I/O Software

Software is organized as series of Layers.
Lower Layer hides Hardware from upper layers.
Upper Layers present nice interface to user.

Goals of I/O software

1- Device independence:

Should be able to run command like:
sort <input> output

This should work if input/output is floppy or disk or terminal

2- Uniform Naming:

User need not to know which name corresponds to which device.

in Unix, disks are integrated in hierarical file system.

all files and devices are addressed the same way by a path name.

:/usr/local/name → floppy

3- Error handling:

handled at low level, close to Hardware.
Usually by device driver (read block)

Structured I/O Software Layers

1- Interrupt Handlers (Lowest)

Each process starting I/O operation block until I/O has completed and interrupt occurs (To make the process run).

blocking — unblocking using semaphores

2- Device Driver

has all the device dependent code.
Use different device drivers for different devices.

it knows the registers in controller and characteristics of device (sector #, cylinders, heads, motor drives, ...)

Device Drivers

it receives I/O requests from a higher level layer, translate it and starts to issue commands to controller (by writing to controller device registers)

3 - Device Independent I/O Software

Perform the following functions:

- uniform interface to all devices
- naming
- protection (Prevent users from accessing devices that they are not entitled to access)

MS-DOS no protection

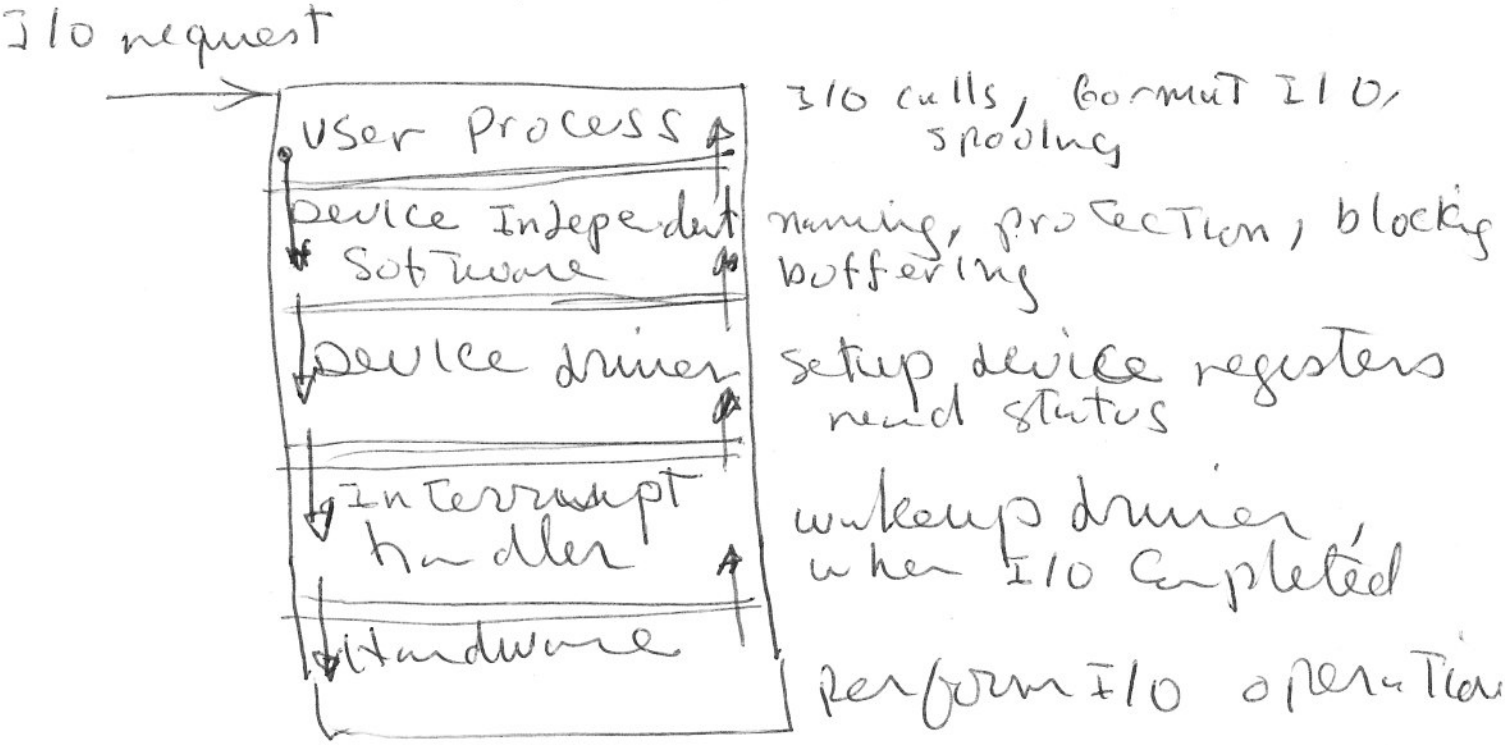
UNIX files are protected by rwx bits.

- Provide uniform block size to user
- Buffering for faster devices.

4- User I/O Software

- Formatting Input/ output like Printf(- -), gets

- Spooling: protecting specific files from direct use by user.
  (print a file could open file for hours -), only completed files go to spooling directory to be printed.
  User puts a complete file in spooling directory.



Operating system and I/O

- I/O Hardware (seen by O.S)
- I/O software (different layers)

Principles of I/O Hardware

1- I/O Devices

O.S divided them to two types:

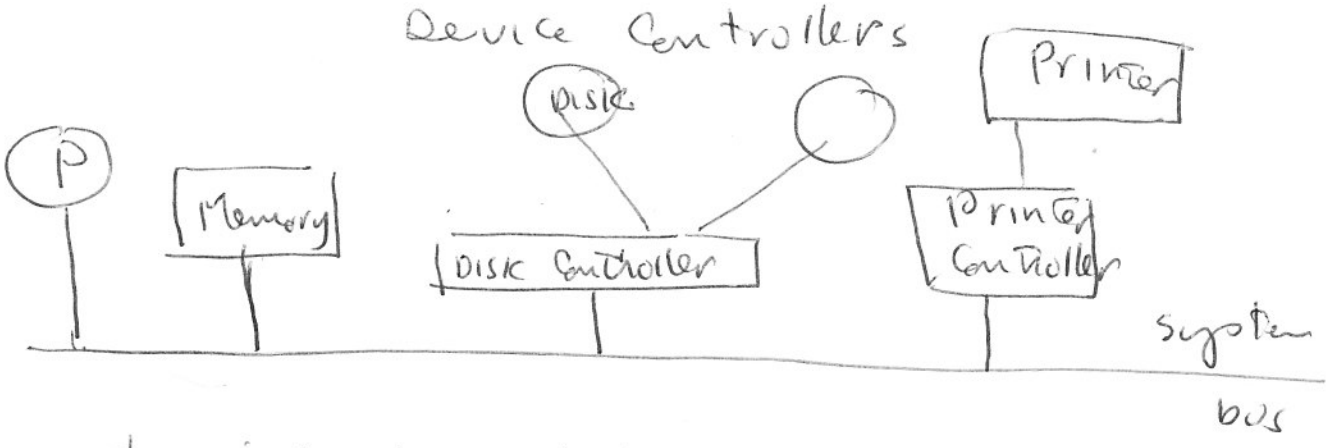
1- Block Devices with block sizes (128 - 1024 byte).
Devices read/write a block independet of other blocks.
Example: DISKS

2- Character Devices: deliver or accept a stream of characters.
example: terminals, Printers, Networks

OS deals with I/O independent of device type. (leaves to device driver).

3- Device Controllers

the electronic component used as adapter (printed circuit board inserted into computer bus).



IF The interface between Controller and Device is standard (IEEE, ANSI, ..) Companies can make Controllers or Devices that fit the interface and OS deals with Controller not Device.

Example Disk Controller converts serial bit stream from disk into block of bytes stored in a Buffer and performs error detection/correction then copies it to memory

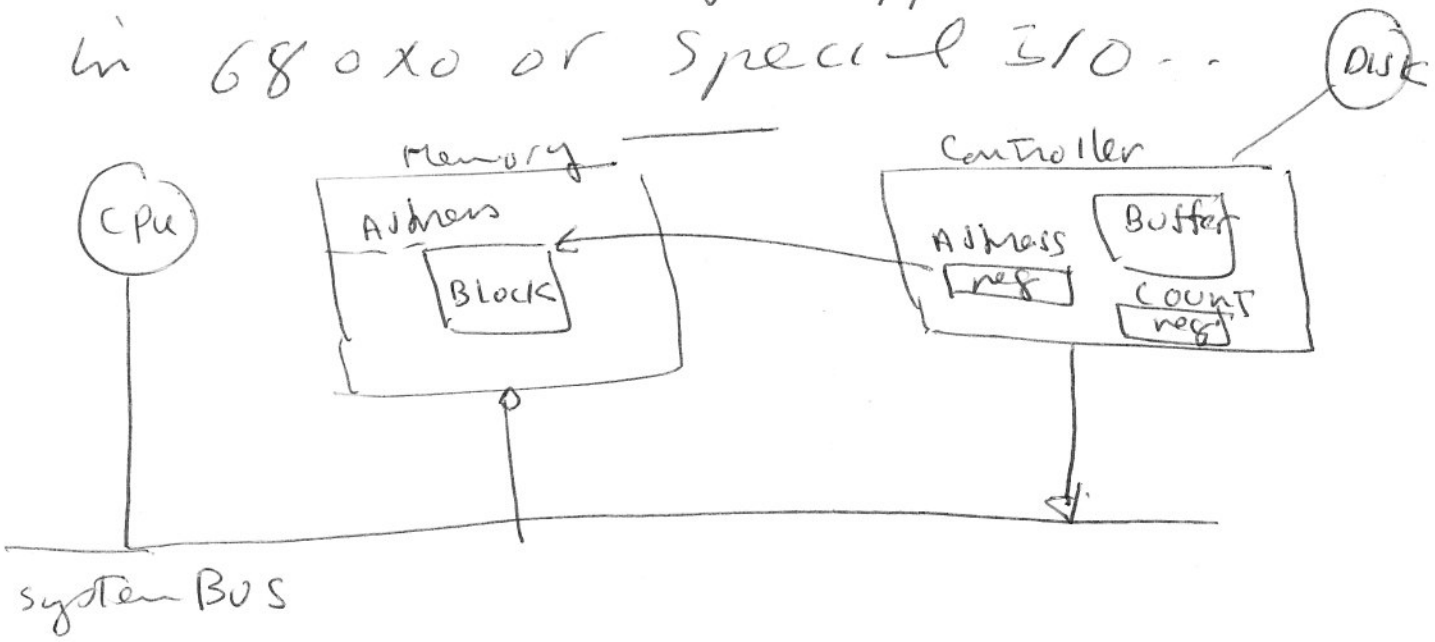
Example CRT terminal Controller

it reads characters (to be displayed) from memory and generates horizontal and VIRTICAL signals for displaying character in correct position.

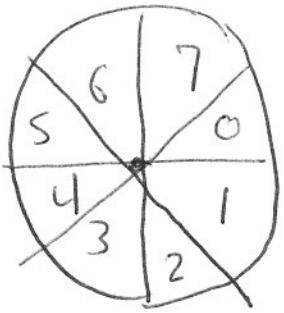
OS only initialize Controller for number of characters per line, number of lines per screen.

Controller Communication with CPU

Controller uses registers that are mapped to memory (memory mapped I/O) as in 68000 or Special I/O..



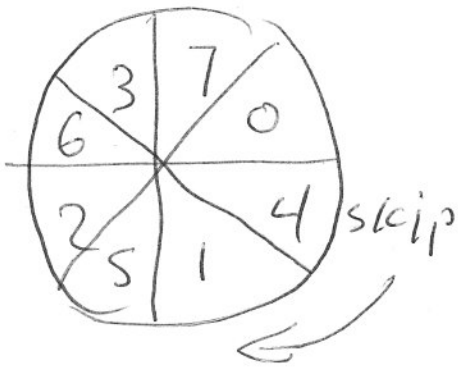
A Buffer is needed to store Data from DISK before transfer it to Memory.



DISK needs 8 rotations to transfer 8 Blocks 0-7.

Transfer Block 4 to Buffer, wait for DMA to transfer Buffer to Memory, next rotation transfer 1 to Buffer, then repeat

Disk with Interleaving:
skipping blocks from disk to allow for time of sending Data from Buffer to memory



no take 1
0 → 1 → 2 → 3
no take 2
4 → 5 → 6 → 7

send 0 → Buffer
send Buffer to memory, skipping 4
send 1 → Buffer, while skipping 5

I/O space for IBM PC

I/O Controller	I/O Address	Interrupt Vector
clock	040 - 030	8
keyboard	060 - 063	9
Hard disk	320 - 32F	13
Floppy DISK	3F0 - 3F7	14
printer	37F - 37F	15

DMA for Controller

- with No DMA, CPU must loops to read the bytes from Controller buffer one at a time and writes it to Memory (one at a time).

This is a waste of CPU Time.

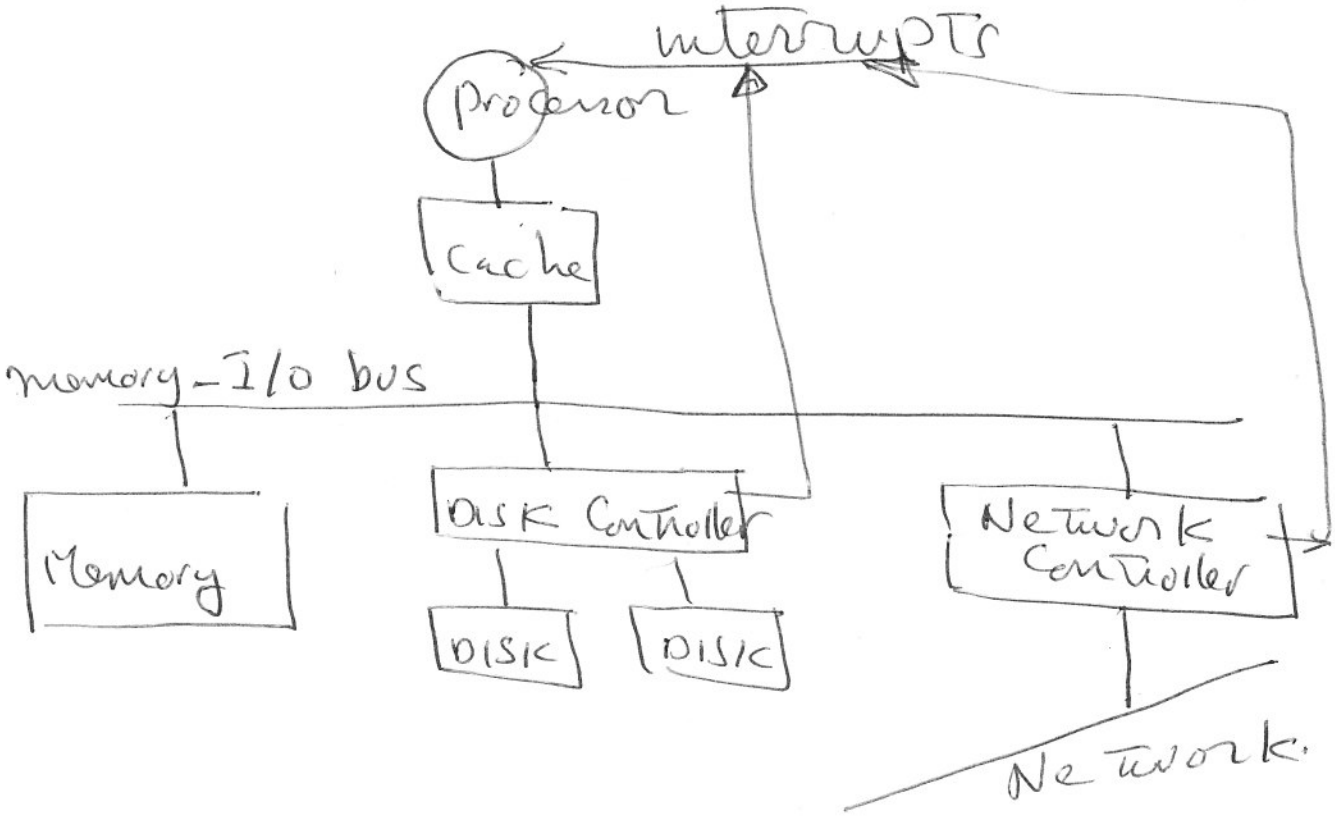
DMA:-

CPU gives Controller memory Address (where block is to go) and number of bytes to be transferred.

Controller not CPU copies first word from buffer to memory, increments DMA Address and decrements Counts.

It repeats this until block is transfered

The Controller interrupt CPU for end of operation



System Block Diagram

I/O Performance

Depends on :-

1- CPU speed, Memory (cache, main memory)

2- Buses

3- I/O Controller

4- I/O Device

5- I/O Software (Algorithms, ...)

I/O Performance

Performance metrics :-

- Throughput : I/O bandwidth for supercomputers (TNX processing APPLICATION), number of Jobs Per Time UNIT
- Response time, latency important in single user machines (PC, workstations) Time to complete one job

---

I/O and overall system performance

slow I/O always neglected but it has a lot of impact on system performance

Amdhal's Law : performance is limited by the portion that cannot be improved

$$\text{speedup} = \frac{1}{(1 - \text{fraction\_enh}) + \frac{\text{fraction\_enh}}{\text{amount of improv.}}}$$

Example find speedup if 60% of Application is improved 5 times.

Answer

$$\text{speedup} = \frac{1}{.4 + \frac{.6}{5}} = 1.92 \text{ Times}$$

Example: if an application takes 100 seconds, and 90s for cpu and 10s for I/O cpu time improves by 50% per year, how much faster will the application run after 5 years.

Answer cpu time after 1 year = $\frac{90}{1.5}$
$= 60$

cpu time in 2 years = $\frac{60}{1.5} = 40$

3 years = $\frac{40}{1.5} = 27$

4 Th yer = $\frac{27}{1.5} = 18$

5 Th = $\frac{27}{1.5} = 12$

speedup cpu = $\frac{90}{12} = 7.5$

speedup = $\frac{1}{(1-.9) + \frac{.9}{7.5}} = \frac{1}{.1+.12}$

$\approx 4.$

I/O Devices

Magnetic Disks

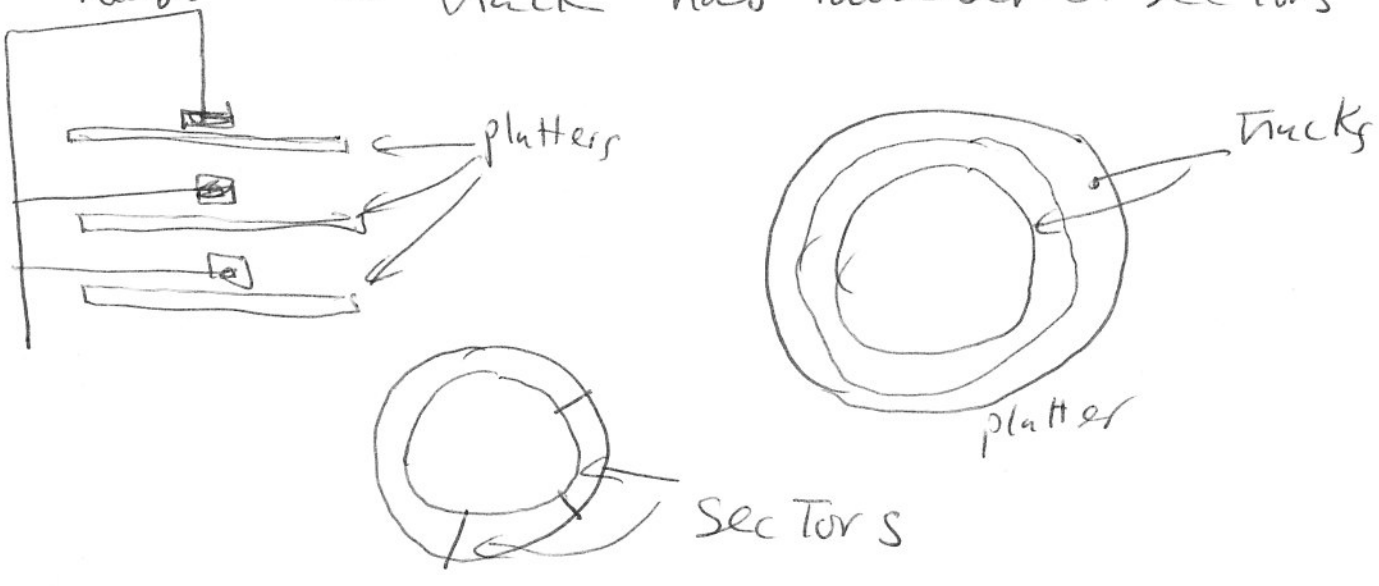
Two types: Floppy and hard disk
both are nonvolatile storage, large, inexpensive, slow, use rotating platter coated with magnetic material and moveable head used for read/write

differences: - hard disk have platters that are more rigid (glass or metal), higher density, spins faster, and can have more than one platter

Hard disk

organization :-

- multiple platters
- platter has multiple tracks
- track has number of sectors



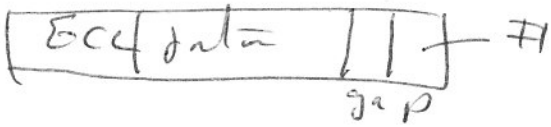
Typical hard disk organization:

500-2000 tracks per platter

32-128 sectors per track

Sector is the smallest unit that can be read/written (block).

Format: Sector number, gap, Data, Error Correction Code



- Same number of sectors per track, and same number of bits per each track (Constant bit density).

  Size of sector (physical) changes with diameter (ID). different timing

cylinder: all tracks under the head at given point in all surfaces.

H. Disk operation

uses operating system:

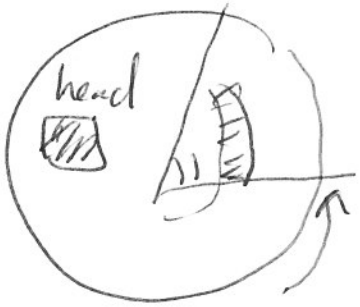
three phases:-

1- Seek: (Average about 12 ms)
To position head over proper track.
Time depends on O/S' scheduling

and could reduce it by 25% - 35%

2 - Rotational:
Must wait for desired sector under head

Time Average = $\frac{1/2 \text{ rot}}{\text{RPM}}$



For 3600 RPM

rotational time = $\frac{1/2}{3600/60}$

= 8.3 ms

3 - Transfer Data:
Time to transfer block of bytes.
Function of size of block, rotation speed and recording density.
≈ rate about 2 - 4 MB/sec.

Example
Find Average time to read/write 512 byte sector for 4500 RPM disk if seek time = 20 ms, transfer rate 2MB/sec and Controller overhead = 2 ms

solution:

Seek Time = 20 ms

Rotational Time = $\frac{1/2}{4500/60}$ = 6.67 ms

Transfer Time = $\frac{512}{2 \times 10^6}$ = .25 ms

Total Time = 20 + 6.67 + .25 + 2 = 29 ms

Characteristics of 1993 magnetic disk

diameter = 5.25 inches

data capacity = 2.8 GB

cylinders = 2627

tracks per cylinder = 21

Sectors per track = 99

Bytes per sector = 512

Rotational speed = 5400 RPM

seek time = 11 ms

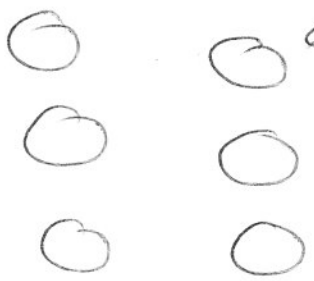
data transfer rate = 4.6 MB/sec.

Disk arrays

RAID (redundant array of inexpensive disks)

new organization:

array of small disks to increase throughput and availability (redundant disks)



Lost information can be reconstructed (redundant).

Problem: reliability is lower $\frac{1}{N}$

DRAM as a secondary storage

SSD
solid state disks built with DRAM with a battery.

- nonvolatile, use operating system for data transfer

Advantages: fast seek Time, higher transfer rate, higher reliability

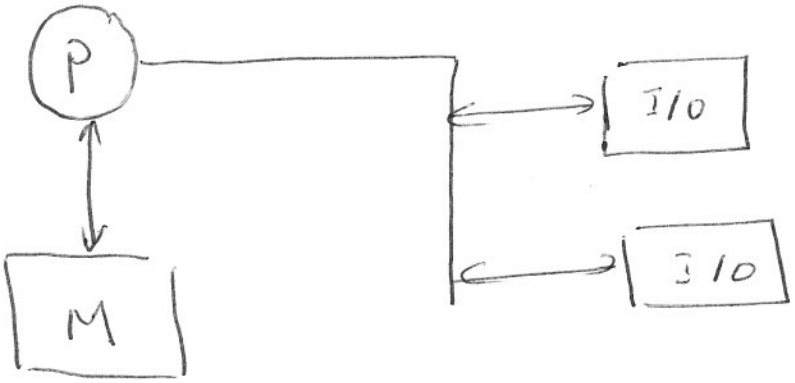
- greater error recovery

disadvantages: Cost, 50 Times higher

optical DISKS

CDs, Read only, optical compact disk
removable, inexpensive
high capacity.

Connecting I/O To Processor and Memory "Buses"



Bus: is a shared communication link to connect multiple subsystems. (set of wires).

Advantages of using Buses:

1- Low cost, single set of wires shared by multiple subsystems.

2- Easy to add new devices

3- Easy to move devices between systems using same bus standard.

Disadvantages:

1- It creates communication bottleneck (only one transaction, other must wait).

2 - Bus speed is limited by
- bus length
- number of devices on the bus

3 - Limited Bandwidth, it has limited number of wires, buffering.

Bus Requirements and Organization

Bus Consists of:

1 - Control lines To signal requests and acknowledge. It indicate type of information on Data lines.
2 - Data lines: Carry Data and Address

Bus Transaction Consists of:

1 - Sending the Address
2 - Receive or send Data

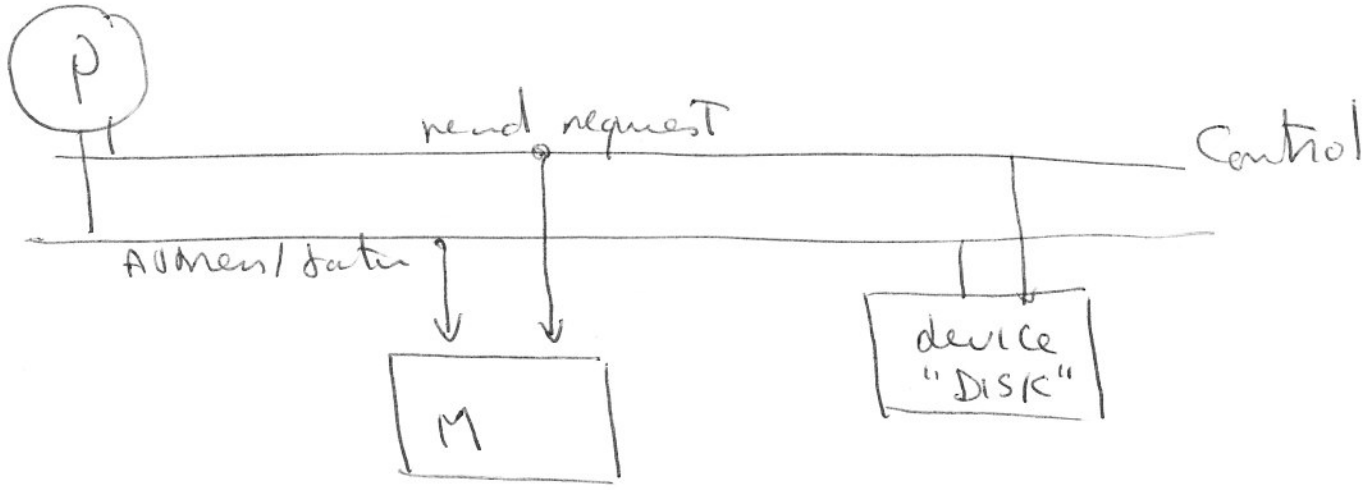
Bus Transaction Examples

Control lines: request
Data lines: Address/Data

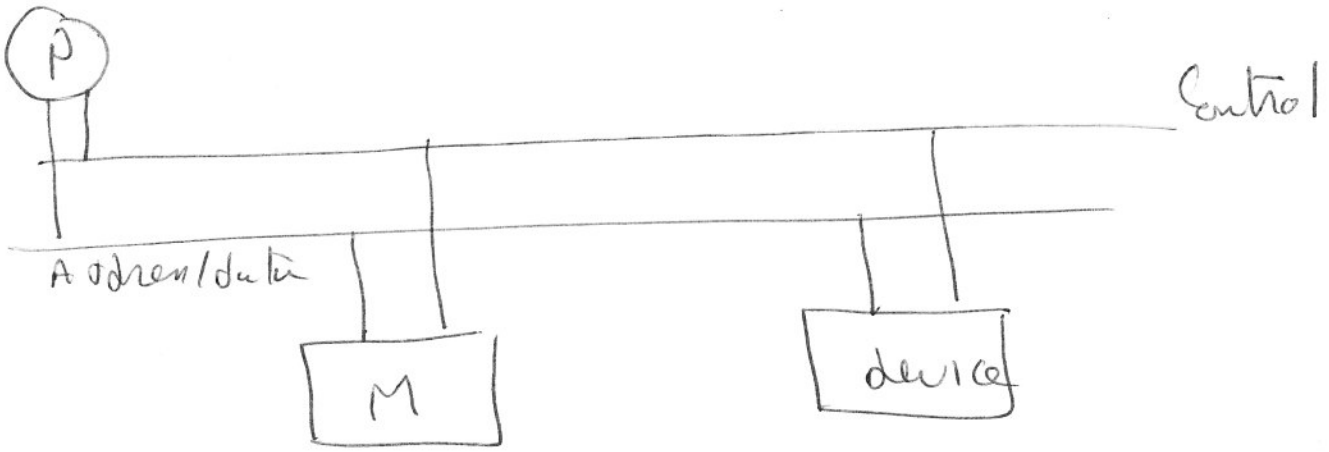
Example OUTPUT operation

Data read from memory and sent to device

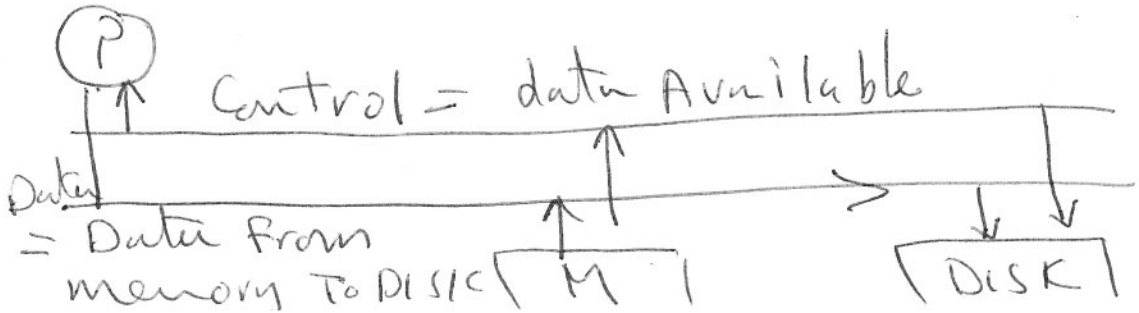
step1: read request (control)



step2 wait for Memory (data Available)

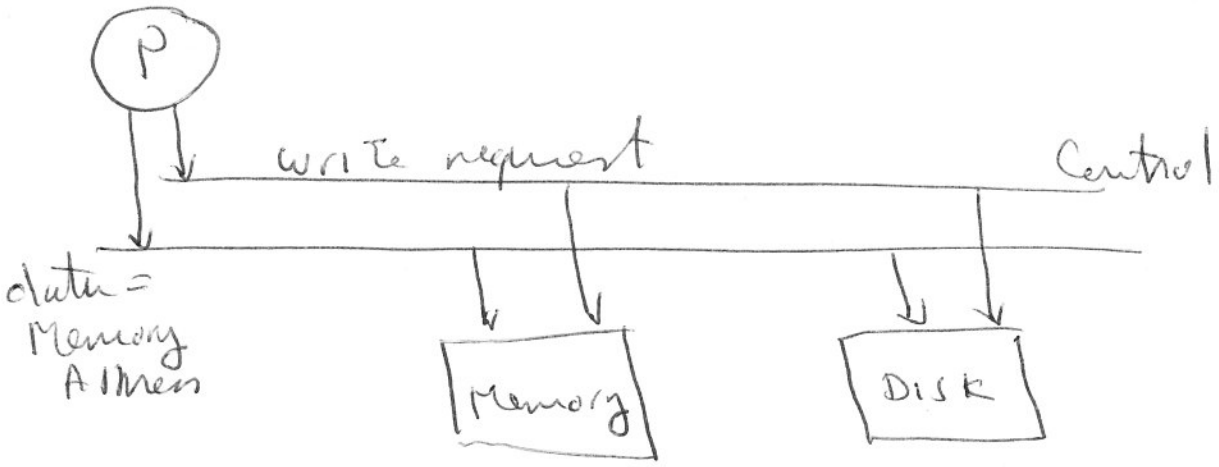


step3 Data ready to I/O

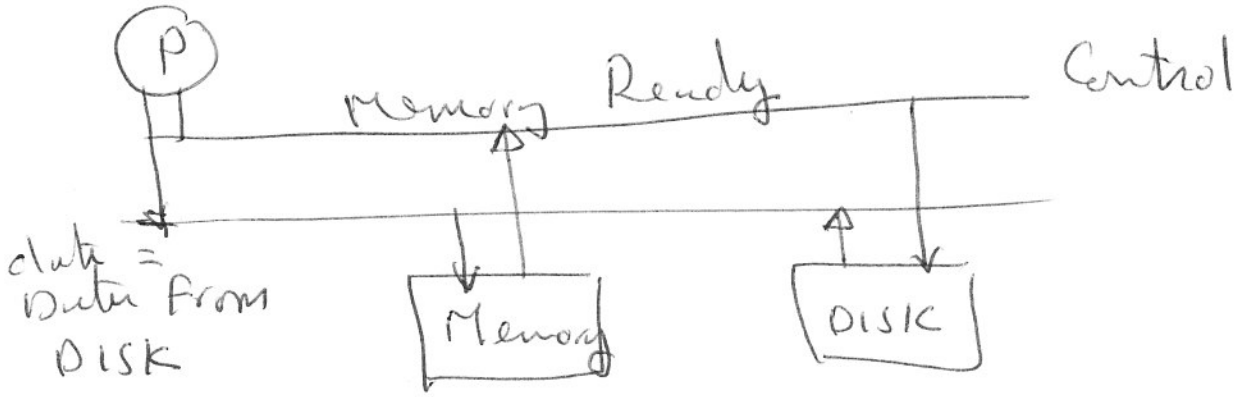


Example 2: Input operation:
Data write from disk to memory

step1: request to write



step2: Data Available



Types of Buses

Three Types:

1- Processor-memory buses:
- short, high speed and design specific
- only need to match memory system
- connects directly to processor

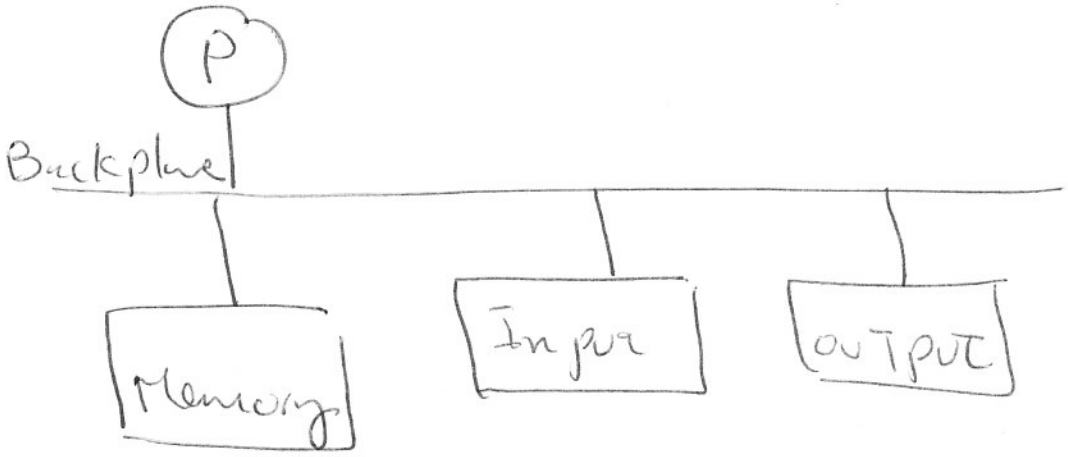
2- I/O Buses:
- Long, slow, should accept wide range of I/O devices
- uses simple interface to devices
- connects to processor-memory bus or backplane bus with Adaptor

3- Back plane BUS:
- Allow processor, memory and I/O Devices to share same bus
- cost advantage: one single bus for all components

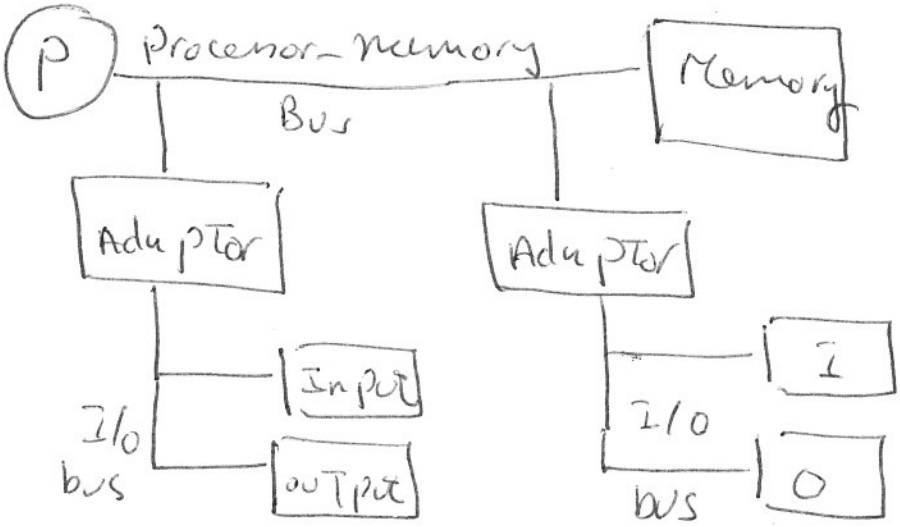
Computer system with Bus organization

1- Backplane Bus (Single Bus organization)

Example: IBM PC



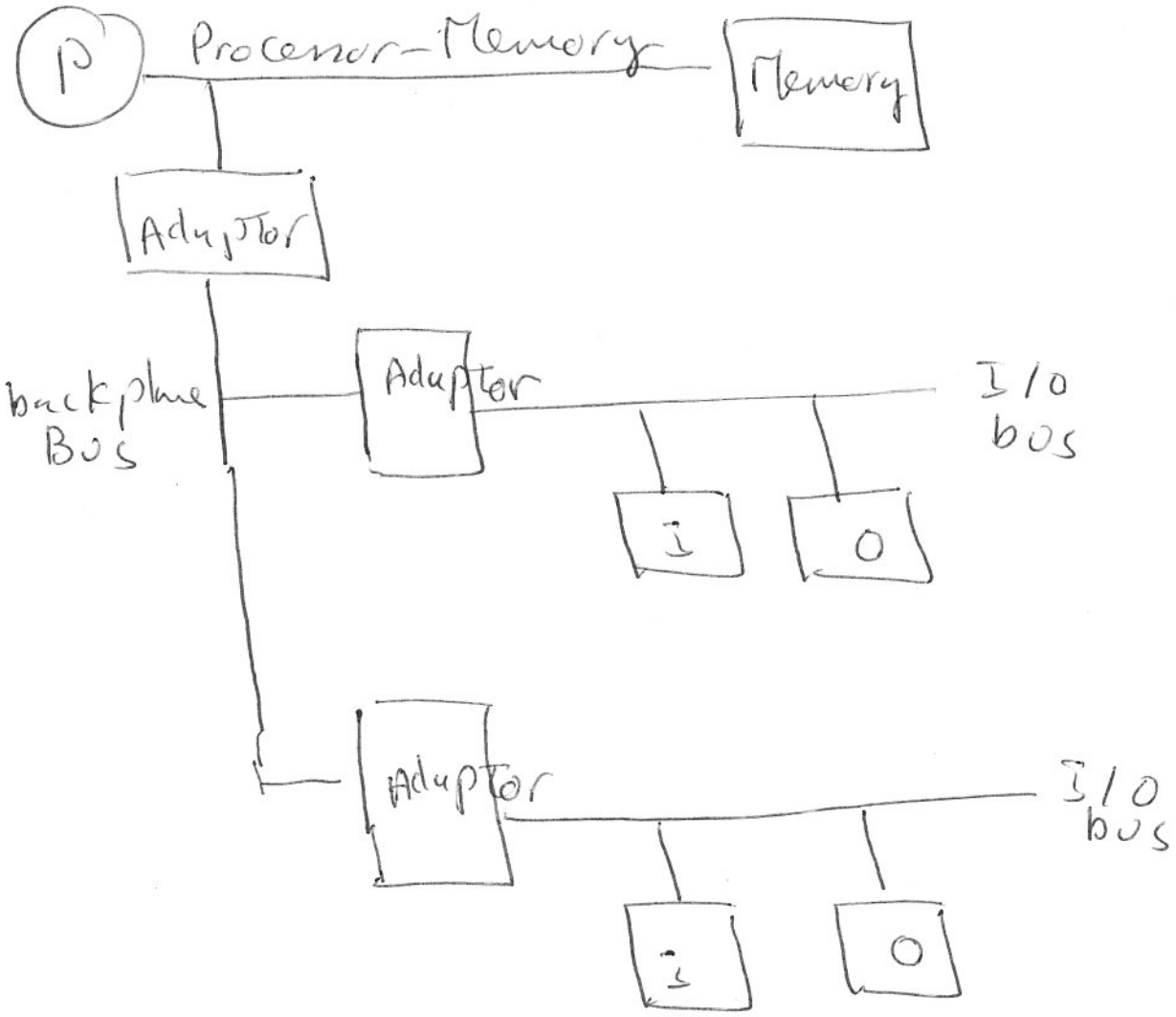
2- Two bus organization



Adaptor: Keep different speed of two bus system

Example: APPle - Macintosh
- NUBUS for procesor-memory
- SCSI BUS for I/O

Three bus organization



Example: IBM RS1600.

Silicon Graphics multiprocessors

Bus protocols and schemes

two types:

1- Synchronous: uses clock in control lines, fixed protocol relative to clock.

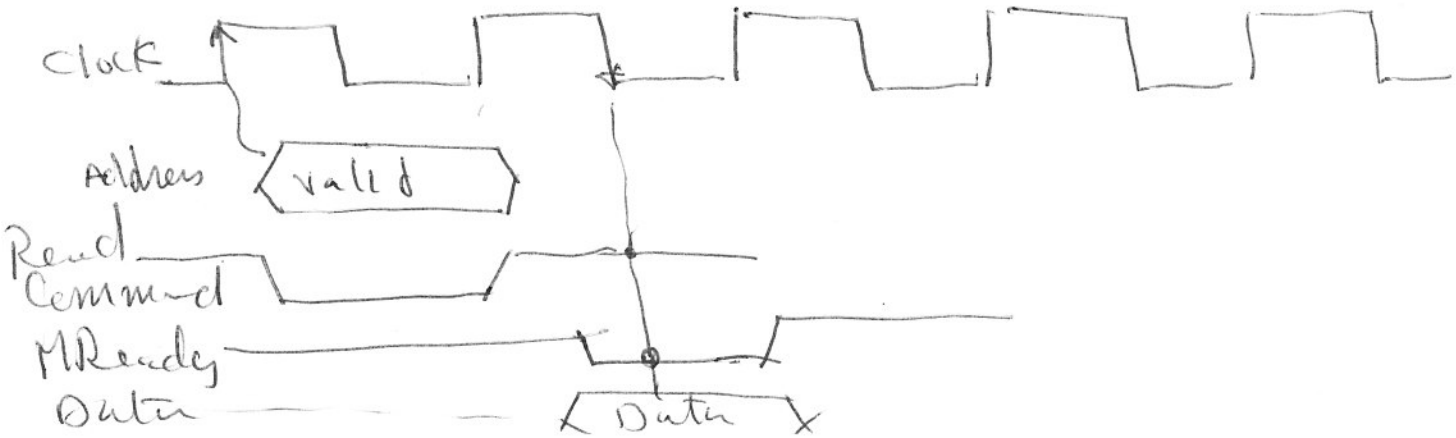
Example: clock #1, Assert read command and Address
- wait for data ready
- clock #5, Data valid

Implementation: use state machine

Advantage: - very fast
- very little logic

disadvantages: - Every device must run at same clock speed
- short, to Avoid clock skew

Example processor - memory bus
Memory Read operation



Type 2: Asynchronous Protocol

- No clock
- Accommodate wide range of devices
- Long (no clock skew)
- Needs handshaking protocol

Control signals:

1 - Read Req: when asserted, it indicates need request for memory and data lines have valid Address.

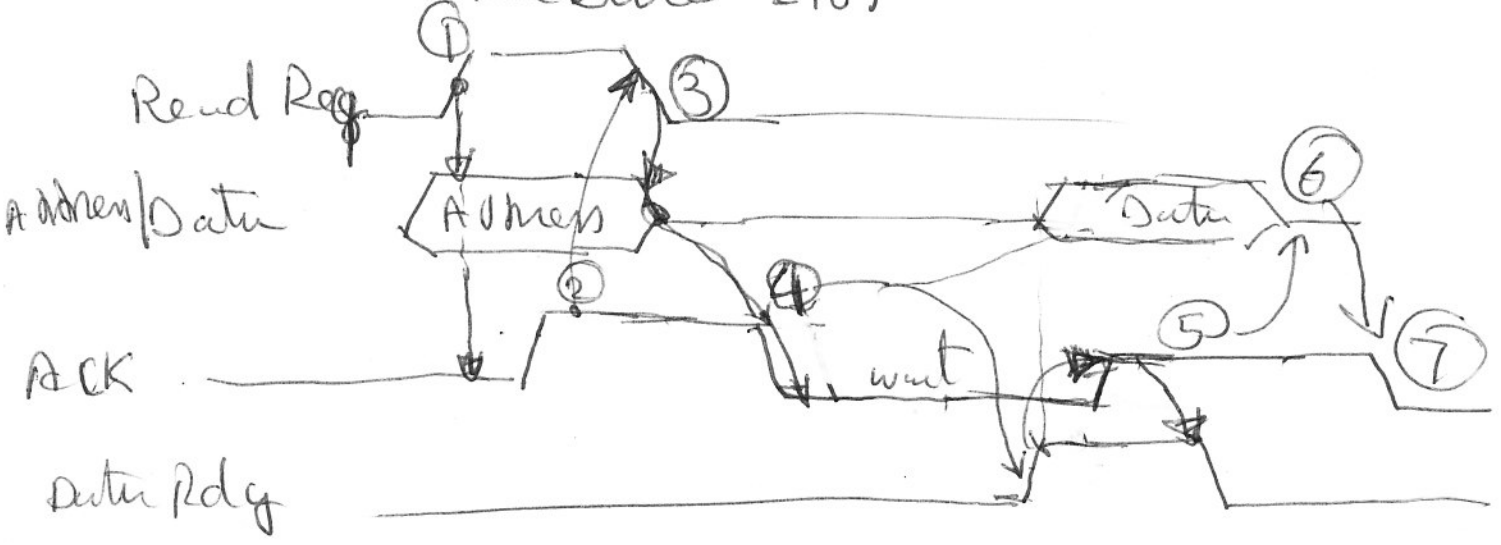
2 - Data Rdy: when asserted, data lines have a valid data

3 - Ack: acknowledge Read Req or Data Rdy.

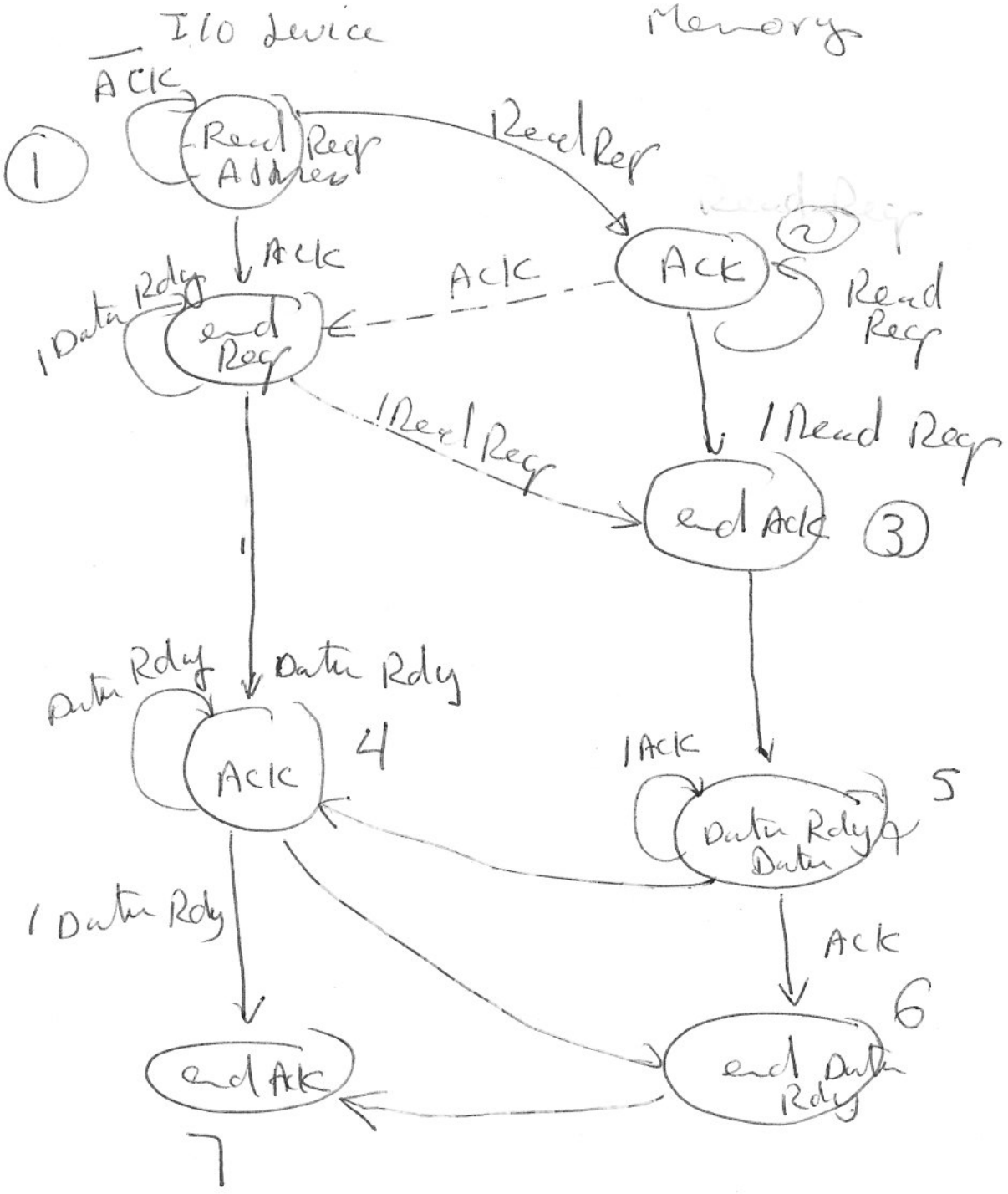
Protocol uses handshaking

means control are asserted until other party report to it (ACK)

Example: output operation (read memory and receive I/O)



State Machine for Asynchonous Protocol

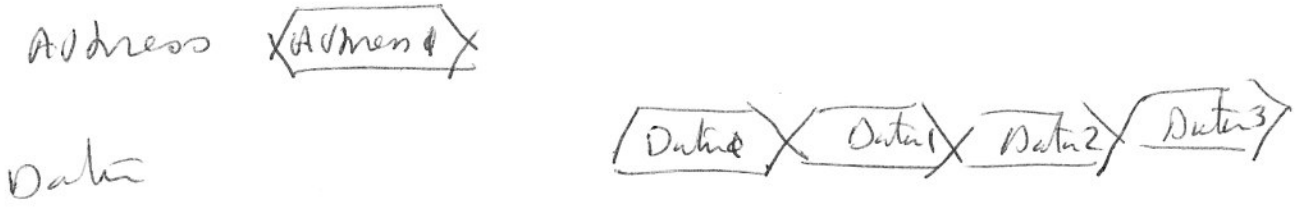


Methods to increase Bus Bandwidth

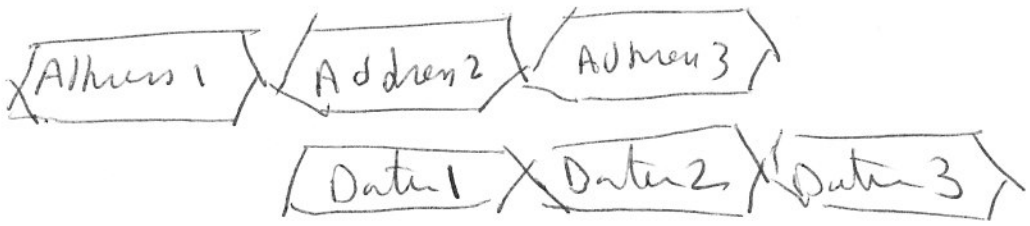
1- By increasing Data bus width, Transfer multiple words. It cost more bus lines

2- By using separate, not multiplexed (Address/data) lines. it costs more and it increases complexity

3- By using Block Transfers:
Bus Transfers multiple words without sending an Address for each word.
Only send first Address



4- Using Split Transaction / Pipelined bus.
while waiting for Data, send next address



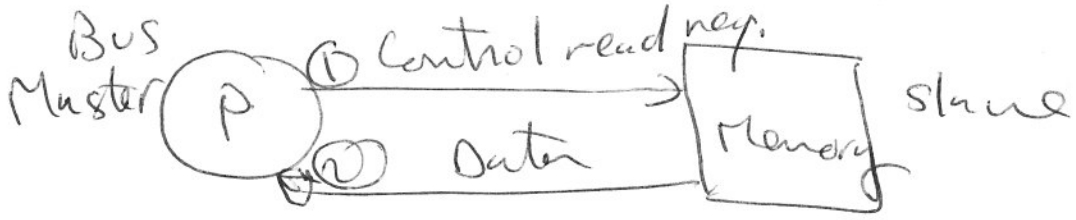
Bus Arbitration
obtaining Access to the Bus

- Need a scheme to control right to access the Bus.
- without a scheme, multiple devices try to use the Bus at the same time causing conflict.

Master - slave scheme

scheme: only Master has the right to access the bus.
- It initiates and controls all bus requests
- A slave responds to read and write requests

Example: Processor - memory system.
Processor is the bus Master. it initiates bus requests to memory.
Memory is a slave and only responds to read/or write requests.



single Bus Master: Processor is the only bus Master.
- All requests are controlled by the processor

Disadvantage: Processor is involved in each transaction.

Multiple Bus Masters
Need Arbitration

Arbitration: to decide which device is the next Bus Master

Bus Arbitration scheme:

1- Device Wanting to use the bus asserts bus request

2- It waits for bus grant, after a grant, The device can use the bus and becomes the bus Master.

3- It Must signal to the arbiter after finish using the bus.

Factors important to Bus Arbitration:

1- Highest priority device should be serviced first (Priority)

2- Even The lowest priority device should be Able to get the Bus (Fairness)

Different Arbitration Schemes

1- Daisy Chain Arbitration

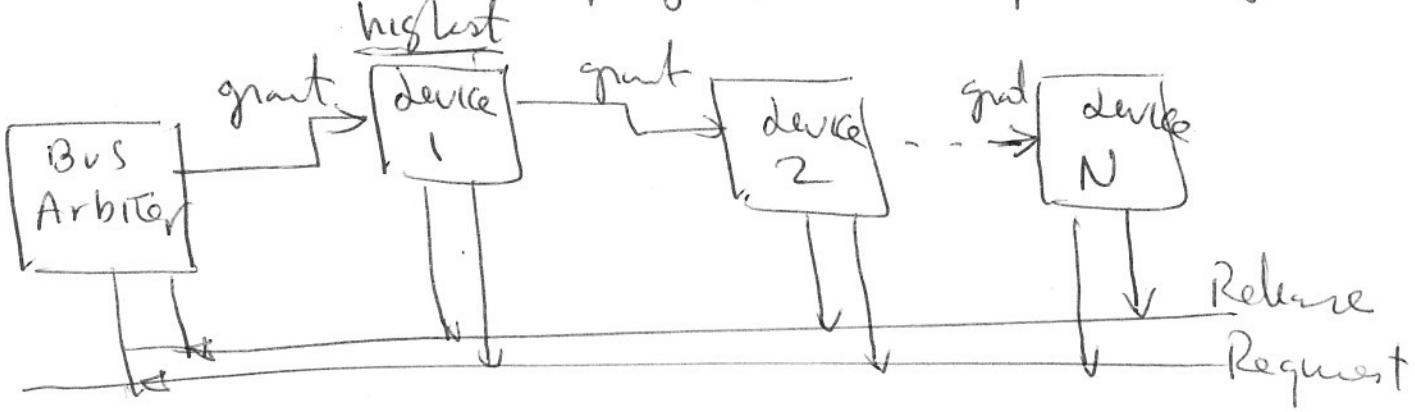
Priority is determined by how close is the device to arbiter

- Operation one bus grant line runs through devices, grant is forwarded to next device if not used.

Advantages: Simple

Disadvantages: 1- Low priority devices (far from arbiter) could be locked out.

2- bus speed is limited by propagation bus grant signal.

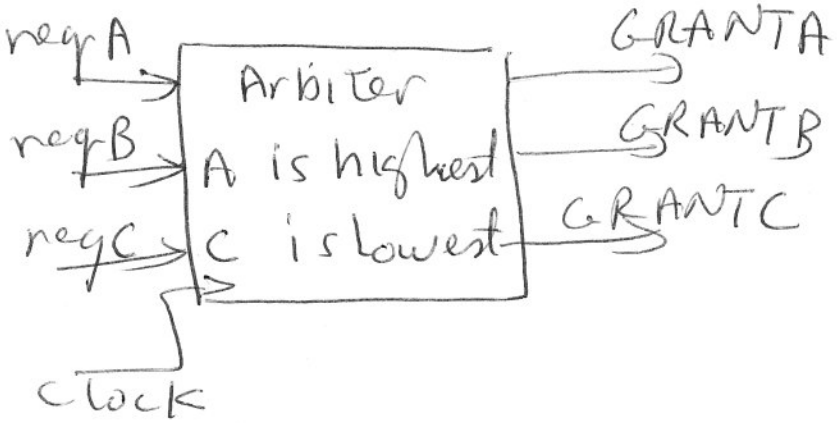


Example If Device 1 is bus Master

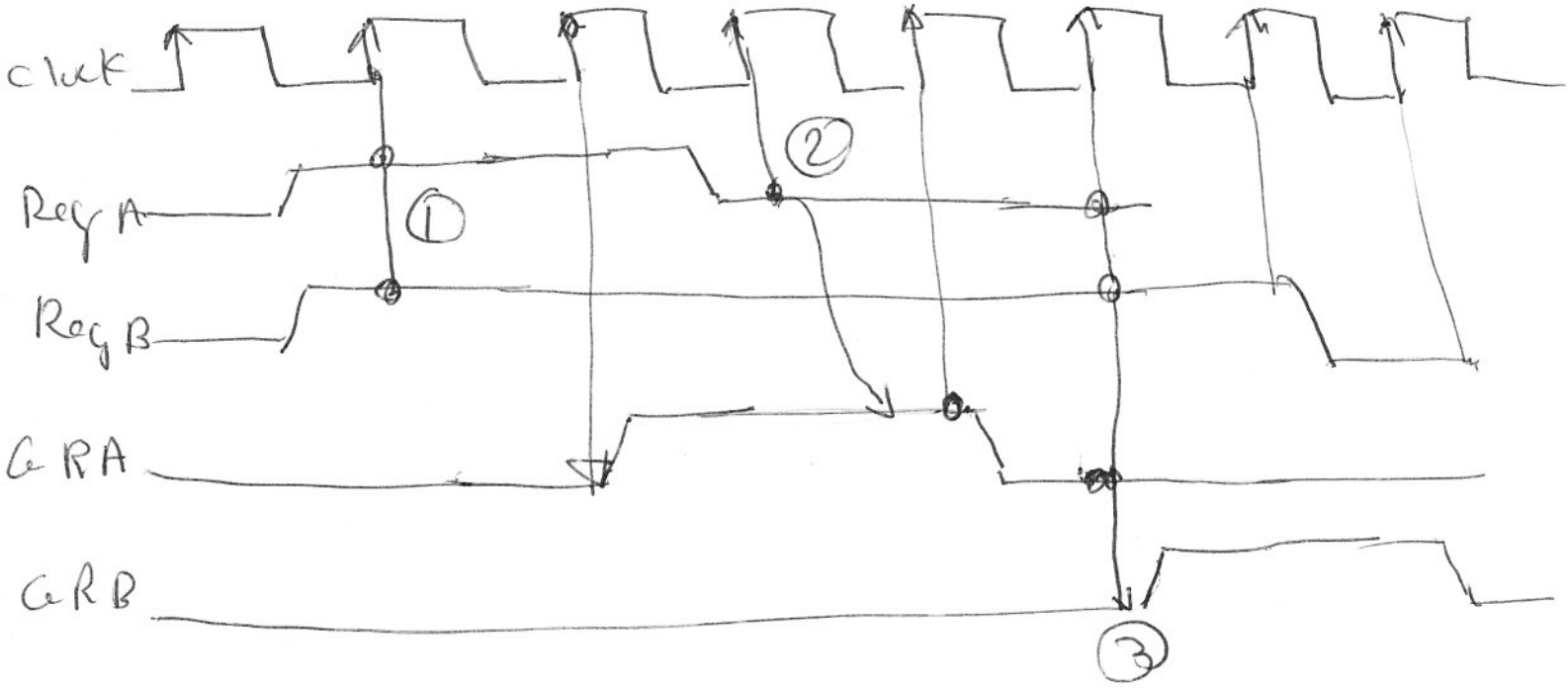
- Device 3 wants the bus
- it waits for bus grant
- Device 1, finish and asserts Release signal, and output grant signal to Device 2
- Device 2, asserts grant signal To Device 3

2 - Centeralized Arbitration

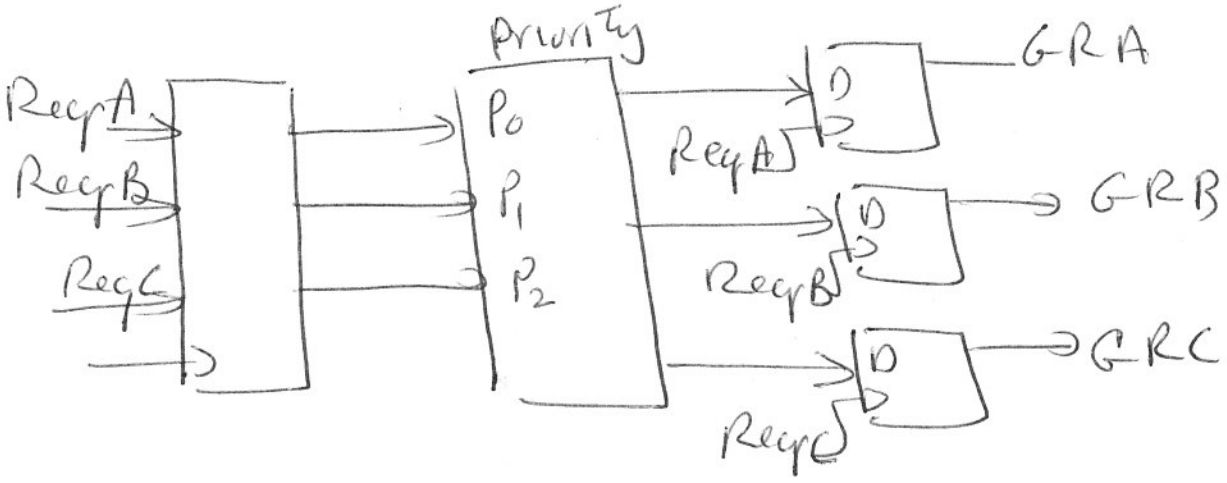
- Multiple requests from devices (parallel arbitration)
- Arbiter chose one device based on priority



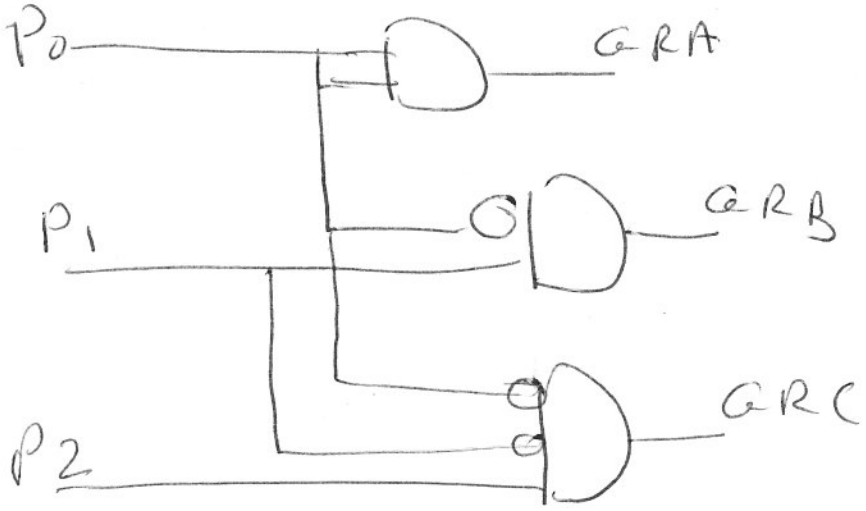
operation device asserts request, waits grant from arbiter



Implementation :-



Priority



3 - Distributed Arbitration

Devices determine who will be granted access. Each device place a code indicating its identity and device can determine the highest priority device.

No central arbitration, need request lines

Example NUBUS for Apple Macintosh

Bus standards

- Bus standards serve as a specification for computer and peripheral manufacturers.
- it ensures that peripherals will be available for a new machine and new peripherals that it will work in any machine

Characteristics of different Bus standards

OPTION	High Performance	Low Cost
Bus width	separate address and data	Multiplexed Addr/Data
Data width	Multiple words (32bits)	Narrow (byte)
Bus Master	Multiple	Single
clocking	Synchronous	Asynchronous

Different Bus standards

Characteristic	VME	NUBUS	PCI	SCSI
Bus type	Backplane	Backplane	Backplane	I/O
Bus width	128	96	50	8
Addr/data	16-32 bits	32 bits	32-64	8
Arbitration	daisy Chain	Distributed	Centralized	self sel.
clocking	Asynchr.	Synchr.	Synchr.	either
Bandwidth	12.4 MB/sec	13.2 MB/sec	132 MB/sec	5 MB/s

PCI BUS

STRUCTURE:

System Pins: clock and reset

Address and Data Pins: 32 lines, Multiplexed

Control pins: Control timing and coordination

Arbitration Pins: Each PCI Master has 2 pins
REQ, GRNT (for centeralize)

Interrupt pins: Each device has its own line

Pin	Type	Function
		(System)
clk	Input	Provide Timing on posedge = 33 MHz
Rst	Input	Forces all PCI registers, control to initial state
		(Address/Data)
AD[31:0]	I/O/Z	Multiplexed bus for Address / Data
C/BE[3:0]	I/O/Z	Multiplexed bus Command and byte enable
		(Control)
FRAME	Z, out	Driven by current bus Master, indicate start of Transfer

Pin	Type	Function
IRDY	Z, out	Driven by current bus Master, indicate that initiator Ready To accept data for read operation or has valid data for write operation
TRDY	Z /out	driven by Target To indicate that it has valid data for read operation or it is ready to accept data for write operation
IDsel	IN	used to initialize chip on configuration
		(Arbitration)
REQ	I/O/Z	indicates that device requires the Bus
GRNT	I/O/Z	indicates that arbiter has granted the device bus access
		(Interrupt pins)
INTA	wired OR open drain	request an interrupt

Interfacing Processor and Peripherals

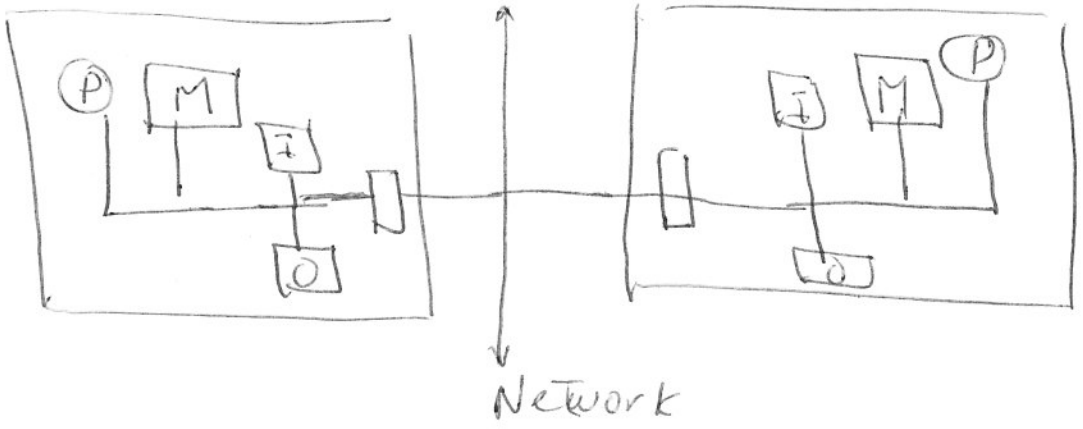
- Introduction
- I/O performance measure
- Types of I/O Devices

Introduction

- I/O Design for:

1- Performance (Latency or Throughput)
2- expandability
3- Resilience in face of failure

Processor Design only for performance



Caches and I/O

Problem with using caches with I/O, because of possibility of having two copies of data, one in main memory and one for cache.

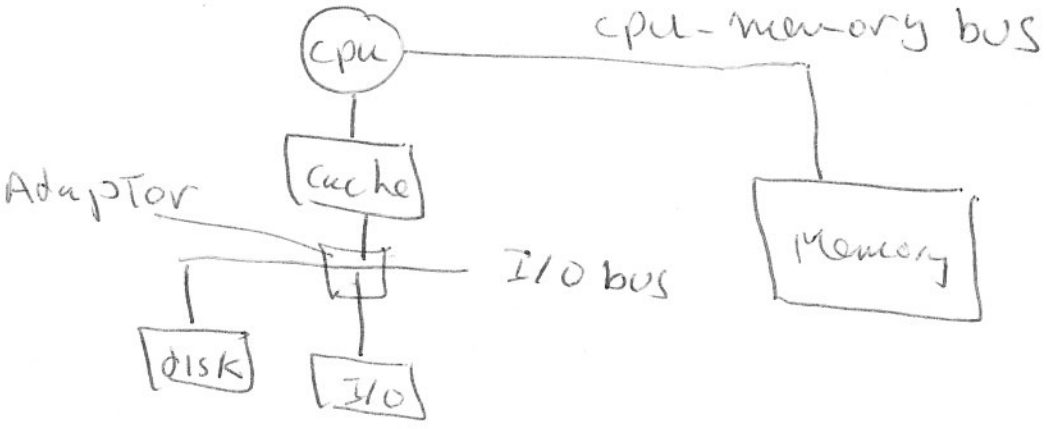
Virtual memory there is 3 levels of memory and could have 3 copies in cache, memory and hard disk.

The problem: cpu or I/O could modify one copy without updating other copies (stale data).

Solution: 1- CPU must needs most recently input data

2- output devices must have the correct data

Implementation: connecting I/O to cpu cache



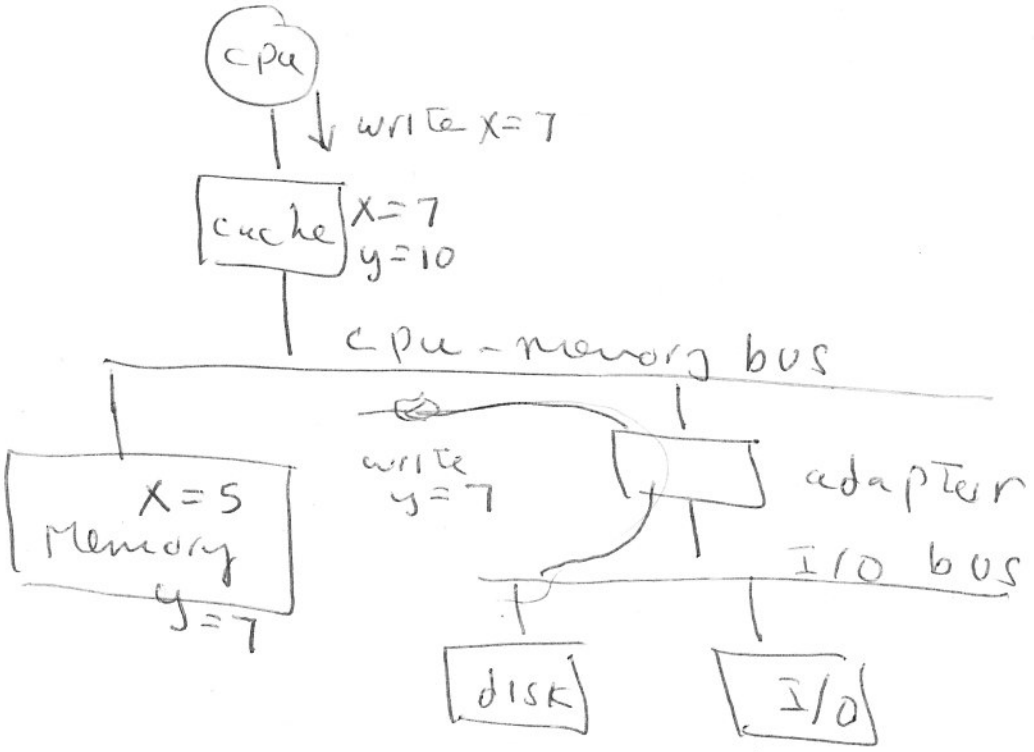
cpu and I/O will see most recently values of data. (no coherency problem)

Disadvantages with Connecting I/O To cpu cache?

1- when I/O access cache, cpu must not access the cache and wait reducing cpu performance

2- I/O and cpu must arbiterate for cache (Complicated, slow)

Connecting I/O to memory:



It does not interfere with cpu but stale data problem could occur.

Conditions for possible stale data:

1- If cpu writes data to cache and does not update Main memory

2- I/O writes data to memory, and does not update cache.

Solutions:

First problem: -Cache have data that is not coherent with memory can be solved if we use write through (slow)

- can use operating system with write back ~~cache~~ to flush cache (Takes Time)

second problem:- ~~o/p~~ I/O changes data in memory, making cache data stale.

- use operating system to forbid caching of I/O input address in memory (Takes Time)

Designing I/O System

issues

- cost
- performance bottleneck
- expandability

steps to design I/O systems:-

1- List all I/O devices to be connected and standard bus that will be used

2- List physical requirements for I/O devices (volume, power, connectors, ...)

3- List the cost of each I/O device including controllers

4- Find cpu demands to support I/O
- I/O instructions (clock cycles, handling interrupts).
- cpu clock cycles while waiting for bus or I/O
- cpu clock cycles to recover from I/O, (flush cache).

5- List memory and I/O bus demands for each I/O device (bandwidth)

6- use simulation and queuing theory to evaluate performance.

optimize design for cost/performance

Operating system Requirements to support I/O

1- Protection to shared I/O resources (only access I/O device that user has rights to use)

2- Provide Abstraction (user does not need to know device details)

3- OS handles interrupts used by I/O devices

4- Provide equitable access to shared I/O

5- Schedule accesses to enhance throughput

Communication between OS and I/O

1- From OS to I/O

- special I/O instructions with device number and command word using processor data bus
- Memory - Mapped I/O
  Portion of address space are assigned to I/O devices, and perform Read/write as memory accesses. Protection is done by the translation table.

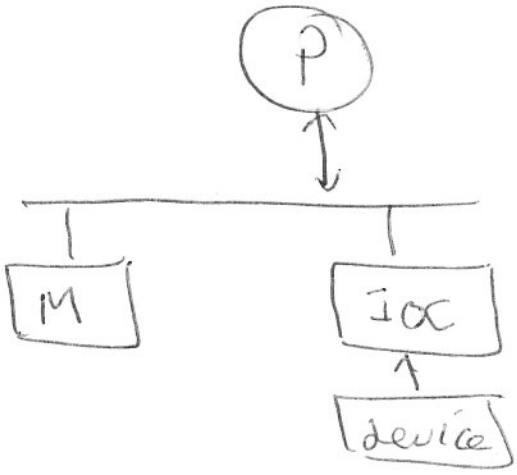
2- From I/O to OS:

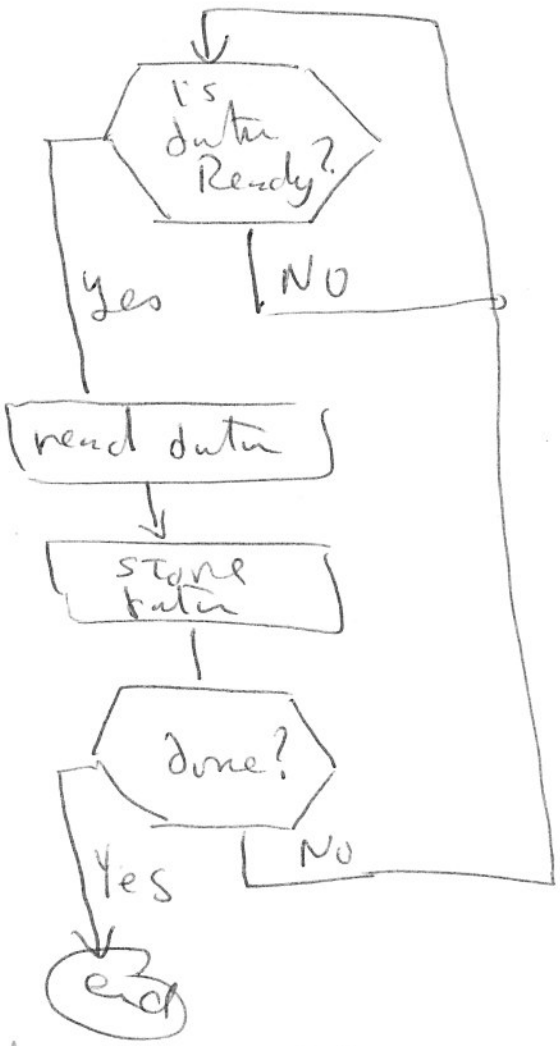
1- Polling:

I/O device put information in status register, and OS periodically check status Reg.

Advantages: - simple, Processor is in Total Control

Disadvantages: Consume of cpu valuable Time





Example: If number of clock cycles per polling operation = 100 cycles, clock is 50 MHz.

find % cpu Time of the following

1- Mouse Polling 30 Times per sec.

2- Floppy disk with 16 bits and data rate of 50KB/s

3- H.D with 1word and 2 M/sec.

Solution

1- Mouse # cycles per sec = 30 x 100

$$\%_{cpu} = \frac{3000}{50 \times 10^6} \times 100 = .006\%$$ okay

2- Floppy # of operations $\frac{50\,000 \text{ KB/s}}{2B}$

$= 25{,}000$ operations/s

# of cycles = 25,000 x 100

$$\%_{cpu} = \frac{25{,}000{,}00 \times 100}{50 \times 10^6} = 5\% \; ??$$

3- H.D # of operations/s $= \frac{2 \times 10^6}{4B} = .5 \times 10^6$

# of cycles $= .5 \times 10^6 \times 100$

$$\text{cpu time} = \frac{50 \times 10^6}{50 \times 10^6} = 100\%$$

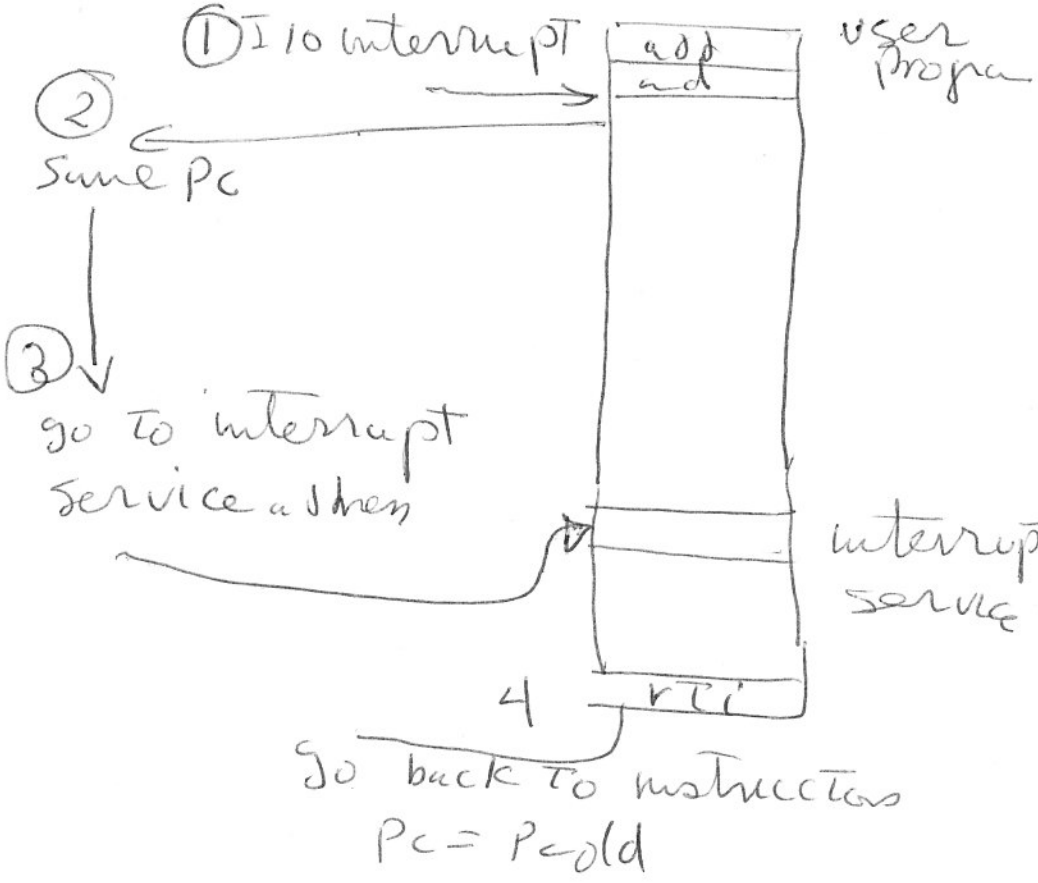
Method 2 — Interrupts

Motivation: reduce cpu waiting time

Features: 1- Asynchronous with respect to instructions execution

2- Need to know device identity and priority

operation: when ready device interrupt processor, send vector address, or status field in cause register



Performance:

Example: If Floppy disk has 50KB/sec and 16bit, uses interrupts that has 100 clock cycles overhead, find % cpu time if Floppy is busy 50% of the time.

Solution

number of transaction/sec = $\frac{50{,}000 \text{ KB/sec}}{2 \text{ B}} = 25{,}000$

% cpu time = $\frac{25{,}000 \times 100}{50 \times 10^6} \times 100 = 5\%$

only 10% will be used from cpu = 0.5%

DMA

Motivation: For high bandwidth devices like H.D overhead of using interrupts still intolerable

Concept: The device controller transfer data directly to/from memory without processor involvement. Interrupt is used on completion of transfer.

Operation:

1- Processor sets DMA with operation (Read/Write), memory Address and number of byte to transfer

2- DMA arbiterates for the bus. when it gets the bus, DMA supplies memory Address and Operation, DMA generates next address and could use some buffering for data (FIFO)

3- Once Transfer is complete, Controller interrupts the processor.

Performance of DMA

Hard disk 2 MB/sec, 50 MHz CPU speed, DMA setup = 1000 cycles, interrupt = 500 cycles.

Transfer size = 4 KB.

Solution

Time of transfer

$$= \frac{4\,KB}{2\,MB} = 2 \times 10^{-3}$$

Cost of transfer = 1000 + 500 = 1500 cycles

$$= \frac{1500}{50 \times 10^6} = 30 \times 10^{-6}\ sec$$

% CPU use = $\frac{30 \times 10^{-6}}{2 \times 10^{-3}} \times 100 = 1.5\%$

DMA and Coherency

DMA could change contents of memory locations that are in CPU cache creating coherency problem.

Solution: OS can flush cache.

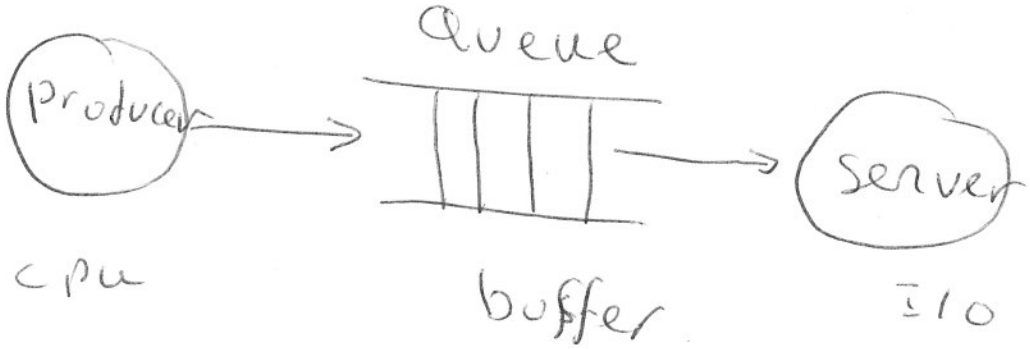
I/O Performance Measures

Performance:

1- response Time (latency)
2- thoughput (bandwidth)

Producer Server Model

To model response Time and Throughput of I/O systems.



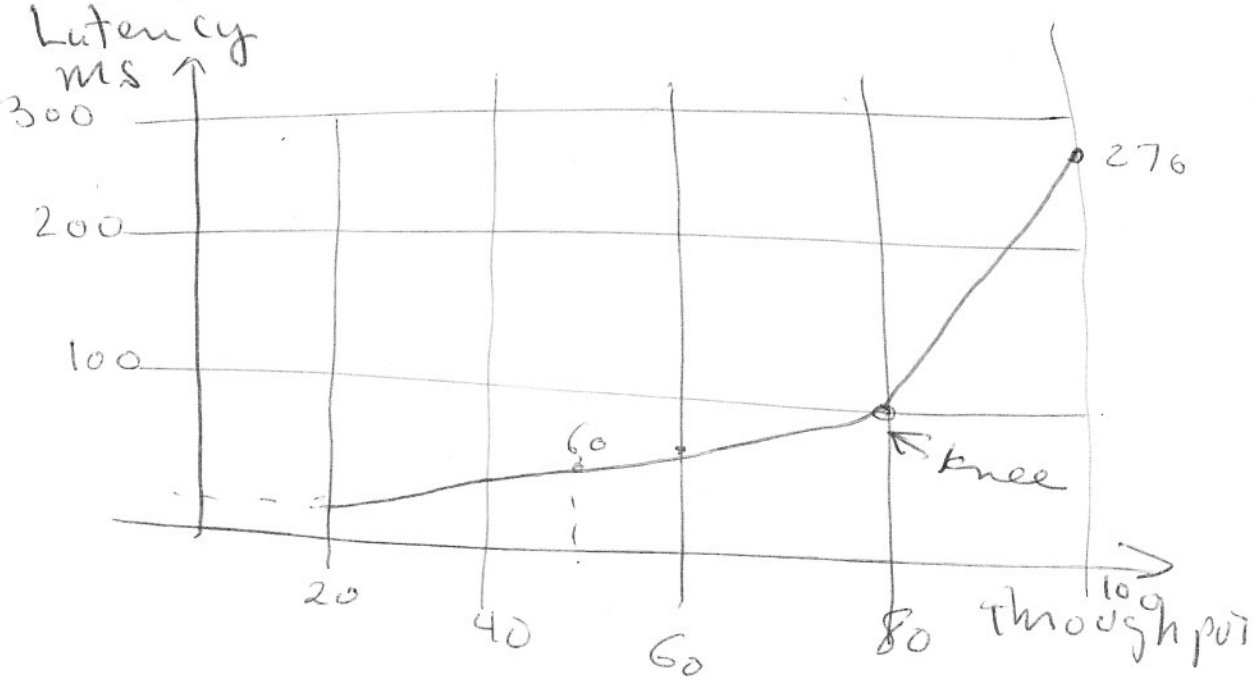
Producer: produces Tasks and places them in buffer

Server: Performs Tasks, Take them from buffer.

Response Time: Time a Task Take from the moment it is placed on buffer until Server finishes the Task.

Thoughput: Average number of tasks completed by server over a Time Period.

Throughput versus Response Time

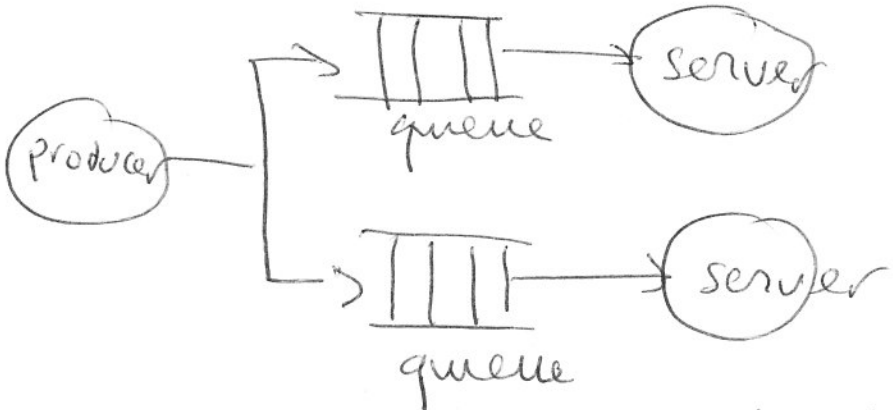


for 50% reduction in throughput, latency is improved by 4 times.

For minimum response time, we can get only 11% of throughput.

Improving throughput does not necessary improve response time.

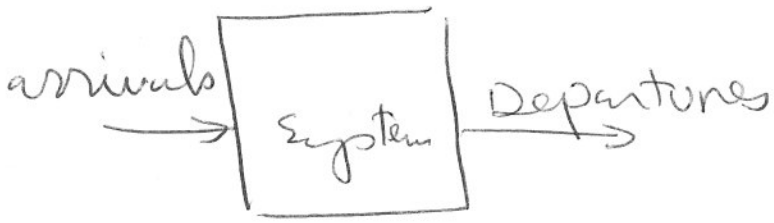
Improving throughput by adding more servers.



Load must be balanced, divided in parallel.

Little Queuing theory

To model a computer system.
CPU is producer, server is I/O or memory unit service those requests.

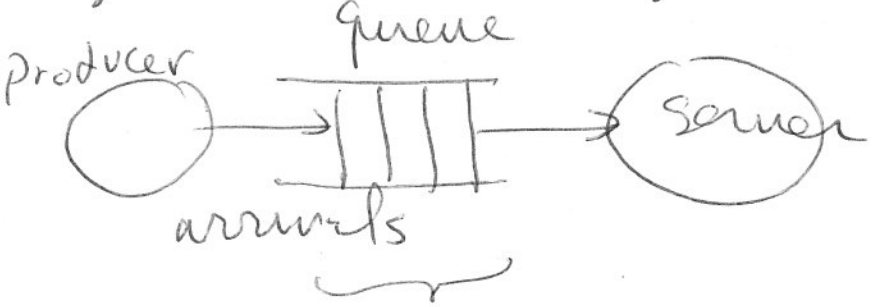


on steady state:

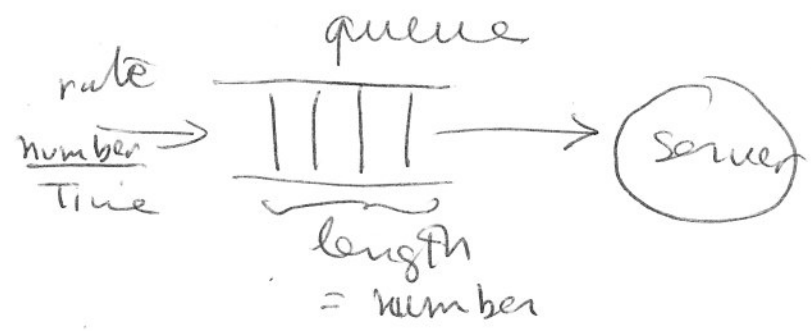
Number of tasks entering system
= number of tasks leaving the system (1)

Average number of tasks in system
= Arrival rate x Mean response Time (2)

Little's Law system in equilibrium (no tasks are generated or destroyed inside system).



queue is needed for tasks to wait until they are served.



Time_server = Average Time To Service Task

$$\text{Service rate} = \frac{1}{\text{Time\_server}}$$ (one task at a Time)

Time_queue = Average Time per Task in queue

Time_system = Average Time/Task (response Time)

= Time_queue + Time_server

Arrival_rate = average number of tasks arriving / per second

Length_server = Average number of Tasks in service

Length_queue = Average number of tasks in queue.

Length_system = average number of tasks in system = Length_queue + Length_server

Length_system = Arrival rate X Time system

$$\text{Server Utilization} = \frac{\text{Arrival rate}}{\text{service rate}}$$

From 0 → 1 (equilibrium)

Example:

Assume an I/O system gets 10 requests per second, and the average time for a disk to service an I/O is 50 ms.

Find the utilization of I/O System.

$$\text{utilization} = \frac{\text{arrival rate}}{\text{service rate}}$$

$$\text{service rate} = \frac{1}{\text{Time of service}} = \frac{1}{50\text{ ms}}$$

$$= 20 \text{ Tasks/Sec}$$

$$\text{utilization} = \frac{10}{20} = 50\%$$

Length of Server = arrival rate x Time of service

Example: find mean number of I/O at the server if I/O arrival rate is 200/s and service time = 50 ms.

Length of Server = arrival rate x Time of service

$$= \frac{200}{s} \times \frac{50}{1000} = 10$$

average 10 Tasks at Server.

Time of the queue = $\text{Time}_{\text{server}} \times \frac{\text{Server utilization}}{1 - \text{Server utilization}}$

Sever utilization = $\frac{\text{Arrival rate}}{\text{service rate}}$

Time of queue = $\text{Time}_{\text{server}} \times \frac{\text{Arrival rate}}{\text{service rate} - \text{Arrival rate}}$

Example: Processor sends 10 I/O per second, average disk service time = 20ms find queue Time.

Solution:

Arrival rate = 10

service rate = $\frac{1}{20\text{ms}} = 50$

Time of queue = $20\text{ms} \times \frac{10}{50 - 10}$

$= 5\text{ms}$

FSM Control for I/O

Example Show how the control for an output transaction to an I/O device from memory (as in Figure 8.7) can be implemented as a pair of finite state machines.

Answer Figure 8.11 shows the two finite state machine controllers that implement the handshaking protocol of Figure 8.10.

If a synchronous bus can be used, it is usually faster than an asynchronous bus because of the overhead required to perform the handshaking. An example demonstrates this.

Performance Analysis of Synchronous versus Asynchronous Buses

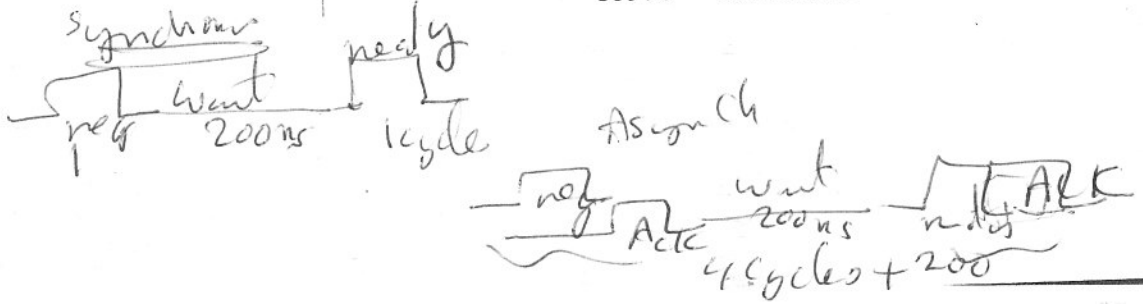
Example We want to compare the maximum bandwidth for a synchronous and an asynchronous bus. The synchronous bus has a clock cycle time of 50 ns, and each bus transmission takes 1 clock cycle. The asynchronous bus requires 40 ns per handshake. The data portion of both buses is 32 bits wide. Find the bandwidth for each bus when performing one-word reads from a 200-ns memory.

Answer First, the synchronous bus, which has 50-ns bus cycles. The steps and times required for the synchronous bus are as follows:

1. Send the address to memory: 50 ns
2. Read the memory: 200 ns
3. Send the data to the device: 50 ns

Thus, the total time is 300 ns. This yields a maximum bus bandwidth of 4 bytes every 300 ns, or

$$\frac{4 \text{ bytes}}{300 \text{ ns}} = \frac{4 \text{ MB}}{0.3 \text{ seconds}} = 13.3 \frac{\text{MB}}{\text{second}}$$



At first glance, it might appear that the asynchronous bus will be *much* slower, since it will take seven steps, each at least 40 ns, and the step corresponding to the memory access will take 200 ns. If we look carefully at Figure 8.10, we realize that several of the steps can be overlapped with the memory access time. In particular, the memory receives the address at the end of step 1 and does not need to put the data on the bus until the beginning of step 5; steps 2, 3, and 4 can overlap with the memory access time. This leads to the following timing:

Step 1: 40 ns

Steps 2, 3, 4: maximum (3 x 40 ns, 200 ns) = 200 ns

Steps 5, 6, 7: 3 x 40 ns = 120 ns

Thus, the total time to perform the transfer is 360 ns, and the maximum bandwidth is

$$\frac{4 \text{ bytes}}{360 \text{ ns}} = \frac{4 \text{ MB}}{0.36 \text{ seconds}} = 11.1 \frac{\text{MB}}{\text{second}}$$

Accordingly, the synchronous bus is only about 20% faster. Of course, to sustain these rates, the device and memory system on the asynchronous bus will need to be fairly fast to accomplish each handshaking step in 40 ns.

Even though a synchronous bus may be faster, the choice between a synchronous and an asynchronous bus has implications not only for data bandwidth but also for an I/O system's capacity in terms of physical distance and the number of devices that can be connected to the bus. Asynchronous buses scale better with technology changes and can support a wider variety of device response speeds. It is for these reasons that I/O buses are often asynchronous, despite the increased overhead.

Increasing the Bus Bandwidth

Although much of the bandwidth of a bus is decided by the choice of a synchronous or asynchronous protocol and the timing characteristics of the bus, several other factors affect the bandwidth that can be attained by a single transfer. The most important of these are the following:

1. *Data bus width*: By increasing the width of the data bus, transfers of multiple words require fewer bus cycles.
2. *Separate versus multiplexed address and data lines*: Our example in Figure 8.8 used the same wires for address and data; including separate lines for addresses will make the performance of writes faster because the address and data can be transmitted in one bus cycle.

Performance Analysis of Two Bus Schemes

Example Suppose we have a system with the following characteristics:

1. A memory and bus system supporting block access of 4 to 16 32-bit words.
2. A 64-bit synchronous bus clocked at 200 MHz, with each 64-bit transfer taking 1 clock cycle, and 1 clock cycle required to send an address to memory.
3. Two clock cycles needed between each bus operation. (Assume the bus is idle before an access.)
4. A memory access time for the first four words of 200 ns; each additional set of four words can be read in 20 ns. Assume that a bus transfer of the most recently read data and a read of the next four words can be overlapped.

Find the sustained bandwidth and the latency for a read of 256 words for transfers that use 4-word blocks and for transfers that use 16-word blocks. Also compute the effective number of bus transactions per second for each case. Recall that a single bus transaction consists of an address transmission followed by data.

Answer For the 4-word block transfers, each block takes

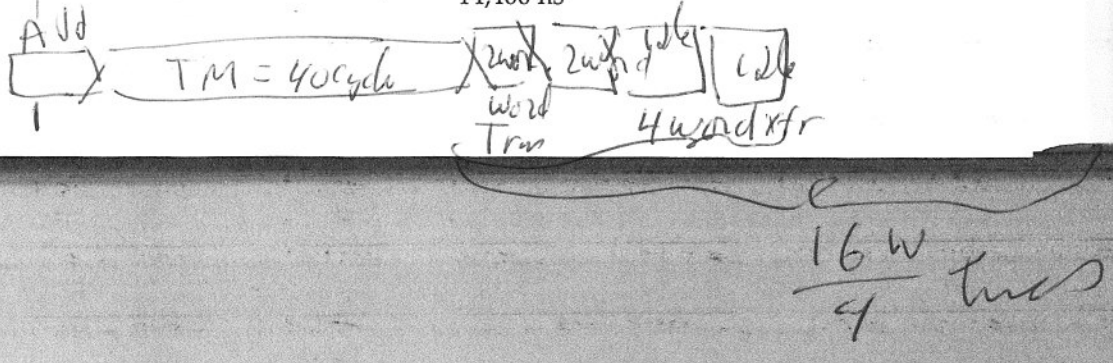
1. 1 clock cycle that is required to send the address to memory
2. $\frac{200 \text{ ns}}{5 \text{ ns/cycle}} = 40$ clock cycles to read memory
3. 2 clock cycles to send the data from the memory
4. 2 idle clock cycles between this transfer and the next

This is a total of 45 cycles, and 256/4 = 64 transactions are needed, so the entire transfer takes 45 × 64 = 2880 clock cycles. Thus the latency is 2880 cycles × 5 ns/cycle = 14,400 ns. The number of bus transactions per second is

$$64 \text{ transactions} \times \frac{1 \text{ second}}{14{,}400 \text{ ns}} = 4.44\text{M transactions/second}$$

The bus bandwidth is

$$(256 \times 4) \text{ bytes} \times \frac{1 \text{ second}}{14{,}400 \text{ ns}} = 71.11 \text{ MB/sec}$$



For the 16-word block transfers, the first block requires

1. 1 clock cycle to send an address to memory
2. 200 ns or 40 cycles to read the first four words in memory
3. 2 cycles to send the data of the block, during which time the read of the four words in the next block is started
4. 2 idle cycles between transfers and during which the read of the next block is completed

Each of the three remaining 4-word blocks requires repeating only the last two steps.

Thus, the total number of cycles for each 16-word block is 1 + 40 + 4 × (2 + 2) = 57 cycles, and 256/16 = 16 transactions are needed, so the entire transfer takes, 57 × 16 = 912 cycles. Thus the latency is 912 cycles × 5 ns/cycle = 4560 ns, which is roughly one-third of the latency for the case with 4-word blocks. The number of bus transactions per second with 16-word blocks is

$$16 \text{ transactions} \times \frac{1 \text{ second}}{4560 \text{ ns}} = 3.51\text{M transactions/second}$$

which is lower than the case with 4-word blocks because each transaction takes longer (57 versus 45 cycles).

The bus bandwidth with 16-word blocks is

$$(256 \times 4) \text{ bytes} \times \frac{1 \text{ second}}{4560 \text{ ns}} = 224.56 \text{ MB/second}$$

which is 3.16 times higher than for the 4-word blocks. The advantage of using larger block transfers is clear.

Elaboration: Another method for increasing the effective bus bandwidth when multiple parties want to communicate on the bus is to release the bus when it is not being used for transmitting information. Consider the example of a memory read that we examined in Figure 8.10. What happens to the bus while the memory access is occurring? In this simple protocol, the device and memory continue to hold the bus during the memory access time when no actual transfer is taking place. An alternative protocol, which releases the bus, would operate like this:

1. The device signals the memory and transmits the request and address.
2. After the memory acknowledges the request, both the memory and device release all control lines.
3. The memory access occurs, and the bus is free for other uses during this period.

Communicating with the Processor

The process of periodically checking status bits to see if it is time for the next I/O operation, as in the previous example, is called *polling*. Polling is the simplest way for an I/O device to communicate with the processor. The I/O device simply puts the information in a Status register, and the processor must come and get the information. The processor is totally in control and does all the work.

The disadvantage of polling is that it can waste a lot of processor time because processors are so much faster than I/O devices. The processor may read the Status register many times, only to find that the device has not yet completed a comparatively slow I/O operation, or that the mouse has not budged since the last time it was polled. When the device has completed an operation, we must still read the status to determine whether it was successful.

Polling can be used in several different ways, depending on the I/O device and whether the I/O device can initiate I/O independently. For example, a mouse is an input-only device that initiates I/O independently, when a user moves the mouse or clicks a button. Because a mouse has a low I/O rate, polling is often used to interface to a mouse. Many other I/O devices, such as a floppy disk or a printer, initiate I/O only under control of the operating system. Thus we need only poll such devices when the OS knows that the device is active. As we will see, this allows polling to be used even when the I/O rate is somewhat higher.

Overhead of Polling in an I/O System

Example Let's determine the impact of polling overhead for three different devices. Assume that the number of clock cycles for a polling operation—including transferring to the polling routine, accessing the device, and restarting the user program—is 400 and that the processor executes with a 500-MHz clock.

Determine the fraction of CPU time consumed for the following three cases, assuming that you poll often enough so that no data is ever lost and assuming that the devices are potentially always busy:

1. The mouse must be polled 30 times per second to ensure that we do not miss any movement made by the user.
2. The floppy disk transfers data to the processor in 16-bit units and has a data rate of 50 KB/sec. No data transfer can be missed.
3. The hard disk transfers data in four-word chunks and can transfer at 4 MB/sec. Again, no transfer can be missed.

Transferring the Data between a Device and Memory

We have seen two different methods that enable a device to communicate with the processor. These two techniques, polling and I/O interrupts, form the basis for two methods of implementing the transfer of data between the I/O device and memory. Both these techniques work best with lower-bandwidth devices, where we are more interested in reducing the cost of the device controller and interface than in providing a high-bandwidth transfer. Both polling and interrupt-driven transfers put the burden of moving data and managing the transfer on the processor. After looking at these two schemes, we will examine a scheme more suitable for higher-performance devices or collections of devices.

We can use the processor to transfer data between a device and memory based on polling. Consider our mouse example. The processor can periodically read the mouse counter values and the position of the mouse buttons. If the position of the mouse or one of its buttons has changed, the operating system can notify the program associated with interpreting the mouse changes.

An alternative mechanism is to make the transfer of data interrupt driven. In this case, the OS would still transfer data in small numbers of bytes from or to the device. But because the I/O operation is interrupt driven, the OS simply works on other tasks while data is being read from or written to the device. When the OS recognizes an interrupt from the device, it reads the status to check for errors. If there are none, the OS can supply the next piece of data, for example, by a sequence of memory-mapped writes. When the last byte of an I/O request has been transmitted and the I/O operation is completed, the OS can inform the program. The processor and OS do all the work in this process, accessing the device and memory for each data item transferred. Let's see how an interrupt-driven I/O interface might work for the floppy disk.

Overhead of Interrupt-Driven I/O

Example Suppose we have the same hard disk and processor we used in the example on page 676, but we use interrupt-driven I/O. The overhead for each transfer, including the interrupt, is 500 clock cycles. Find the fraction of the processor consumed if the hard disk is only transferring data 5% of the time.

number of transfers/sec = $\frac{4\text{MB}}{1.6\text{B}}$ = 250K

only 5% of 250K × 500 × 2ns

Answer The interrupt rate when the disk is busy is the same as the polling rate. Hence,

$$\begin{aligned}\text{Cycles per second for disk} &= 250\text{K} \times 500 \\ &= 125 \times 10^6 \text{ cycles per second}\end{aligned}$$

solution 1.25%

3. Once the DMA transfer is complete, the controller interrupts the processor, which can then determine by interrogating the DMA device or examining memory whether the entire operation completed successfully.

There may be multiple DMA devices in a computer system. For example, in a system with a single processor-memory bus and multiple I/O buses, each I/O bus controller will often contain a DMA processor that handles any transfers between a device on the I/O bus and the memory. Let's see how much of the processor is consumed using DMA to handle our hard-disk example.

Overhead of I/O Using DMA

Example Suppose we have the same processor and hard disk as our earlier example on page 676. Assume that the initial setup of a DMA transfer takes 1000 clock cycles for the processor, and assume the handling of the interrupt at DMA completion requires 500 clock cycles for the processor. The hard disk has a transfer rate of 4 MB/sec and uses DMA. If the average transfer from the disk is 8 KB, what fraction of the 500-MHz processor is consumed if the disk is actively transferring 100% of the time? Ignore any impact from bus contention between the processor and DMA controller.

Answer Each DMA transfer takes

$$\frac{8 \text{ KB}}{4 \frac{\text{MB}}{\text{second}}} = 2 \times 10^{-3} \text{ seconds}$$

So if the disk is constantly transferring, it requires

$$\frac{1000 + 500 \frac{\text{cycles}}{\text{transfer}}}{2 \times 10^{-3} \frac{\text{seconds}}{\text{transfer}}} = 750 \times 10^3 \frac{\text{clock cycles}}{\text{second}}$$

Since the processor runs at 500 MHz,

$$\text{Fraction of processor consumed} = \frac{750 \times 10^3}{500 \times 10^6} = 1.5 \times 10^{-3} = 0.2\%$$