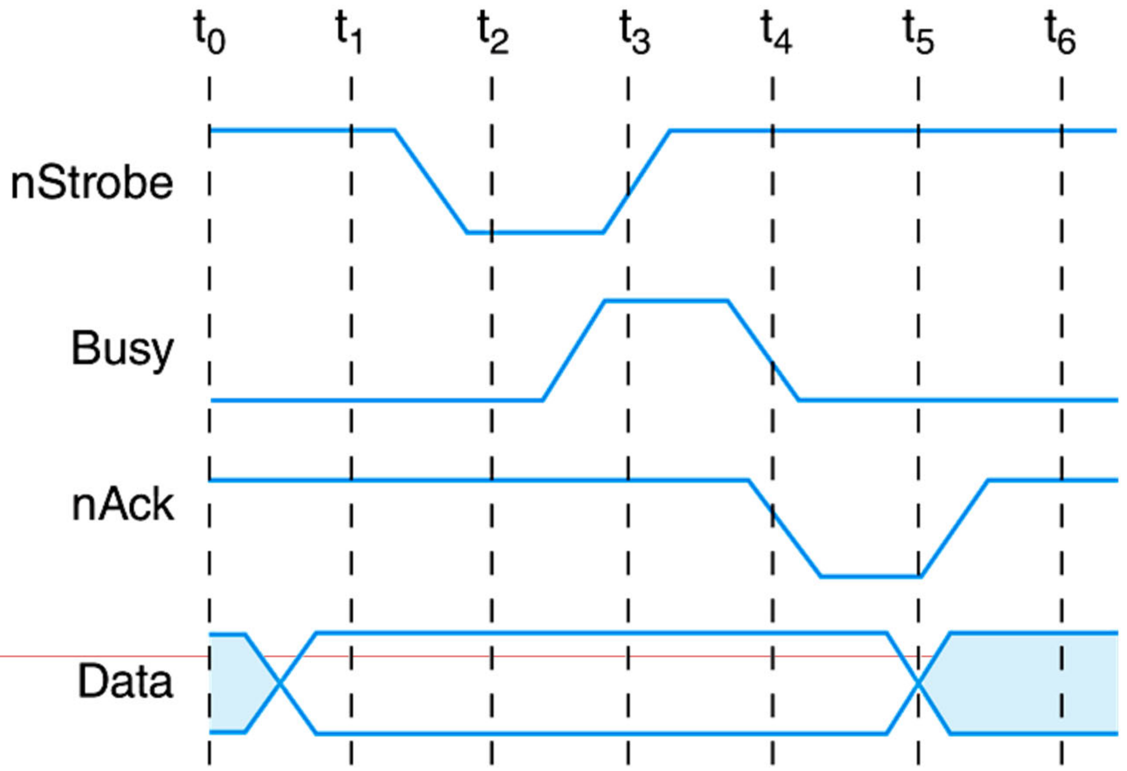


Time	Salient Bus Signal	Meaning
t_0	Assert Write	Bus is needed for writing (not reading)
t_0	Assert Address	Indicates where bytes will be written
t_1	Assert Request	Request write to address on address lines
t_2	Assert Ready	Acknowledges write request, bytes placed on data lines
t_3–t_7	Data Lines	Write data (requires several cycles)
t_8	Lower Ready	Release bus

7.5 Data Transmission Modes

- Bytes can be conveyed from one point to another by sending their **encoding signals simultaneously** using *parallel data transmission* or by sending them one bit at a time in ***serial data transmission***.

  – Parallel data transmission for a printer **resembles** the signal protocol of a memory bus:



7.5 Data Transmission Modes

- In parallel data transmission, the interface requires **one conductor for each bit**.
- Parallel cables are **fatter** than serial cables.
- Compared with parallel data interfaces, **serial** communications interfaces:
  - Require **fewer conductors**.
  - Are **less susceptible** to attenuation.
  - Can transmit data **farther and faster**.

Serial communications interfaces are suitable for time-sensitive (***isochronous***) data such as voice and video.

Chapter 7 Conclusion

- I/O systems are **critical** to the overall performance of a computer system.
- **Amdahl's** Law **quantifies** this assertion.
- **I/O systems consist** of memory blocks, cabling, control circuitry, interfaces, and media.
- I/O control **methods** include programmed I/O, interrupt-based I/O, DMA, and channel I/O.