EE 4683/5683: COMPUTER ARCHITECTURE

Lecture 6B: Memory Design

Avinash Kodi, kodi@ohio.edu
Acknowledgement: Onur Mutlu

Agenda

- Advanced Cache Optimizations
- Memory – DRAM
- Virtual Memory and I/O

11 Advanced Cache Optimizations

- Reducing hit time
1. Small and simple caches
2. Way prediction
3. Trace caches

- Increasing cache bandwidth
4. Pipelined caches
5. Multibanked caches
6. Nonblocking caches

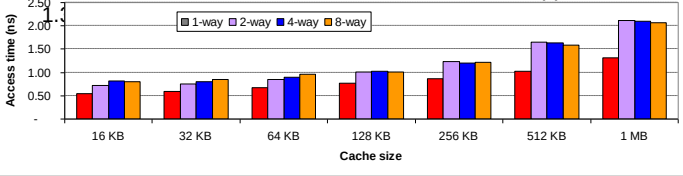
- Reducing Miss Penalty
7. Critical word first
8. Merging write buffers

- Reducing Miss Rate
9. Compiler optimizations

- Reducing miss penalty or miss rate via parallelism
10. Hardware prefetching
11. Compiler prefetching

1. Fast Hit times via Small and Simple Caches

- Index tag memory and then compare takes time
- ⇒ Small cache can help hit time since smaller memory takes less time to index
  - E.g., L1 caches same size for 3 generations of AMD microprocessors: K6, Athlon, and Opteron
  - Also L2 cache small enough to fit on chip with the processor avoids time penalty of going off chip
- Simple ⇒ direct mapping
  - Can overlap tag check with data transmission since no choice
- Access time estimate for 90 nm using CACTI model 4.0
  - Median ratios of access time relative to the direct-mapped caches are 1.32,



2. Fast Hit times via Way Prediction

- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?
- Way prediction: keep extra bits in cache to predict the "way," or block within the set, of next cache access.
  - Multiplexor is set early to select desired block, only 1 tag comparison performed that clock cycle in parallel with reading the cache data
  - Miss ⇒ 1st check other blocks for matches in next clock cycle

- Accuracy ≈ 85%
- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
  - Used for instruction caches vs. data caches

3. Fast Hit times via Trace Cache (Pentium 4 only; and last time?)

- Find more instruction level parallelism?
  How avoid translation from x86 to microops?
- Trace cache in Pentium 4

1. Dynamic traces of the executed instructions vs. static sequences of instructions as determined by layout in memory
   - Built-in branch predictor
2. Cache the micro-ops vs. x86 instructions
   - Decode/translate from x86 to micro-ops on trace cache miss

\+ 1. ⇒ better utilize long blocks (don't exit in middle of block, don't enter at label in middle of block)

\- 1. ⇒ complicated address mapping since addresses no longer aligned to power-of-2 multiples of word size

\- 1. ⇒ instructions may appear multiple times in multiple dynamic traces due to different branch outcomes

4: Increasing Cache Bandwidth by Pipelining

- Pipeline cache access to maintain bandwidth, but higher latency
- Instruction cache access pipeline stages:
  - 1: Pentium
  - 2: Pentium Pro through Pentium III
  - 4: Pentium 4
- ⇒ greater penalty on mispredicted branches
- ⇒ more clock cycles between the issue of the load and the use of the data

5. Increasing Cache Bandwidth: Non-Blocking Caches

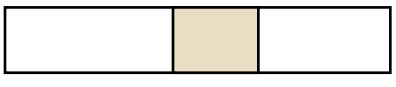
- *Non-blocking cache* or *lockup-free cache* allow data cache to continue to supply cache hits during a miss
  - requires F/E bits on registers or out-of-order execution
  - requires multi-bank memories
- "*hit under miss*" reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- "*hit under multiple miss*" or "*miss under miss*" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
  - Requires muliple memory banks (otherwise cannot support)
  - Penium Pro allows 4 outstanding memory misses

6: Increasing Cache Bandwidth via Multiple Banks

- Rather than treat the cache as a single monolithic block, divide into independent banks that can support simultaneous accesses
  - E.g.,T1 ("Niagara") L2 has 4 banks
- Banking works best when accesses naturally spread themselves across banks ⇒ mapping of addresses to banks affects behavior of memory system
- Simple mapping that works well is "sequential interleaving"
  - Spread block addresses sequentially across banks
  - E,g, if there 4 banks, Bank 0 has all blocks whose address modulo 4 is 0; bank 1 has all blocks whose address modulo 4 is 1; …

7. Reduce Miss Penalty: Early Restart and Critical Word First

- Don't wait for full block before restarting CPU
- *Early restart*—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Spatial locality ⇒ tend to want next sequential word, so not clear size of benefit of just early restart
- *Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block
  - Long blocks more popular today ⇒ Critical Word 1st Widely used



8. Merging Write Buffer to Reduce Miss Penalty

- Write buffer to allow processor to continue while waiting to write to memory
- If buffer contains modified blocks, the addresses can be checked to see if address of new data matches the address of a valid write buffer entry
- If so, new data are combined with that entry
- Increases block size of write for write-through cache of writes to sequential words, bytes since multiword writes more efficient to memory
- The Sun T1 (Niagara) processor, among many others, uses write merging

9. Reducing Misses by Compiler Optimizations

- McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software
- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts(using tools they developed)
- Data
  - *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays
  - *Loop Interchange*: change nesting of loops to access data in order stored in memory
  - *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap
  - *Blocking*: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows

Merging Arrays Example

```
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of stuctures */
struct merge {
   int val;
   int key;
};
struct merge merged_array[SIZE];
```

Reducing conflicts between val & key;
improve spatial locality

Loop Interchange Example

```
/* Before */
for (k = 0; k < 100; k = k+1)
   for (j = 0; j < 100; j = j+1)
        for (i = 0; i < 5000; i = i+1)
              x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
   for (i = 0; i < 5000; i = i+1)
        for (j = 0; j < 100; j = j+1)
              x[i][j] = 2 * x[i][j];
```

Sequential accesses instead of striding through
memory every 100 words; improved spatial locality

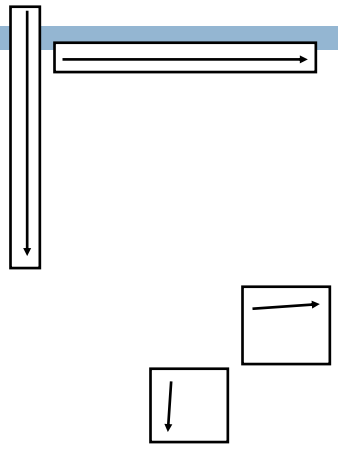
Loop Fusion Example

```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
   {    a[i][j] = 1/b[i][j] * c[i][j];
        d[i][j] = a[i][j] + c[i][j];}
```

2 misses per access to `a` & `c` vs. one miss per access; improve spatial locality

Blocking Example

```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
      {r = 0;
       for (k = 0; k < N; k = k+1){
         r = r + y[i][k]*z[k][j];};
       x[i][j] = r;
      };
```



- Two Inner Loops:
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]
- Capacity Misses a function of N & Cache Size:
  - $2N^3 + N^2$ => (assuming no conflict; otherwise …)
- Idea: compute on BxB submatrix that fits

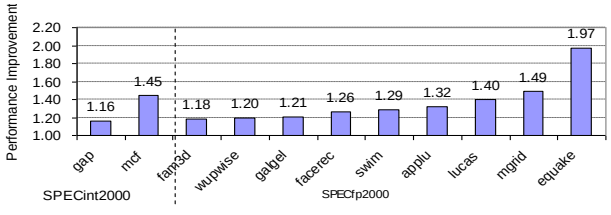
Blocking Example

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
    for (j = jj; j < min(jj+B-1,N); j = j+1)
      {r = 0;
       for (k = kk; k < min(kk+B-1,N); k = k+1) {
          r = r + y[i][k]*z[k][j];};
       x[i][j] = x[i][j] + r;
      };
```

- B called *Blocking Factor*
- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- Conflict Misses Too?

10. Reducing Misses by Hardware Prefetching of Instructions & Data

- Prefetching relies on having extra memory bandwidth that can be used without penalty
- Instruction Prefetching
  - Typically, CPU fetches 2 blocks on a miss: the requested block and the next consecutive block.
  - Requested block is placed in instruction cache when it returns, and prefetched block is placed into instruction stream buffer
- Data Prefetching
  - Pentium 4 can prefetch data into L2 cache from up to 8 streams from 8 different 4 KB pages
  - Prefetching invoked if 2 successive L2 cache misses to a page, if distance between those cache blocks is < 256 bytes



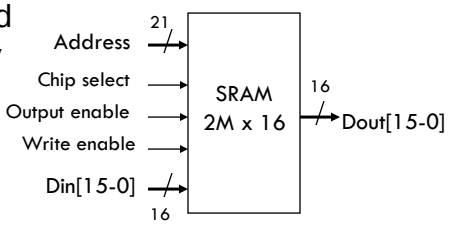
11. Reducing Misses by Software Prefetching Data

- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

Memory Hierarchy Technologies

20

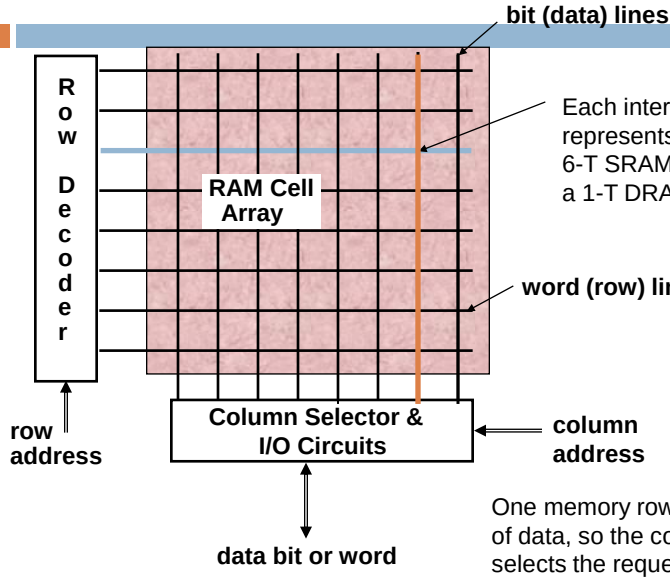
- Caches use *SRAM* for speed and technology compatibility
  - Low density (6 transistor cells), high power, expensive, fast
  - Static: content will last "forever" (until power turned off)



- Main Memory uses *DRAM* for size (density)
  - High density (1 transistor cells), low power, cheap, slow
  - Dynamic: needs to be "refreshed" regularly (~ every 8 ms)

Classical RAM Organization (~Square)

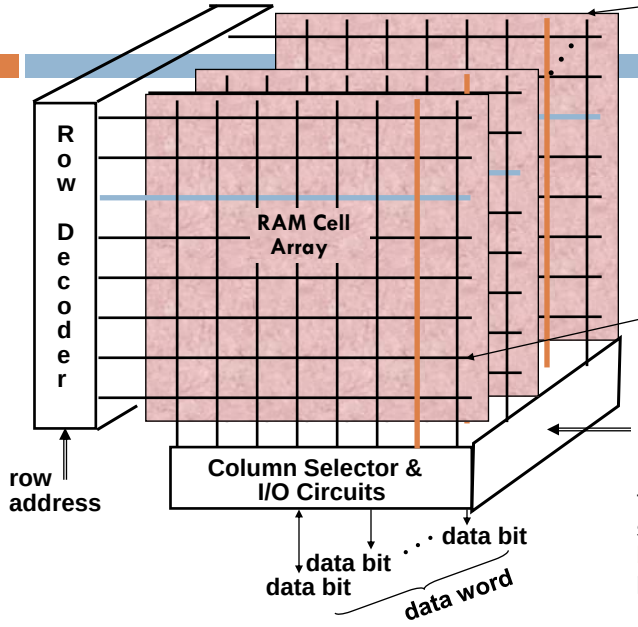
21



One memory row holds a block of data, so the column address selects the requested bit or word from that block

Classical DRAM Organization (~Square Planes

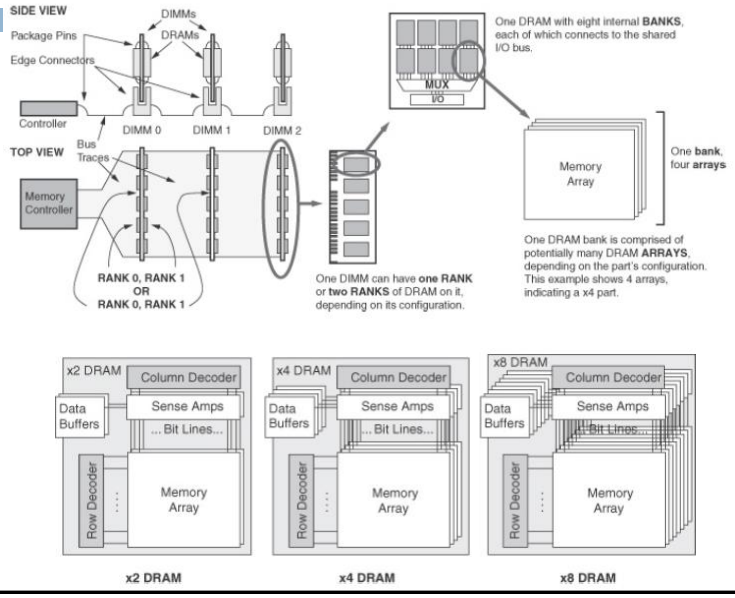
22



The column address selects the requested bit from the row in each plane

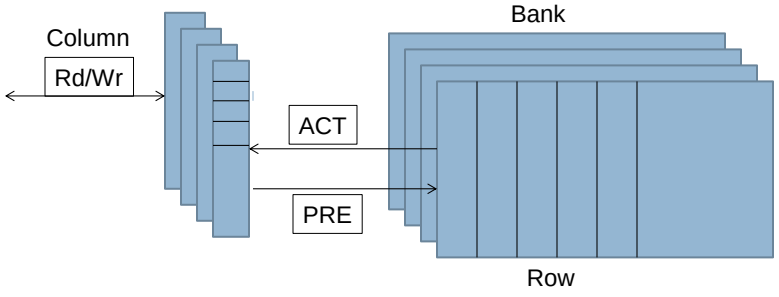
Ranks, Banks, DIMM and DRAM

23



Internal Organization of DRAM

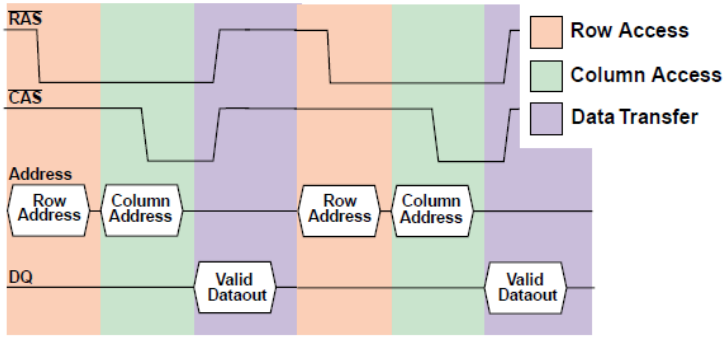
24



- Modern DRAMs are organized as banks, typically four in DDR3
- Sending a PRE (precharge) opens or closes a bank
- A row is sent with the ACT (activate) which causes a buffer to transfer to a buffer
- When row in buffer, successive column addresses at whatever the width of the DRAM (typically 4, 8 or 16 bits)

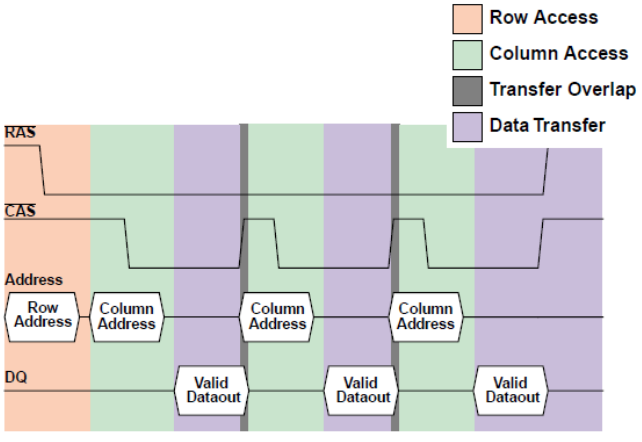
Read Timing for Conventional DRAM

25



Read Timing for Fast Page Mode

26



Quest for DRAM Performance

27

1. Fast Page mode
   - Add timing signals that allow repeated accesses to row buffer without another row access time
   - Such a buffer comes naturally, as each array will buffer 1024 to 2048 bits for each access
2. Synchronous DRAM (SDRAM)
   - Add a clock signal to DRAM interface, so that the repeated transfers would not bear overhead to synchronize with DRAM controller
3. Double Data Rate (DDR SDRAM)
   - Transfer data on both the rising edge and falling edge of the DRAM clock signal ⇒ doubling the peak data rate
   - DDR2 lowers power by dropping the voltage from 2.5 to 1.8 volts + offers higher clock rates: up to 400 MHz
   - DDR3 drops to 1.5 volts + higher clock rates: up to 800 MHz

- Improved Bandwidth, not Latency

DRAM name based on Peak Chip Transfers / Sec
DIMM name based on Peak DIMM MBytes / Sec

Stan-dard	Clock Rate (MHz)	M transfers / second	DRAM Name	Mbytes/s/ DIMM	DIMM Name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10664	PC10700
DDR3	800	1600	DDR3-1600	12800	PC12800

x 2 x 8

Fastest for sale 4/06 ($125/GB)

Need for Error Correction!

29

- Motivation:
  - Failures/time *proportional* to number of bits!
  - As DRAM cells shrink, more vulnerable
- Went through period in which failure rate was low enough without error correction that people didn't do correction
  - DRAM banks too large now
  - Servers always corrected memory systems
- Basic idea: add redundancy through parity bits
  - Common configuration: Random error correction
    - SEC-DED (single error correct, double error detect)
    - One example: 64 data bits + 8 parity bits (11% overhead)
  - Really want to handle failures of physical components as well
    - Organization is multiple DRAMs/DIMM, multiple DIMMs
    - Want to recover from failed DRAM and failed DIMM!
    - "Chip kill" handle failures width of single DRAM chip

Limitations of DRAM

30

- Need for main memory capacity and bandwidth increasing
  - DRAM capacity is hard to scale
- Main memory energy/power is a key design concern
  - DRAM consumes high power due to leakage and refresh
- DRAM technology scaling is ending
  - DRAM capacity, cost, and energy/power is hard to scale

Promise of Emerging Technologies

31

- Idea is to augment rather than replace DRAM, and maybe in future replace
- Some emerging resistive technologies appear promising
  - Phase Change Memory (PCM)
  - Spin Torque Transfer Magnetic Memory (STT-RAM)
  - Memristors
- But can they enable or replacing or even surpass DRAM?

Charge vs Resistive Memories

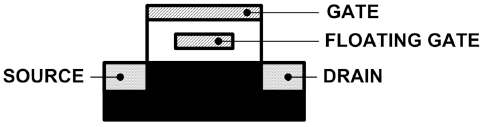
32

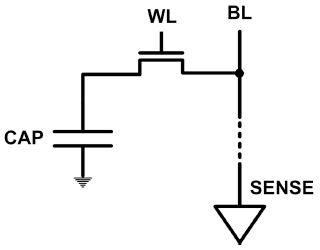
- Charge memory (FLASH, DRAM)
  - Write data by capturing charge, Q
  - Read data by detecting voltage, V
- Resistive memory (PCM, STT-RAM, memristors)
  - Write data by pulsing current dQ/dt
  - Read data by detecting resistance, R

Limits of Charge Memory

33

- Difficult charge placement and control
  - Flash: floating gate charge
  - DRAM: capacitor charge, transistor leakage
- Reliable sensing becomes difficult as charge stored unit size reduces





Emerging Resistive Memory Technologies

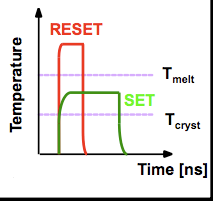
34

- PCM
  - Inject current to change material phase
  - Resistance determined by phase
- STT-MRAM
  - Inject current to change magnetic polarity
  - Resistance determined by ploarity
- Memristors
  - Inject current to change atomic structure
  - Resistance determined by atom distance

Phase Change Memory

35

- Phase change material (chalcogenide glass) exists in two states:
  - Amorphous: low optical reflexivity and high electrical resistance
  - Crystalline: high optical reflexivity and low electrical resistance
- PCM is resistive memory: High resistance (0), Low resistance (1)
- Write: change phase via current injection
  - SET: sustained current to heat cell above Tcrsyt
  - RESET: cell heated above Tmelt and quenched
- Read: detect phase via material resistance



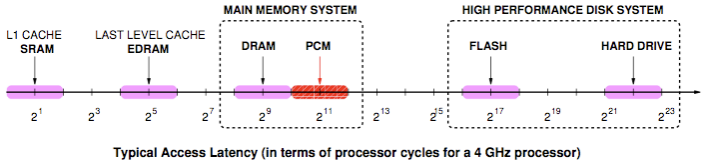
PCM Advantages

36

- Scales better than DRAM, Flash
  - Requires current pulses, which can scale linearly with technology
  - Expected to scale upto 9 nm
  - Prototyped at 20 nm
- Can be denser than DRAM
  - Multiple bits/cell due to large resistance range
- Non-volatile
  - Retain data > 10 years at 85C
- No refresh power

PCM Disadvantages

37



Typical Access Latency (in terms of processor cycles for a 4 GHz processor)

- Latency – comparable but slower than DRAM
  - Read latency: 50 nsec (4x DRAM, 10^{-3}x NAND Flash)
  - Write latency: 150 nsec (12x DRAM)
  - Write bandwidth: 0.1x DRAM, 1x NAND Flash
  - Dynamic Energy: 2-43x DRAM, 1x Flash
  - Endurance: 10^{-8}x DRAM
  - Cell Size: 1.5x DRAM,2-3x Flash

Virtual Memory

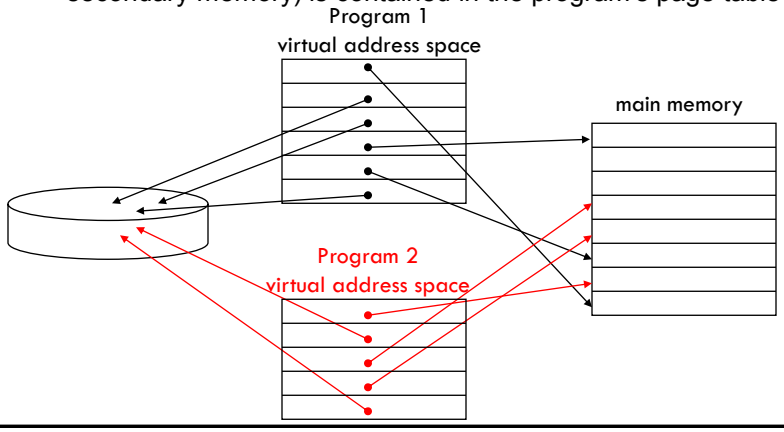
38

- Use main memory as a "cache" for secondary memory
  - Allows efficient and safe sharing of memory among multiple programs
  - Provides the ability to easily run programs larger than the size of physical memory
  - Simplifies loading a program for execution by providing for code relocation (i.e., the code can be loaded anywhere in main memory)
- What makes it work? – again the Principle of Locality
  - A program is likely to access a relatively small portion of its address space during any period of time
- Each program is compiled into its own address space – a "virtual" address space
  - During run-time each virtual address must be translated to a physical address (an address in main memory)

Two Programs Sharing Physical Memory

39

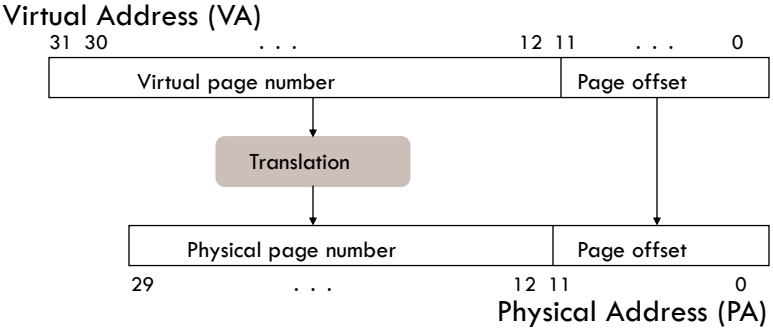
- A program's address space is divided into pages (all one fixed size) or segments (variable sizes)
  - The starting location of each page (either in main memory or in secondary memory) is contained in the program's page table



Address Translation

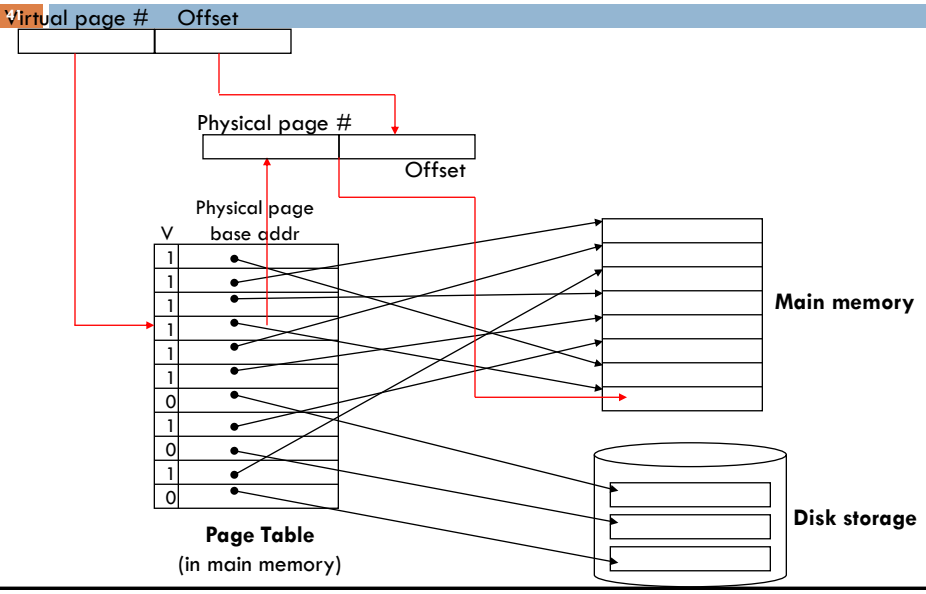
40

- A virtual address is translated to a physical address by a combination of hardware and software



- So each memory request *first* requires an address translation from the virtual space to the physical space
  - A virtual memory miss (i.e., when the page is not in physical memory) is called a page fault

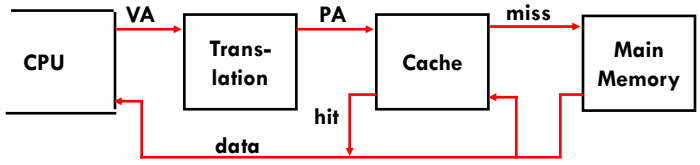
Address Translation Mechanisms



Virtual Addressing with a Cache

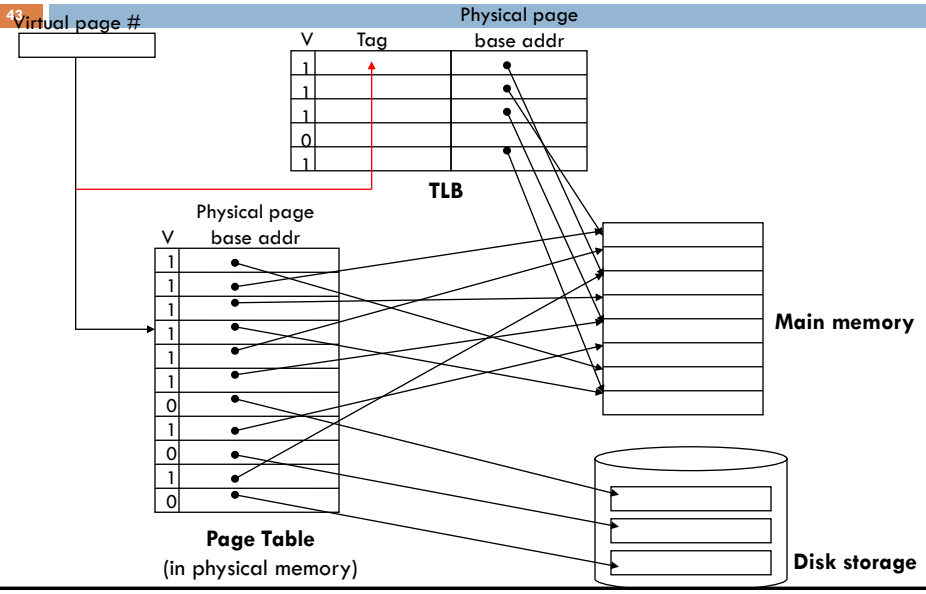
42

- Thus it takes an *extra* memory access to translate a VA to a PA



- This makes memory (cache) accesses very expensive (if every access was really *two* accesses)
- The hardware fix is to use a Translation Lookaside Buffer (TLB) – a small cache that keeps track of recently used address mappings to avoid having to do a page table lookup

Making Address Translation Fast



Translation Lookaside Buffers (TLBs)

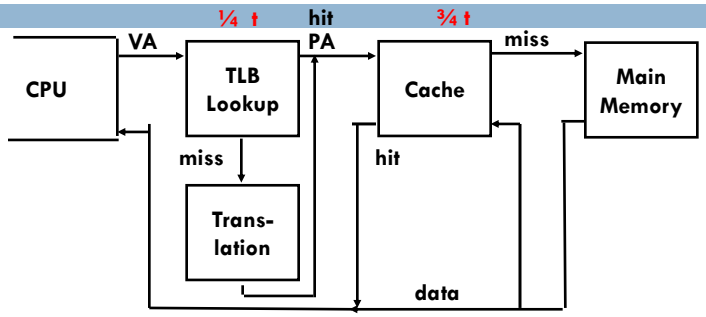
- Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

V	Virtual Page #	Physical Page #	Dirty	Ref	Access

- TLB access time is typically smaller than cache access time (because TLBs are much smaller than caches)
  - TLBs are typically not more than 128 to 256 entries even on high end machines

A TLB in the Memory Hierarchy

45



- A TLB miss – is it a page fault or merely a TLB miss?
  - If the page is loaded into main memory, then the TLB miss can be handled (in hardware or software) by loading the translation information from the page table into the TLB
    - Takes 10's of cycles to find and load the translation info into the TLB
  - If the page is not in main memory, then it's a true page fault
    - Takes 1,000,000's of cycles to service a page fault
- TLB misses are much more frequent than true page faults

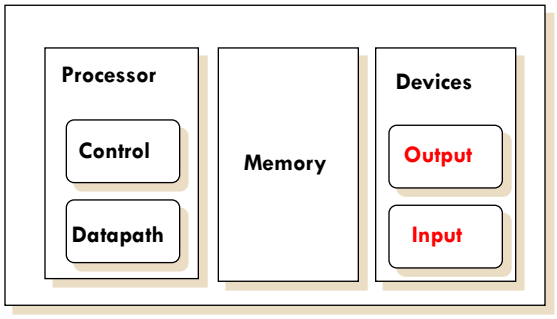
The Hardware/Software Boundary

46

- What parts of the virtual to physical address translation is done by or assisted by the hardware?
  - Translation Lookaside Buffer (TLB) that caches the recent translations
    - TLB access time is part of the cache hit time
    - May allot an extra stage in the pipeline for TLB access
  - Page table storage, fault detection and updating
    - Page faults result in interrupts (precise) that are then handled by the OS
    - Hardware must support (i.e., update appropriately) Dirty and Reference bits (e.g., ~LRU) in the Page Tables
  - Disk placement
    - Bootstrap (e.g., out of disk sector 0) so the system can service a limited number of page faults before the OS is even loaded

Review: Major Components of a Computer

47

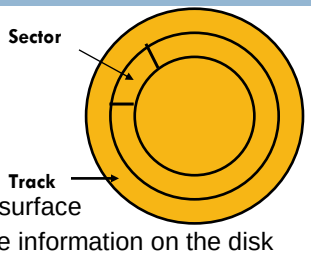


- Important metrics for an I/O system
  - Performance
  - Expandability
  - Dependability
  - Cost, size, weight

Magnetic Disk

48

- Purpose
  - Long term, nonvolatile storage
  - Lowest level in the memory hierarchy
    - slow, large, inexpensive
- General structure
  - A rotating platter coated with a magnetic surface
  - A moveable read/write head to access the information on the disk

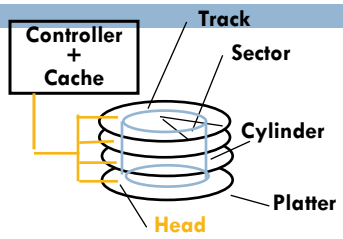


- Typical numbers
  - 1 to 4 (1 or 2 surface) platters per disk of 1" to 5.25" in diameter (3.5" dominate in 2004)
  - Rotational speeds of 5,400 to 15,000 RPM
  - 10,000 to 50,000 tracks per surface
    - cylinder - all the tracks under the head at a given point on all surfaces
  - 100 to 500 sectors per track
    - the smallest unit that can be read/written (typically 512B)

Magnetic Disk Characteristic

49

- Disk read/write components
  1. Seek time: position the head over the proper track (3 to 14 ms avg)
     - due to locality of disk references the actual average seek time may be only 25% to 33% of the advertised number
  2. Rotational latency: wait for the desired sector to rotate under the head (½ of 1/RPM converted to ms)
     - 0.5/5400RPM = 5.6ms to 0.5/15000RPM = 2.0ms
  3. Transfer time: transfer a block of bits (one or more sectors) under the head to the disk controller's cache (30 to 80 MB/s are typical disk transfer rates)
     - the disk controller's "cache" takes advantage of spatial locality in disk accesses
       - cache transfer rates are much faster (e.g., 320 MB/s)
  4. Controller time: the overhead the disk controller imposes in performing a disk I/O access (typically < .2 ms)



Typical Disk Access Time

50

- The average time to read or write a 512B sector for a disk rotating at 10,000RPM with average seek time of 6ms, a 50MB/sec transfer rate, and a 0.2ms controller overhead

If the measured average seek time is 25% of the advertised average seek time, then

- The rotational latency is usually the largest component of the access time

Typical Disk Access Time

51

- The average time to read or write a 512B sector for a disk rotating at 10,000RPM with average seek time of 6ms, a 50MB/sec transfer rate, and a 0.2ms controller overhead

  Avg disk read/write = 6.0ms + 0.5/(10000RPM/(60sec/minute))+ 0.5KB/(50MB/sec) + 0.2ms = 6.0 + 3.0 + 0.01 + 0.2 = 9.21ms

  If the measured average seek time is 25% of the advertised average seek time, then

  Avg disk read/write = 1.5 + 3.0 + 0.01 + 0.2 = 4.71ms

- The rotational latency is usually the largest component of the access time

Input and Output Devices

52

- 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
Graphics display	output	human	800.0000-8000.0000
Network/LAN	input or output	machine	100.0000-1000.0000
Magnetic disk	storage	machine	240.0000-2560.0000

8 orders of magnitude range

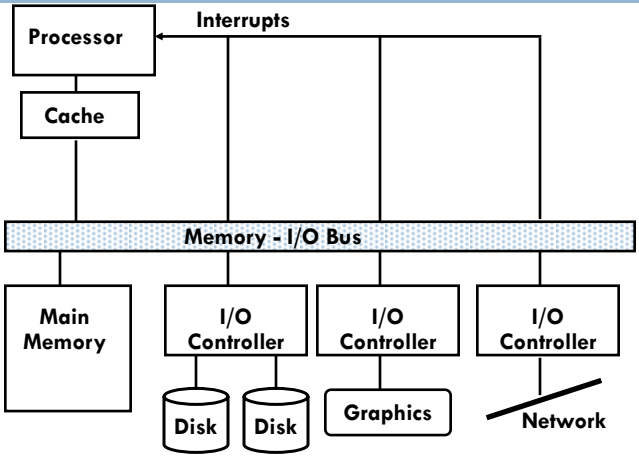
I/O Performance Measures

53

- I/O bandwidth (throughput) – amount of information that can be input (output) and communicated across an interconnect (e.g., a bus) to the processor/memory (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
  - An especially important performance metric in real-time systems
- Many applications require both high throughput and short response times

A Typical I/O System

54



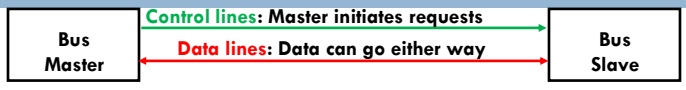
I/O System Interconnect Issues

55

- A bus is a shared communication link (a single set of wires used to connect multiple subsystems) that needs to support a range of devices with widely varying latencies and data transfer rates
  - Advantages
    - Versatile – new devices can be added easily and can be moved between computer systems that use the same bus standard
    - Low cost – a single set of wires is shared in multiple ways
  - Disadvantages
    - Creates a communication bottleneck – bus bandwidth limits the maximum I/O throughput
- The maximum bus speed is largely limited by
  - The length of the bus
  - The number of devices on the bus

Bus Characteristics

56



- Control lines
  - Signal requests and acknowledgments
  - Indicate what type of information is on the data lines
- Data lines
  - Data, addresses, and complex commands
- Bus transaction consists of
  - Master issuing the command (and address) – request
  - Slave receiving (or sending) the data – action
  - Defined by what the transaction does to memory
    - Input – inputs data from the I/O device to the memory
    - Output – outputs data from the memory to the I/O device

Types of Buses

57

- Processor-memory bus (proprietary)
  - Short and high speed
  - Matched to the memory system to maximize the memory-processor bandwidth
  - Optimized for cache block transfers
- I/O bus (industry standard, e.g., SCSI, USB, Firewire)
  - Usually is lengthy and slower
  - Needs to accommodate a wide range of I/O devices
  - Connects to the processor-memory bus or backplane bus
- Backplane bus (industry standard, e.g., ATA, PCIexpress)
  - The backplane is an interconnection structure within the chassis
  - Used as an intermediary bus connecting I/O busses to the processor-memory bus

Synchronous and Asynchronous Buses

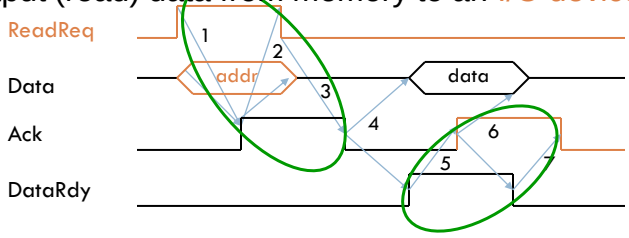
58

- Synchronous bus (e.g., processor-memory buses)
  - Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
  - Advantage: involves very little logic and can run very fast
  - Disadvantages:
    - Every device communicating on the bus must use same clock rate
    - To avoid clock skew, they cannot be long if they are fast
- Asynchronous bus (e.g., I/O buses)
  - It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq, Ack, DataRdy)
  - Advantages:
    - Can accommodate a wide range of devices and device speeds
    - Can be lengthened without worrying about clock skew or synchronization problems
  - Disadvantage: slow(er)

Asynchronous Bus Handshaking Protocol

59

- Output (read) data from memory to an I/O device



I/O device signals a request by raising ReadReq and putting the addr on the data lines

1. Memory sees ReadReq, reads addr from data lines, and raises Ack
2. I/O device sees Ack and releases the ReadReq and data lines
3. Memory sees ReadReq go low and drops Ack
4. When memory has data ready, it places it on data lines and raises DataRdy
5. I/O device sees DataRdy, reads the data from data lines, and raises Ack
6. Memory sees Ack, releases the data lines, and drops DataRdy
7. I/O device sees DataRdy go low and drops Ack

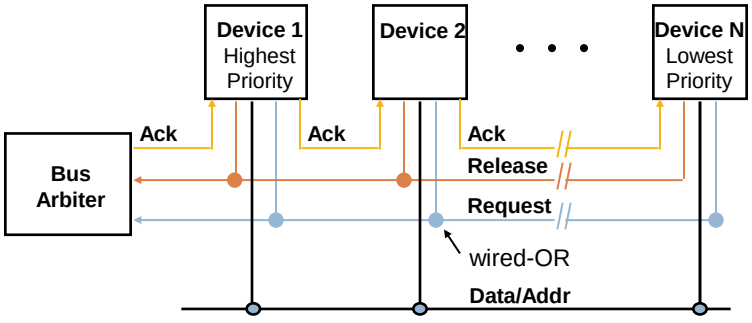
The Need for Bus Arbitration

60

- Multiple devices may need to use the bus at the same time so must have a way to arbitrate multiple requests
- Bus arbitration schemes usually try to balance:
  - Bus priority – the highest priority device should be serviced first
  - Fairness – even the lowest priority device should never be completely locked out from the bus
- Bus arbitration schemes can be divided into four classes
  - Daisy chain arbitration – see next slide
  - Centralized, parallel arbitration – see next-next slide
  - Distributed arbitration by self-selection – each device wanting the bus places a code indicating its identity on the bus
  - Distributed arbitration by collision detection – device uses the bus when its not busy and if a collision happens (because some other device also decides to use the bus) then the device tries again later (Ethernet)

Daisy Chain Bus Arbitration

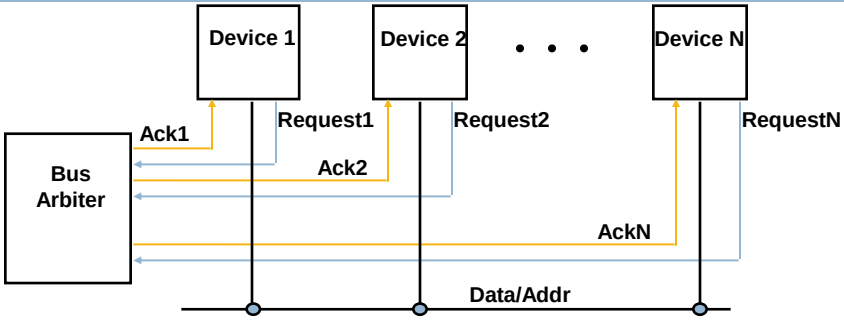
61



- Advantage: simple
- Disadvantages:
  - Cannot assure fairness – a low-priority device may be locked out indefinitely
  - Slower – the daisy chain grant signal limits the bus speed

Centralized Parallel Arbitration

62



- Advantages: flexible, can assure fairness
- Disadvantages: more complicated arbiter hardware
- Used in essentially all processor-memory buses and in high-speed I/O buses

Buses in Transition

- Companies are transitioning from synchronous, parallel, *wide* buses to asynchronous *narrow* buses
  - Reflection on wires and clock skew makes it difficult to use 16 to 64 parallel wires running at a high clock rate (e.g., ~400 MHz) so companies are transitioning to buses with a few one-way wires running at a very high "clock" rate (~2 GHz)

	PCI	PCIexpress	ATA	Serial ATA
Total # wires	120	36	80	7
# data wires	32 – 64 (2-way)	2 x 4 (1-way)	16 (2-way)	2 x 2 (1-way)
Clock (MHz)	33 – 133	635	50	150
Peak BW (MB/s)	128 – 1064	300	100	375 (3 Gbps)

Communication of I/O Devices and Processor

64

- How the processor directs the I/O devices
  - Special I/O instructions
    - Must specify both the device and the command
  - Memory-mapped I/O
    - Portions of the high-order memory address space are assigned to each I/O device
    - 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
- How the I/O device communicates with the processor
  - 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
  - Interrupt-driven I/O – the I/O device issues an interrupts to the processor to indicate that it needs attention

Interrupt-Driven I/O

65

- An I/O interrupt is asynchronous wrt instruction execution
  - Is not associated with any instruction so doesn't prevent any instruction from completing
    - You can pick your own convenient point to handle the interrupt
- With I/O interrupts
  - Need a way to identify the device generating the interrupt
  - Can have different urgencies (so may need to be prioritized)
- 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
  - Cause an interrupt (I/O device) and detect an interrupt and save the necessary information to resume normal processing after servicing the interrupt (processor)

Direct Memory Access (DMA)

66

- For high-bandwidth devices (like disks) interrupt-driven I/O would consume a *lot* of processor cycles
- DMA – the I/O controller has the ability to transfer 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 I/O DMA controller manages the entire transfer (possibly thousand of bytes in length), arbitrating for the bus
  3. When the DMA transfer is complete, the I/O controller interrupts the processor to let it know that the transfer is complete
- There may be multiple DMA devices in one system
  - Processor and I/O controllers contend for bus cycles and for memory

The DMA Stale Data Problem

67

- In systems with caches, there can be two copies of a data item, one in the cache and one in the main memory
  - For a DMA read (from disk to memory) – the processor will be using stale data if that location is also in the cache
  - For a DMA write (from memory to disk) and a write-back cache – the I/O device will receive stale data if the data is in the cache and has not yet been written back to the memory
- The coherency problem is solved by
  1. Routing all I/O activity through the cache – expensive and a large negative performance impact
  2. Having the OS selectively invalidate the cache for an I/O read or force write-backs for an I/O write (flushing)
  3. Providing hardware to selectively invalidate or flush the cache – need a hardware snooper

I/O and the Operating System

68

- The operating system acts as the interface between the I/O hardware and the program requesting I/O
  - To protect the shared I/O resources, the user program is not allowed to communicate directly with the I/O device
- Thus OS must be able to give commands to I/O devices, handle interrupts generated by I/O devices, provide equitable access to the shared I/O resources, and schedule I/O requests to enhance system throughput
  - I/O interrupts result in a transfer of processor control to the supervisor (OS) process