

CS 135: Computer Architecture 1

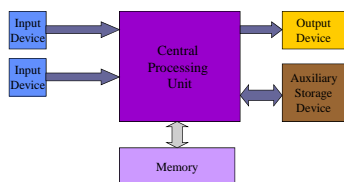
Instructor: Prof. Bhagi Narahari
Dept. of Computer Science
Course URL: www.seas.gwu.edu/~narahari/cs135/

The Memory Hierarchy...

- Brief Overview of Memory Design
 - What is the memory hierarchy ?
 - components
 - Focus on Cache design
 - How does Cache memory work ?
 - How are addresses mapped to Cache
 - How to rewrite code to get better cache performance ? –code optimization
 - How do disks work ?
 - Virtual memory – what is it ?

CS 135: Computer Architecture, Bhagi Narahari

A Computer



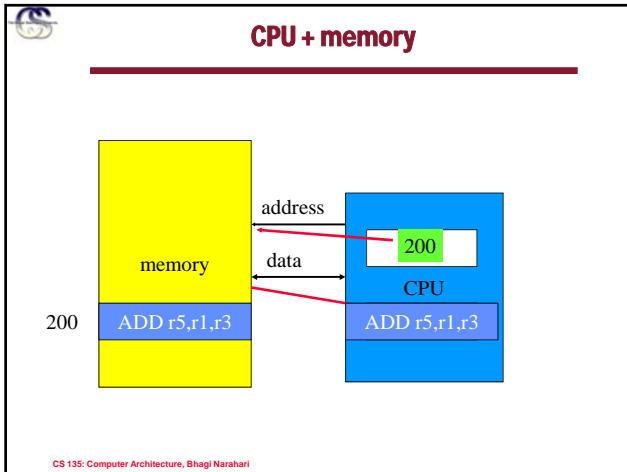
The computer is composed of input devices, a central processing unit, a memory unit and output devices.

CS 135: Computer Architecture, Bhagi Narahari

Computer organization - Recap

- CPU has two components:
 - Arithmetic and Logic Unit (ALU)
 - Performs arithmetic operations
 - Performs logical operations
 - Control Unit
 - Controls the action of the other computer components so that instructions are executed in the correct sequence
- Memory
 - Contains instructions and data
 - CPU requests data, data fetched from memory
 - How long does it take to fetch from memory ?

CS 135: Computer Architecture, Bhagi Narahari



Memory Unit

- An ordered sequence of storage cells, each capable of holding a piece of data.
- Address space
 - Size of memory: N bit address space = 2^N memory locations
- Addressability
 - Size of each memory location – k bits
- Total memory size = $k \cdot 2^N$ bits
- Assumption thus far: Processor/CPU gets data or instruction from some memory address (Inst fetch or Load/Store instruction)
 - But how is memory actually organized ?
 - Can everything we need fit into a memory that is close to the CPU ?

CS 135: Computer Architecture, Bhagi Narahari

Where does run-time stack fit into this. .

- We looked at how variables in C are allocated memory addresses
 - Each function has activation record
 - Compiler takes care of allocation of memory addresses to variables
- Question now is: where are these memory addresses and how long does it take to fetch the contents into the processor register
 - LD R0, A
 - We know the address of A, but how long will it take to go into memory and fetch into register R0 ?

CS 135: Computer Architecture, Bhagi Narahari

Memory Technology

- Random access memory
 - Can read from any location by supplying address of data
- Memory Comes in Many Flavors
 - Main RAM memory Key features
 - RAM is packaged as a chip.
 - Basic storage unit is a cell (one bit per cell).
 - Multiple RAM chips form a memory.
 - SRAM (Static Random Access Memory) or DRAM (Dynamic Random Access Memory)
 - ROM, EPROM, EEPROM, Flash, etc.
 - Read only memories – store OS
 - “Secondary memory” Disks, Tapes, etc.
- Difference in speed, price and “size”
 - Fast is small and/or expensive
 - Large is slow and/or cheap

CS 135: Computer Architecture, Bhagi Narahari

How is memory really organized ?

- Many types of memory with different speeds
- Processor speed and memory speed mismatched
 - Data transferred between memory and processor
 - Instructions or data
- What does processor do while waiting for data to be transferred ?
 - Idle – processor is stalled leading to slowdown in speed and lower performance
- Why can't we have memory as fast as processor
 - Technology, cost, size
- What is the solution then ?

CS 135: Computer Architecture, Bhagi Narahari

Memory Hierarchies

- Some fundamental and enduring properties of hardware and software:
 - Fast storage technologies cost more per byte and have less capacity.
 - The gap between CPU and main memory speed is widening.
 - Well-written programs tend to exhibit good locality.
- These fundamental properties complement each other beautifully.
- They suggest an approach for organizing memory and storage systems known as a **memory hierarchy**.

CS 135: Computer Architecture, Bhagi Narahari

An Example Memory Hierarchy

Items on a desktop (register) or in a drawer (cache) are more readily accessible than those in a file cabinet (main memory) or in a closet in another room

keep more frequently accessed items on desktop, next frequent in drawer, etc. and things you need a lot less often in the closet in another room!

CS 135: Computer Architecture, Bhagi Narahari

An Example Memory Hierarchy

Smaller, faster, and costlier (per byte) storage devices

Larger, slower, and cheaper (per byte) storage devices

CS 135: Computer Architecture, Bhagi Narahari

Memory Hierarchies

- **Key Principles**
 - Locality – most programs do not access code or data uniformly
 - Smaller hardware is faster
- **Goal**
 - Design a memory hierarchy “with cost almost as low as the cheapest level of the hierarchy and speed almost as fast as the fastest level”
 - This implies that we be clever about keeping more likely used data as “close” to the CPU as possible
- **Levels provide subsets**
 - Anything (data) found in a particular level is also found in the next level below.
 - Each level maps from a slower, larger memory to a smaller but faster memory

CS 135: Computer Architecture, Bhagi Narahari

Locality

- **Principle of Locality:**
 - Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.
 - **Temporal locality:** Recently referenced items are likely to be referenced in the near future.
 - **Spatial locality:** Items with nearby addresses tend to be referenced close together in time.

CS 135: Computer Architecture, Bhagi Narahari

Locality

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

Locality Example:

- **Data**
 - Reference array elements in succession (stride-1 reference pattern): **Spatial locality**
 - Reference `sum` each iteration: **Temporal locality**
- **Instructions**
 - Reference instructions in sequence: **Spatial locality**
 - Cycle through loop repeatedly: **Temporal locality**

CS 135: Computer Architecture, Bhagi Narahari

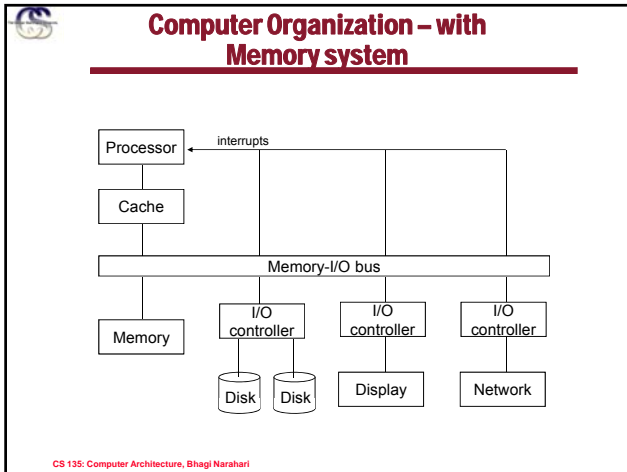
Memory Hierarchy: The Tradeoff

	register reference	L1-cache reference	L2-cache reference	memory reference	disk memory reference
size:	608 B	128k B	512kB – 4MB	128 MB	27GB
speed:	1.4 ns	4.2 ns	16.8 ns	112 ns	9 ms
\$/Mbyte:			\$90/MB	\$2.6/MB	\$0.01/MB
block size:	4 B	4 B	16 B	4-8 KB	

larger, slower, cheaper →

(Numbers are for a Alpha 21264 at 700MHz)

CS 135: Computer Architecture, Bhagi Narahari



Memory Hierarchy: Key concepts

- We shall return to a detailed discussion of the various components of the memory hierarchy
 - What is a Cache memory
 - How does Memory access and Disk access work ?
 - How should we organize the memory hierarchy
- How can we rewrite our code to improve performance
 - Based on memory access, type of instructions, etc.

CS 135: Computer Architecture, Bhagi Narahari

A Simple Model of Memory ...


- Sequence of addresses
 - How many ?
- CPU generates request for memory location – i.e., an address
 - How long does it take to get this data ?
 - Depends where it is in the Memory hierarchy
- Simplified Model for memory hierarchy:
 - small amount of On-chip Cache memory
 - Larger amount of off-chip Main memory
 - Huge Disk

CS 135: Computer Architecture, Bhagi Narahari

Memory Access times

- memory access time
 - On-chip Cache takes 1 processor cycle
 - Main memory takes a number (10-50) processor cycles
 - Disk takes a huge amount
- Simple model we will use for now:
 - Memory = Cache + Main memory
 - Small size Cache = not everything fits in it
- Simplified Cache organization:
 - Cache consists of a set of blocks each of some number of bytes
 - Only a block can be fetched into and out of cache
 - Eg; if block is 16 bytes, then load 16 bytes into cache
 - Cannot load a single byte


CS 135: Computer Architecture, Bhagi Narahari



Memory Access times using Simplified Model

- If data is found in Cache then time =1
 - Called a **cache hit**
- Else time is Main memory access time
 - **Cache miss**, means read from next level
- Note: need a 'control unit' to determine if location is in cache or not
 - **Cache controller**
- Why does concept of caching work ?
 - Principle of Locality
 - Programs access data nearby, or data/instructions that were used recently

CS 135: Computer Architecture, Bhagi Narahari




Summary: Memory Access time optimization

- If each access to memory leads to a **cache hit** then time to fetch from memory is one cycle
 - **Program performance is good!**
- If each access to memory leads to a **cache miss** then time to fetch from memory is much larger than 1 cycle
 - **Program performance is bad!**
- Design Goal:


How to arrange data/instructions so that we have as few cache misses as possible.

CS 135: Computer Architecture, Bhagi Narahari



Some details of Memory Organization


CS 135: Computer Architecture, Bhagi Narahari



Random-Access Memory (RAM)

- Static RAM (**SRAM**)
 - This is what we saw before in the example of a memory built from latches (transistors).
 - Like a register – once written it keeps its value till next write
 - Retains value indefinitely, as long as it is kept powered.
 - Relatively insensitive to disturbances such as electrical noise.
 - Faster and more expensive than DRAM.
- Dynamic RAM (**DRAM**)
 - Each cell stores a bit with a capacitor and transistor.
 - Value must be refreshed every 10-100 ms else we lose data
 - Sensitive to disturbances.
 - Slower and cheaper than SRAM.
 - Many types of DRAMS
 - SDRAM, DDR-DRAM, Video DRAM, etc.


CS 135: Computer Architecture, Bhagi Narahari



Nonvolatile Memories

- DRAM and SRAM are volatile memories
 - Lose information if powered off.
- Nonvolatile memories retain value even if powered off.
 - Generic name is read-only memory (**ROM**).
 - Misleading because some ROMs can be read and modified.
- Types of ROMs
 - Programmable ROM (**PROM**)
 - Eraseable programmable ROM (**EPROM**)
 - Electrically eraseable PROM (**EEPROM**)
 - Flash memory
- Firmware
 - Program stored in a ROM
 - Boot time code, BIOS (basic input/output system)
 - graphics cards, disk controllers.


CS 135: Computer Architecture, Bhagi Narahari



SRAM vs DRAM Summary

	Tran. per bit	Access time	Persist?	Sensitive?	Cost	Applications
SRAM	6	1X	Yes	No	100x	cache memories
DRAM	1	10X	No	Yes	1X	Main memories, frame buffers


CS 135: Computer Architecture, Bhagi Narahari



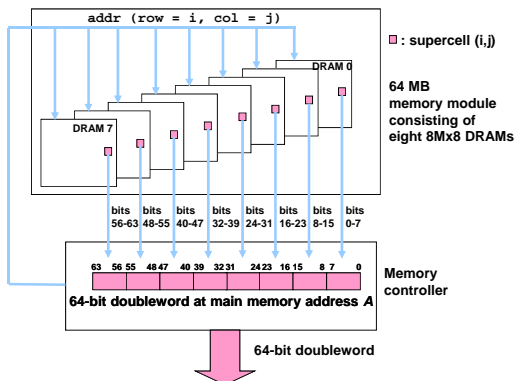
Building the Memory

- Build large memory using several smaller memory chips
 - How to organize a large memory using several “modules”
- CPU generates an address request
 - The address can be in any module
 - Need to figure out which one

CS 135: Computer Architecture, Bhagi Narahari



Memory Modules for 64 bit Processor



addr (row = i, col = j)

□ : supercell (i,j)

64 MB memory module consisting of eight 8Mx8 DRAMs

bits 56-63, 48-55, 40-47, 32-39, 24-31, 16-23, 8-15, 0-7

Memory controller

64-bit doubleword at main memory address A

64-bit doubleword

CS 135: Computer Architecture, Bhagi Narahari

Typical Bus Structure Connecting CPU and Memory

- A **bus** is a collection of parallel wires that carry address, data, and control signals.
- Buses are typically shared by multiple devices.

CS 135: Computer Architecture, Bhagi Narahari

Memory Read Transaction (1)

- CPU places address **A** on the memory bus.

Load operation: Load R0, A
or in Intel: `movl A, R0`

CS 135: Computer Architecture, Bhagi Narahari

Memory Read Transaction (2)

- Main memory reads **A** from the memory bus, retrieves word **x**, and places it on the bus.

Load operation: Load R0, A
Or in Intel: `movl A, R0`

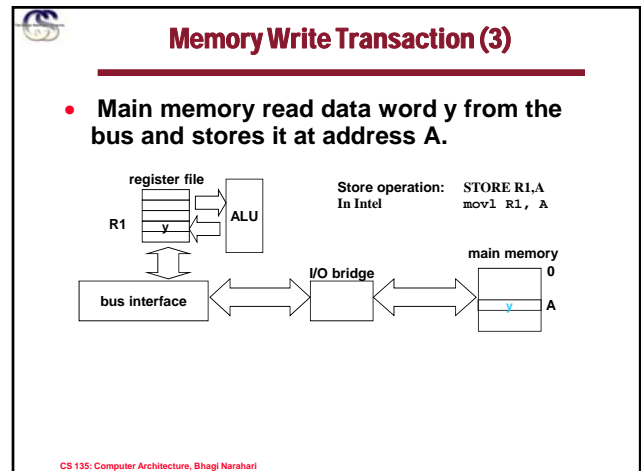
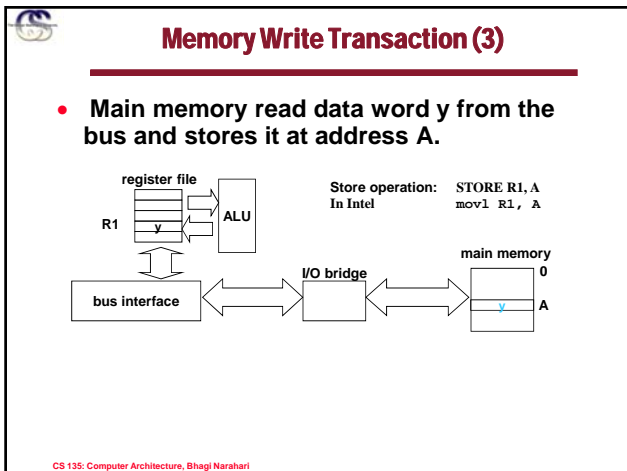
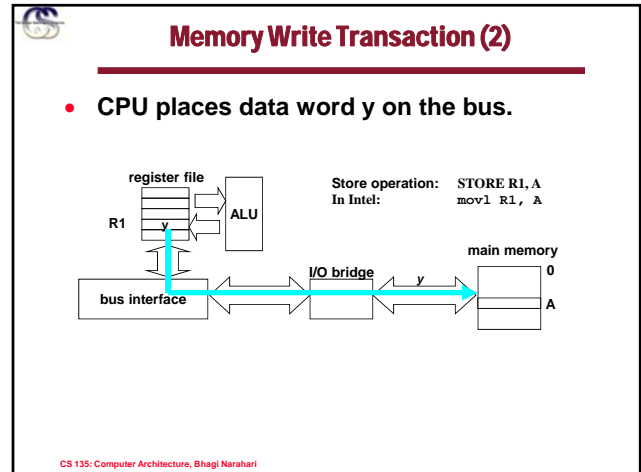
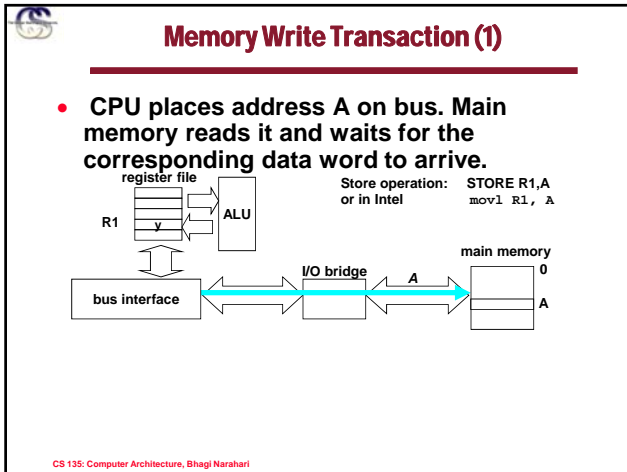
CS 135: Computer Architecture, Bhagi Narahari

Memory Read Transaction (3)

- CPU reads word **x** from the bus and copies it into register **%eax**.

Load operation: Load R0, A
or in Intel: `movl A, R0`

CS 135: Computer Architecture, Bhagi Narahari



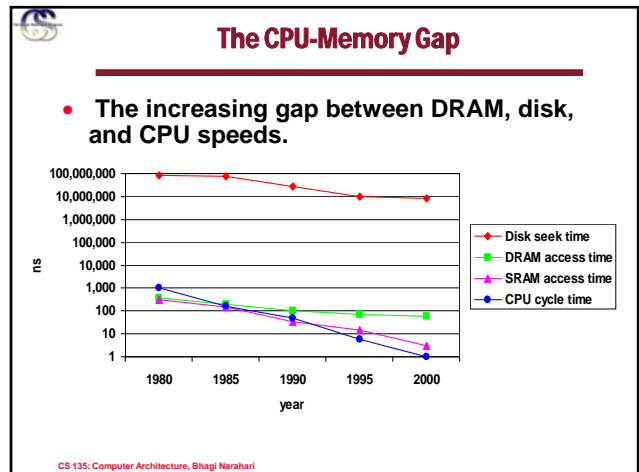
Memory Access time and Performance ?

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$$CPU = IC * CPI * Clk$$

• How does memory access time fit into CPU time equation ?

CS 135: Computer Architecture, Bhagi Narahari



Performance

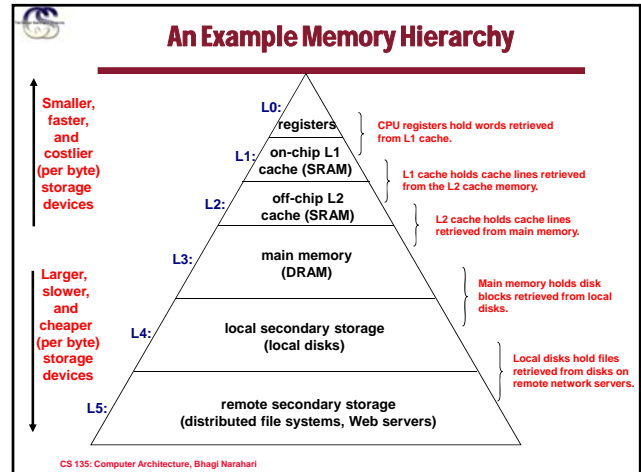
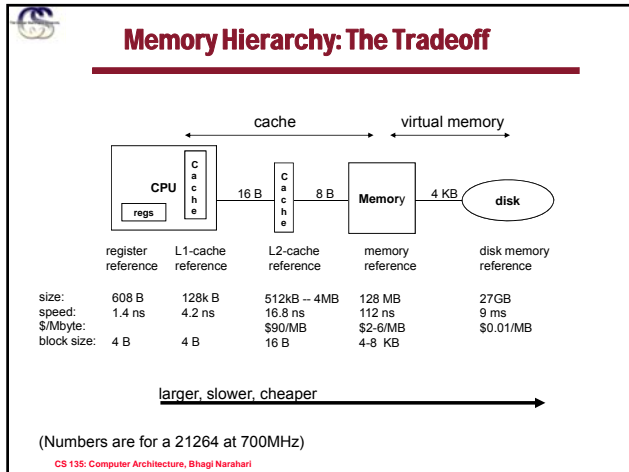
- Simplified model:**
 - Processor is (1) in execution or (2) waits for memory
 - “effective”(Real) CPI increases
 - execution time = (execution cycles + memory stall cycles) × cycle time
- Improve performance = decrease stall cycles**
 - Decrease time to access memory
 - How ?

CS 135: Computer Architecture, Bhagi Narahari

Memory Hierarchies

- Organize memory as a **memory hierarchy**.
- Key Principles**
 - Locality – most programs do not access code or data uniformly
 - Smaller hardware is faster
- Goal**
 - Design a memory hierarchy “with cost almost as low as the cheapest level of the hierarchy and speed almost as fast as the fastest level”
 - This implies that we be clever about keeping more likely used data as “close” to the CPU as possible
- Levels provide subsets**
 - Anything (data) found in a particular level is also found in the next level below.
 - Each level maps from a slower, larger memory to a smaller but faster memory

CS 135: Computer Architecture, Bhagi Narahari



Recap -- Locality

- Principle of Locality:
 - Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.
 - Temporal locality:** Recently referenced items are likely to be referenced in the near future.
 - Spatial locality:** Items with nearby addresses tend to be referenced close together in time.

Locality Example:

```

sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;

```


- Data
 - Reference array elements in succession: **spatial locality**
 - Reference `sum` each iteration: **temporal locality**
- Instructions
 - Reference instructions in sequence: **spatial locality**
 - Cycle through loop repeatedly: **temporal locality**

CS 135: Computer Architecture, Bhagi Narahari

Cache memory

- Why ?
- How does it work ?
- How is the memory organized ?


CS 135: Computer Architecture, Bhagi Narahari



Why Cache ?

- **Gap between main memory speed and processor speed**
 - Reading data/inst from memory will take more than 1 processor cycle
 - Increases time to execute program
- **Place a small but fast memory close to the processor**
- **Why does this work**
 - Principle of Locality


CS 135: Computer Architecture, Bhagi Narahari



Simple Model of Memory Hierarchy...

- **Sequence of addresses**
 - How many ?
- **CPU generates request for memory location – i.e., an address**
 - How long does it take to get this data ?
 - Depends where it is in the Memory hierarchy
- **Simplified Model for memory hierarchy:**
 - small amount of Fast On-chip Cache memory
 - Larger amount of off-chip Main memory
 - Huge Disk


CS 135: Computer Architecture, Bhagi Narahari



How does Cache memory work ?

- **Address space = 2^N words each of some size K bits**
 - N bit address
- **Memory addresses go from 0 to 2^N-1**
 - These are the addresses that the processor requests in the Load or Store instructions, or when fetching a new instruction (value in PC)
- **Some of these memory locations are placed in the cache**
 - If you see it in the cache then don't need to go all the way to memory to read them
 - Faster time to read/write inst/data!

CS 135: Computer Architecture, Bhagi Narahari



Memory Access times

- **memory access time**
 - On-chip Cache takes 1 processor cycle
 - Main memory takes a number (10-50) processor cycles
 - Disk takes a huge amount
- **Simple model we will use:**
 - Memory = Cache + Main memory
 - Small size Cache = not everything fits in it
- **Cache organization:**
 - Cache consists of a set of blocks each of some number of bytes
 - Only a block can be fetched into and out of cache
 - Eg; if block is 16 bytes, then load 16 bytes into cache
 - Cannot load a single byte

CS 135: Computer Architecture, Bhagi Narahari

Memory Access times using Simplified Model

- If data is found in Cache then time = 1
 - Called a **cache hit**
- Else time is Main memory access time
 - **Cache miss**, means read from next level
- Note: need a 'control unit' to determine if location is in cache or not
 - **Cache controller**
- Why does concept of caching work ?
 - Principle of Locality
 - Programs access data nearby, or data/instructions that were used recently

CS 135: Computer Architecture, Bhagi Narahari

Terminology Summary

- **Hit**: data appears in block in upper level (i.e. block X in cache)
 - **Hit Rate**: fraction of memory access found in upper level
 - **Hit Time**: time to access upper level which consists of
 - RAM access time + Time to determine hit/miss
- **Miss**: data needs to be retrieved from a block in the lower level (i.e. block Y in memory)
 - **Miss Rate** = 1 - (Hit Rate)
 - **Miss Penalty**: Extra time to replace a block in the upper level +
 - Time to deliver the block the processor
- Hit Time << Miss Penalty (500 instructions on Alpha 21264)

CS 135: Computer Architecture, Bhagi Narahari

Memory Hierarchy--Performance

- Placing the fastest memory near the CPU can result in increases in performance
- Consider the number of cycles the CPU is stalled waiting for a memory access:
 - memory stall cycles**
 - CPU execution time = (CPU clk cycles + Memory stall cycles) * clk cycle time.
 - Memory stall cycles = number of misses * miss penalty
 - **Fewer misses in cache = better performance!**

CS 135: Computer Architecture, Bhagi Narahari

Cache Performance – Simplified Models

- Memory stall cycles =
 - number of misses * miss penalty =
- = $IC * (\text{memory accesses/instruction}) * \text{miss rate} * \text{miss penalty}$
- Hit rate/ratio r = number of requests that are hits/total number requests
 - Miss rate = 1 - (hit rate)
- Cost of memory access = $rC_h + (1-r)C_m$
 - C_h is cost/time from cache, C_m is cost/time when miss – fetch from memory
 - Extend to multiple levels of cache
 - Hit ratios for level 1, level 2, etc.
 - Access times for level 1, level 2, etc.
 - $r_1 C_{h1} + r_2 C_{h2} + (1 - r_1 - r_2) C_m$

CS 135: Computer Architecture, Bhagi Narahari



So how well does your memory perform ?

- What is the average time to access memory ?
- An application program has a miss rate
- The memory design gives us hit time and miss penalty
- So what is the average memory access time (AMAT) ?
 - Hit time : if found in cache
 - Hit time + miss penalty : if not in cache

CS 135: Computer Architecture, Bhagi Narahari



Average Memory Access Time

$$AMAT = HitTime + (1 - h) \times MissPenalty$$

- **Hit time**: basic time of every access.
- **Hit rate (h)**: fraction of access that hit
 - usually substituted by **miss rate m = (1-h)**
- **Miss penalty**: extra time to fetch a block from lower level, including time to replace in CPU

CS 135: Computer Architecture, Bhagi Narahari



Example 1..

- System = Processor, Cache, Main Memory
 - Cache time = 1 processor cycle
 - Memory access time = 50 processor cycles
- Suppose out of 1000 memory accesses (due to Load/Store and Inst fetch)
 - 40 misses in the cache
 - 960 hit in the Cache
 - Miss ratio = $40/1000 = 4\%$
- Average memory access time with and without cache ?


CS 135: Computer Architecture, Bhagi Narahari



Example..

- Average memory access time with and without cache ?
- $AMAT\text{-}cache = 1 + \text{miss ratio} \times \text{miss penalty}$
 - $1 + (0.04) \times 50 = 3$
- $AMAT\text{-}without\text{-}cache = 50$
- What happens if miss ratio increases ?


CS 135: Computer Architecture, Bhagi Narahari



Example 2

- In reality we have an application with a instruction mix
 - CPU time = $IC * CPI * Clock$
 - Effective CPI = CPI + Average Stalls per instruction
- How many memory access per instruction ?
 - When is memory accessed ?
- Example: suppose your program has 20% Load/Store operations


CS 135: Computer Architecture, Bhagi Narahari



Example 2

- Suppose your program has 20% Load/Store operations
 - Memory is accessed once for each instruction PLUS again for each Load/Store
 - Out of 100 instructions, we will have 120 accesses to memory
 - Average memory accesses per instruction = $120/100 = 1.2$
- How many stall cycles (waiting for memory) ?
 - $1.2 * 0.04 * 50 = 1.2 * 2 = 2.4$
 - Processor is stalled 2.4 cycles each inst


CS 135: Computer Architecture, Bhagi Narahari



Example 2 – complete example

- Assume “ideal” CPI (without memory stalls) is 1.5
- What is the CPU time now ?
- CPU = IC * CPI * Clock
 - IC * CPI is the number of CPU cycles
- CPI is number of cycles for CPU execution per instruction
- Each instruction stalls for some number of cycles
- IC * (CPI + Avg Stall cycles) * Clock
 - $IC * (1.5 + 2.4) * clock = IC * 3.9$ clock cycles


CS 135: Computer Architecture, Bhagi Narahari



Cache Memory Hardware Design

- Main memory has 2^N locations
- Cache has 2^k locations
 - Smaller than main memory
 - How to “organize” these cache locations ?
- Processor generates N bit address
- How do we look at this N bit address and decide (a) if it is in cache and (b) where to place it in cache ?


CS 135: Computer Architecture, Bhagi Narahari



Cache Design--Questions

- **Q1: Where can a block be placed in the upper level?**
 - block placement
- **Q2: How is a block found if it is in the upper level?**
 - block identification
 - Use the address bits to identify the block
- **Q3: Which block should be replaced on a miss?**
 - block replacement
- **Q4: What happens on a write?**
 - Write strategy


CS 135: Computer Architecture, Bhagi Narahari



Unified or Separate I-Cache and D-Cache

- **Two types of accesses:**
 - Instruction fetch
 - Data fetch (load/store instructions)
- **Unified Cache**
 - One large cache for both instructions and data
 - Pros: simpler to manage, less hardware complexity
 - Cons: how to divide cache between data and instructions? Confuses the standard harvard architecture model; optimizations difficult
- **Separate Instruction and Data cache**
 - Instruction fetch goes to I-Cache
 - Data access from Load/Stores goes to D-cache
 - Pros: easier to optimize each
 - Cons: more expensive; how to decide on sizes of each


CS 135: Computer Architecture, Bhagi Narahari



Definitions

- Cache has a total size – number of bytes in cache
- Transfers take place in **blocks**
 - A whole block is transferred between memory and cache
- Locating a block requires two attributes:
 - Size of block
 - Organization of blocks within the cache
- **Block size (also referred to as line size)**
 - Granularity at which cache operates
 - Each block is contiguous series of bytes in memory and begins on a naturally aligned boundary
 - Eg: cache with 16 byte blocks
 - each contains 16 bytes
 - First byte aligned to 16 byte boundaries in address space
 - Low order 4 bits of address of first byte would be 0000
 - **Smallest usable block size is the natural word size of the processor**
 - Else would require splitting an access across blocks and slows down translation

CS 135: Computer Architecture, Bhagi Narahari



Memory viewed as blocks

- If cache block size = K bytes, then memory can be viewed as contiguous set of blocks each of size K

16 byte
memory

16 byte memory,
With 4 byte sized
Cache blocks;
4 blocks of memory

CS 135: Computer Architecture, Bhagi Narahari

Where can a block be placed in a cache? - Cache Organization

- If each block has only one place it can appear in the cache, it is said to be "direct mapped"
 - and the mapping is usually $(\text{Block address}) \text{ MOD } (\text{Number of blocks in the cache})$
- If a block can be placed anywhere in the cache, it is said to be fully associative
- If a block can be placed in a restrictive set of places in the cache, the cache is set associative.
 - A set is a group of blocks in the cache. A block is first mapped onto a set, and then the block can be placed anywhere within that set.
 - $(\text{Block address}) \text{ MOD } (\text{Number of sets in the cache})$
 - if there are n blocks in a set, the cache is called n -way set associative

CS 135: Computer Architecture, Bhagi Narahari

Where can a block be placed in a cache?

Fully Associative

1 2 3 4 5 6 7 8

Block 12 can go anywhere

Direct Mapped

1 2 3 4 5 6 7 8

Block 12 can go only into Block 4
(12 mod 8)

Set Associative

1 2 3 4 5 6 7 8

Block 12 can go anywhere in set 0
(12 mod 4)

Memory:

1 2 3 4 5 6 7 8 9

CS 135: Computer Architecture, Bhagi Narahari

Cache Organizations

- **Direct Mapped vs Fully Associate**
 - Direct mapped is not flexible enough:
 - if $X \pmod K = Y \pmod K$ then X and Y cannot both be located in cache
 - Fully associative allows any mapping, implies all locations must be searched to find the right one – expensive hardware
- **Set Associative**
 - Compromise between direct mapped and fully associative
 - Allow many-to-few mappings
 - On lookup, subset of address bits used to generate an index
 - BUT index now corresponds to a set of entries which can be searched in parallel – more efficient hardware implementation than fully associative, but due to flexible mapping behaves more like fully associative

CS 135: Computer Architecture, Bhagi Narahari


Cache Design: How is data found in Cache?

- Processor generates address request – some N bit address
- **Two issues:**
 - How do we know if a data item is in the cache?
 - If it is, how do we find it? .. In which cache block do we look
- "direct mapped"

For each item of data at the lower level,
 there is exactly one location in the cache where it might be.

 e.g., lots of items at the lower level share locations in the upper level


CS 135: Computer Architecture, Bhagi Narahari



Addressing.

- **Memory address of N bits**
 - This is what is requested by processor
- **We need mapping from Addresses to Cache**
- **How to break up these N bits into fields so that the cache controller can easily determine if the address requested is already stored in cache ?**
- **Focus on Direct mapped:**
 - How to implement Mod.P where $P=2^K$
 - How to determine which cache block to map the address -- **index**
 - How to determine which of the words is the one in memory -- **tag**
 - Since we are working with blocks, how to find the specific word within the block -- **offset**
 - **N bit address is broken into these 3 fields!**

CS 135: Computer Architecture, Bhagi Narahari



Example


Memory

Memory block 1
Memory block 2
Memory block 3
Memory block 4
Memory block 5

Cache

Block
cache block 0
Cache block 1
Cache block 2
Cache block 3


CS 135: Computer Architecture, Bhagi Narahari



Addressing -- Example

- **8 bit address**
 - Byte addressability
- **4 byte cache blocks**
 - Each cache block has 4 bytes
- **Total size of cache = 16 bytes**
- **Which cache block do we place address 01101101**
- **How do we know which block from memory is in the cache block ?**
 - Is it 01101101 or 11001101

CS 135: Computer Architecture, Bhagi Narahari



Example

Memory

(0,1,2,3)
(4,5,6,7)
(8,9,10,11)
(12,13,14,15)
(16,17,18,19)

Addresses
in each block

Cache

Block
0
1
2
3

4 bytes in
each block
Addresses:
(0,1,2,3)

CS 135: Computer Architecture, Bhagi Narahari

Addressing -- Example

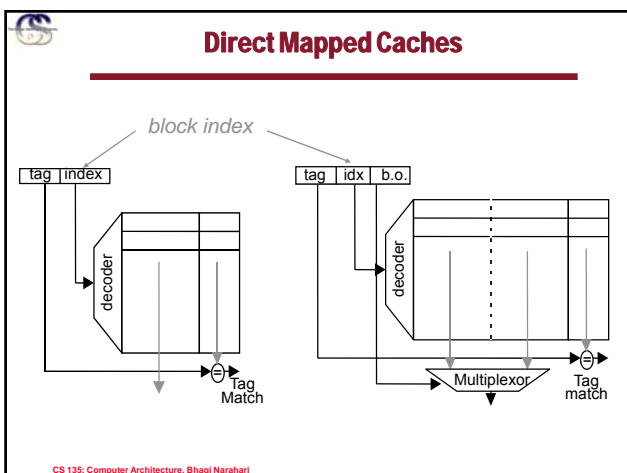
- 8 bit address
 - Byte addressability
- 4 byte cache blocks
 - Each cache block has 4 bytes = need 2 bits to specify which byte within the block
- Total size of cache = 16 bytes
 - 4 blocks in cache = apply Modulo 4 mapping
- Which cache block do we place 01101101
 - Last two LSB are offset within a block
 - Next 2 LSB are modulo 4 and specify cache block
- How do we know which block from memory is in the cache block ?
 - Is it 01101101 or 11001101

CS 135: Computer Architecture, Bhagi Narahari

So how complex is the hardware required ?

- Given N bit address, the cache controller needs to determine if the requested word is in cache or not
 - If not, then send the address to the memory bus
- This process has to be built into hardware
 - Complexity of the circuit determines time taken to determine if word is in cache
 - Hit time is function of complexity of circuit
 - For direct mapped, can you think of a hardware design ?

CS 135: Computer Architecture, Bhagi Narahari



Cache Block Replacement ?

- When we run out of space on the cache and need to bring in new block into cache
 - Replace a block that is currently in cache to create space
- Which block to replace ?
 - Easy in Direct mapped cache – no choice!
 - More complex in Associative Caches since we have a choice

CS 135: Computer Architecture, Bhagi Narahari

What happens on write?

- **Write through –**
 - Propagate each write through cache and on to next level written to both the block in the cache and to the block in lower level memory
- **Write back –**
 - The information is written only to the block in the cache, memory is written only when block is replaced
 - A dirty bit is kept in the block description area
- **Tradeoffs??**

CS 135: Computer Architecture, Bhagi Narahari

Performance

- **Simplified model:**

$$\text{execution time} = (\text{execution cycles} + \text{stall cycles}) \times \text{cycle time}$$

$$\text{stall cycles} = \# \text{ of instructions} \times \text{miss ratio} \times \text{miss penalty}$$
- **Two ways of improving performance:**
 - decreasing the miss ratio
 - decreasing the miss penalty

What happens if we increase block size?

CS 135: Computer Architecture, Bhagi Narahari

Performance

- **Increasing the block size tends to decrease miss rate:**

Block size (bytes)	gcc (blue diamonds)	spice (orange squares)	gcc (green triangles)	gcc (red circles)
4	~15%	~38%	~10%	~5%
16	~12%	~25%	~8%	~4%
64	~10%	~20%	~7%	~3%
256	~10%	~28%	~7%	~3%

- **Use split caches because more spatial locality in code:**

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

CS 135: Computer Architecture, Bhagi Narahari

Decreasing miss penalty with multilevel caches

- **Add a second level cache:**
 - often primary cache is on the same chip as the processor
 - use SRAMs to add another cache above primary memory (DRAM)
 - miss penalty goes down if data is in 2nd level cache
- **Using multilevel caches:**
 - try and optimize the hit time on the 1st level cache
 - try and optimize the miss rate on the 2nd level cache

CS 135: Computer Architecture, Bhagi Narahari



Summary: Memory Access time optimization

- If each access to memory leads to a **cache hit** then time to fetch from memory is one cycle
 - Program performance is good!
- If each access to memory leads to a **cache miss** then time to fetch from memory is much larger than 1 cycle
 - Program performance is bad!
- Design Goal:
How to arrange data/instructions so that we have as few cache misses as possible.

CS 135: Computer Architecture, Bhagi Narahari



How about rewriting code to improve cache rates?

CS 135: Computer Architecture, Bhagi Narahari



Secondary Memory

- CPU fetches from memory, memory is a sequence of 2^N locations
- What happens if main memory (chip capacity) is less than 2^N
- Data and programs may be stored in a non-volatile component
 - MS-Word executable is on disk
 - Your application data
- What happens if more than one process is running on the system
 - Multiprogramming
 - What is the address space available to each user ?
- Need to use Disks!


CS 135: Computer Architecture, Bhagi Narahari



What is multiprogramming and Why?

- Processor overlaps execution of two processes/programs
 - When one is waiting for I/O, the other is "swapped" in to the processor
 - Save the "state" of the process being swapped out
- Processes need to share memory
 - Each has its own address space
- Leads to better throughput and utilization


CS 135: Computer Architecture, Bhagi Narahari



The Complete memory hierarchy

- Processor has a set of registers
 - Processor instructions operate on contents in the registers
- Small, fast cache memory placed near the processor
- Main memory sitting outside the chip
 - If data is not in the cache then fetch from main memory
 - Takes longer to access main memory than cache
- Disk sitting outside the motherboard
 - If data/program not in main memory then fetch/load from disk
 - Takes much longer to access disk than main memory


CS 135: Computer Architecture, Bhagi Narahari



Why does memory hierarchy work ?


- Principle of locality!
 - Exploit at each level.

CS 135: Computer Architecture, Bhagi Narahari



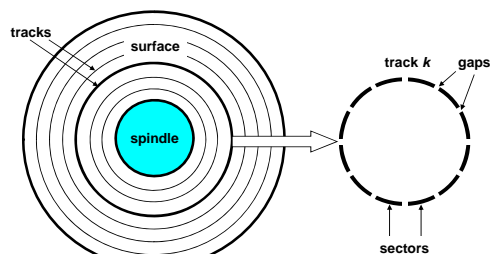
How do disks work ?

CS 135: Computer Architecture, Bhagi Narahari



Disk Geometry

- Disks consist of **platters**, each with two **surfaces**.
- Each surface consists of concentric rings called **tracks**.
- Each track consists of **sectors** separated by **gaps**.



The diagram illustrates the physical structure of a disk. On the left, a series of concentric circles represent tracks on a central spindle. Labels point to 'tracks', 'surface', and 'spindle'. An arrow points from one of the tracks to a magnified view on the right. This magnified view shows a single track divided into segments labeled 'sectors', with the spaces between them labeled 'gaps'. A specific track is labeled 'track k'.

CS 135: Computer Architecture, Bhagi Narahari

Disk Geometry (Multiple-Platter View)

- Aligned tracks form a cylinder.

The diagram illustrates the physical structure of a disk drive. It shows three platters (0, 1, and 2) stacked vertically, each with two surfaces (0-5). A central spindle is shown. A cylinder k is highlighted, showing that tracks at the same radial distance across different platters form a cylinder.

CS 135: Computer Architecture, Bhagi Narahari

Disk Capacity

- Capacity:** maximum number of bits that can be stored.
 - Vendors express capacity in units of gigabytes (GB), where $1 \text{ GB} = 10^9$.
- Capacity is determined by these technology factors:
 - Recording density** (bits/in): number of bits that can be squeezed into a 1 inch segment of a track.
 - Track density** (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment.
 - Areal density** (bits/in²): product of recording and track density.
- Modern disks partition tracks into disjoint subsets called **recording zones**
 - Each track in a zone has the same number of sectors, determined by the circumference of innermost track.
 - Each zone has a different number of sectors/track

CS 135: Computer Architecture, Bhagi Narahari

Disk Operation (Single-Platter View)

- The disk surface spins at a fixed rotational rate.

The diagram shows a single platter with concentric tracks and a central spindle. A read/write head is attached to an arm that can move radially. The head flies over the disk surface on a thin cushion of air.

The read/write head is attached to the end of the arm and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.

CS 135: Computer Architecture, Bhagi Narahari

Disk Operation (Single-Platter View)

- The disk surface spins at a fixed rotational rate.

The diagram shows a single platter with concentric tracks and a central spindle. A read/write head is attached to an arm that can move radially. The head flies over the disk surface on a thin cushion of air.

The read/write head is attached to the end of the arm and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.

CS 135: Computer Architecture, Bhagi Narahari

Disk Access Time

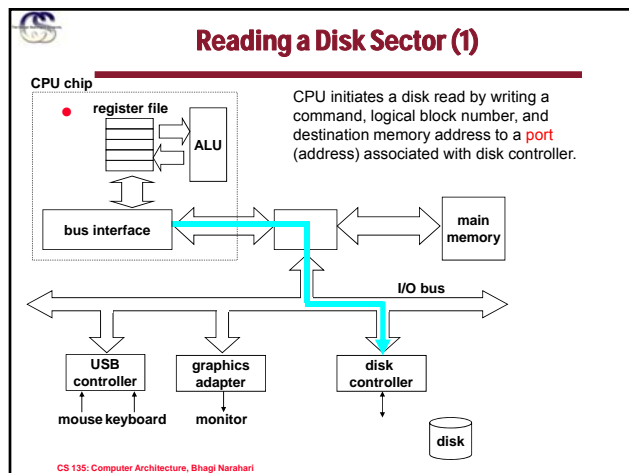
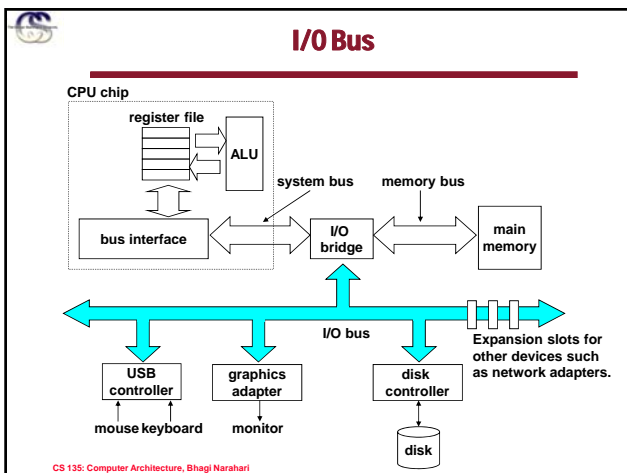
- Average time to access some target sector approximated by :
 - $T_{\text{access}} = T_{\text{avg seek}} + T_{\text{avg rotation}} + T_{\text{avg transfer}}$
- **Seek time** (Tavg seek)
 - Time to position heads over cylinder containing target sector.
 - Typical Tavg seek = 9 ms
- **Rotational latency** (Tavg rotation)
 - Time waiting for first bit of target sector to pass under r/w head.
 - $T_{\text{avg rotation}} = 1/2 \times 1/\text{RPMs} \times 60 \text{ sec}/1 \text{ min}$
- **Transfer time** (Tavg transfer)
 - Time to read the bits in the target sector.
 - $T_{\text{avg transfer}} = 1/\text{RPM} \times 1/(\text{avg \# sectors/track}) \times 60 \text{ secs}/1 \text{ min}$.

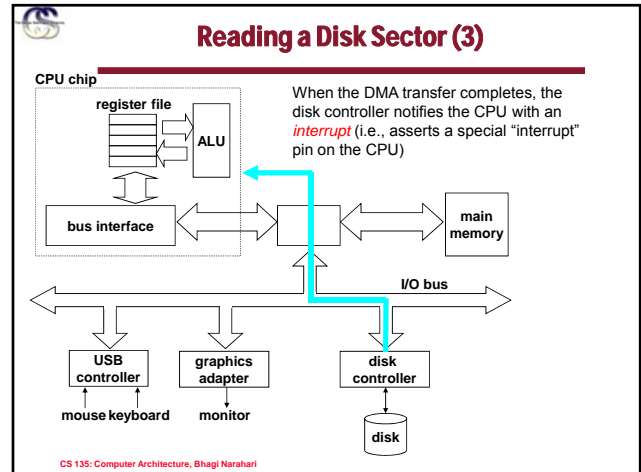
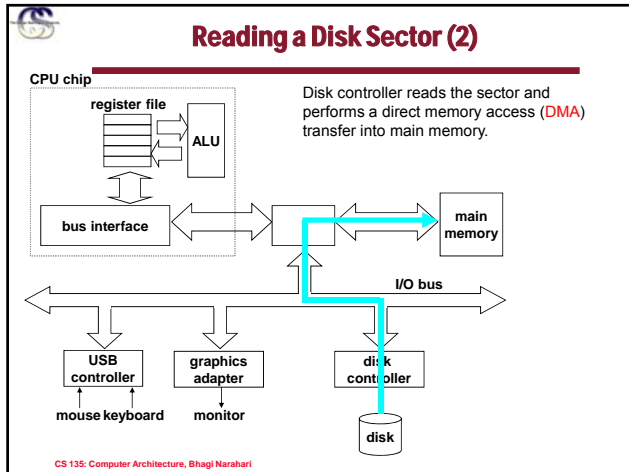
CS 135: Computer Architecture, Bhagi Narahari

Logical Disk Blocks

- Modern disks present a simpler abstract view of the complex sector geometry:
 - The set of available sectors is modeled as a sequence of b-sized **logical blocks** (0, 1, 2, ...)
- **Mapping between logical blocks and actual (physical) sectors**
 - Maintained by hardware/firmware device called **disk controller**.
 - Converts requests for logical blocks into (surface, track, sector) triples.
- Allows controller to set aside spare cylinders for each zone.
 - Accounts for the difference in "**formatted capacity**" and "**maximum capacity**".

CS 135: Computer Architecture, Bhagi Narahari





Storage Trends

	metric	1980	1985	1990	1995	2000	2000:1980
SRAM	\$/MB	19,200	2,900	320	256	100	190
	access (ns)	300	150	35	15	2	100
DRAM	\$/MB	8,000	880	100	30	1	8,000
	access (ns)	375	200	100	70	60	6
	typical size(MB)	0.064	0.256	4	16	64	1,000
Disk	\$/MB	500	100	8	0.30	0.05	10,000
	access (ms)	87	75	28	10	8	11
	typical size(MB)	1	10	160	1,000	9,000	9,000

(Culled from back issues of Byte and PC Magazine)

CS 135: Computer Architecture, Bhagi Narahari

CPU Clock Rates

	1980	1985	1990	1995	2000	2000:1980
processor	8080	286	386	Pent	P-III	
clock rate(MHz)	1	6	20	150	750	750
cycle time(ns)	1,000	166	50	6	1.6	750

CS 135: Computer Architecture, Bhagi Narahari

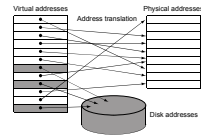
Last step: Virtual Memory

- N bit address space...
- If word is not in main memory then it is on disk – need a controller to manage this ..analogous to cache controller
 - Virtual memory management system
- Note: transfers take place from disk in “blocks” of M bytes (sector) – **page** of data
- Memory consists of a number of pages
 - Determine if the page is in main memory or not
- N bit address broken into fields which determine page number, and whether page is in the memory or disk
- If page size is 1024 bits...how to organize the N bit address ??

CS 135: Computer Architecture, Bhagi Narahari

Virtual Memory

- Main memory can act as a cache for the secondary storage (disk)

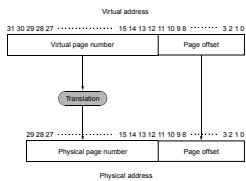


- Advantages:
 - illusion of having more physical memory
 - program relocation
 - protection

CS 135: Computer Architecture, Bhagi Narahari

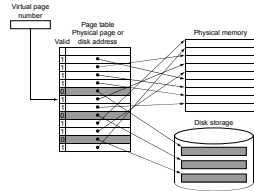
Pages: virtual memory blocks

- Page faults: the data is not in memory, retrieve it from disk
 - huge miss penalty, thus pages should be fairly large (e.g., 4KB)
 - reducing page faults is important (LRU is worth the price)
 - can handle the faults in software instead of hardware
 - using write-through is too expensive so we use writeback

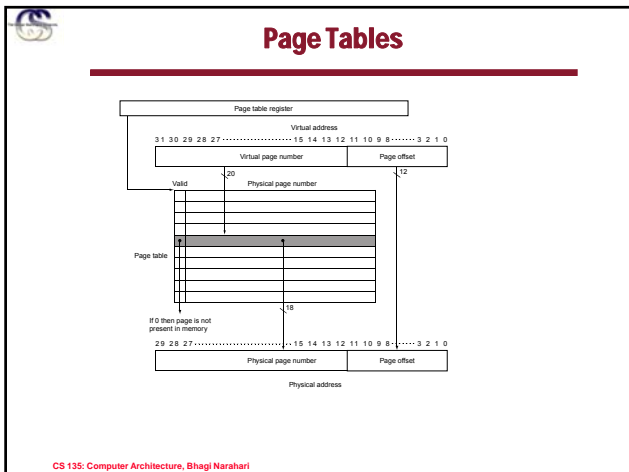


CS 135: Computer Architecture, Bhagi Narahari

Page Tables



CS 135: Computer Architecture, Bhagi Narahari



More on Virtual memory

- This is part of the Operating system
 - CS 156 focus is the OS utilities

CS 135: Computer Architecture, Bhagi Narahari