

Program Performance

Part 1:

Memory Design

1

Course Objectives: Where are we....

- Bits&bytes: Logic devices
 - HW building blocks
- Processor: ISA, datapath
 - Using building blocks to assemble a processor (LC3)
- Programming the processor: Assembly
- Translating high level programs to Proc
 - Implementing C on LC3
- Working with C
 - Need fluency in C to get a good understanding of "systems" topics

2

2

Executing High level Programs

- User application written in high level language
- Program runs on a processor
- How are high level programs implemented on processor ?
 - Run-time stack, allocation of variables, translation of high level code to machine code
 - Map high level data structures to low level data structures
 - Struct to linear mapping in memory
- What else does software developer want after program is implemented correctly ?
- PERFORMANCE!

3

3

Next...the end is near !

- Performance of programs
 - What to measure
 - Model ?
 - Technology trends
- Memory organization basics
 - Memory hierarchy: cache, main memory, etc.
- How to rewrite your program to make it run faster...code optimization....Project 6 (i.e, take home exam)
- “real” processors...how to improve performance
 - Pipelining, ILP, Multi-core

4

4

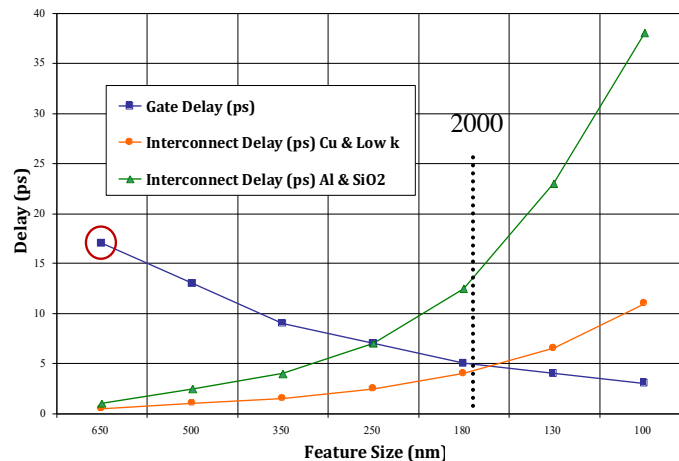
Technology Trends & Performance

- Speed will depend on clock cycle (frequency) of the circuits
 - How fast can we switch the transistors
 - Feed the signal to the gate of MOS transistor, how long for the transistor to throw the switch
 - How large is the transistor – feature size
- **Moore's Law**
 - Founder of Intel hypothesized on rate of increase in performance
 - It is not a law in the sense of laws of physics, etc.
 - Observations: performance doubles every 18 months
 - If you knew this, how would it guide your business decisions?
 - Would you bet on hardware IP or software IP ?

5

5

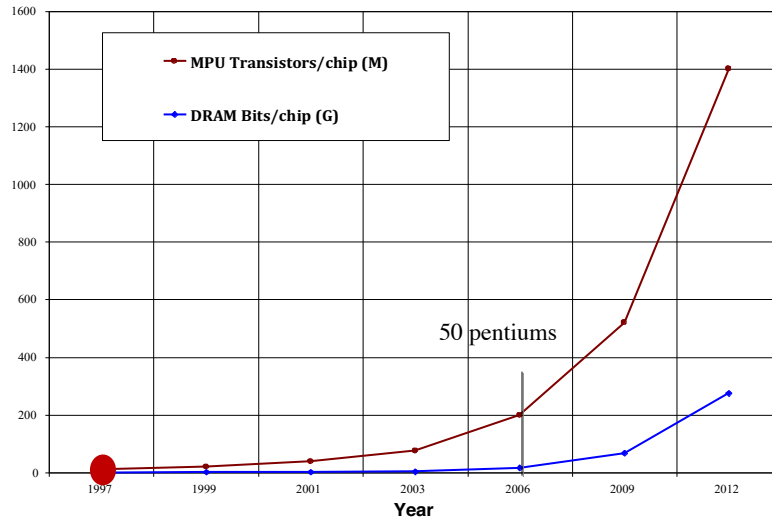
Tech Trends: Delay vs. Feature/Transistor Size



Bohr, M. T., "Interconnect Scaling - The Real Limiter To High Performance ULSI", Proceedings of the IEEE International Electron Devices, pages 241-242.

6

How many Transistors can we pack into a single Chip



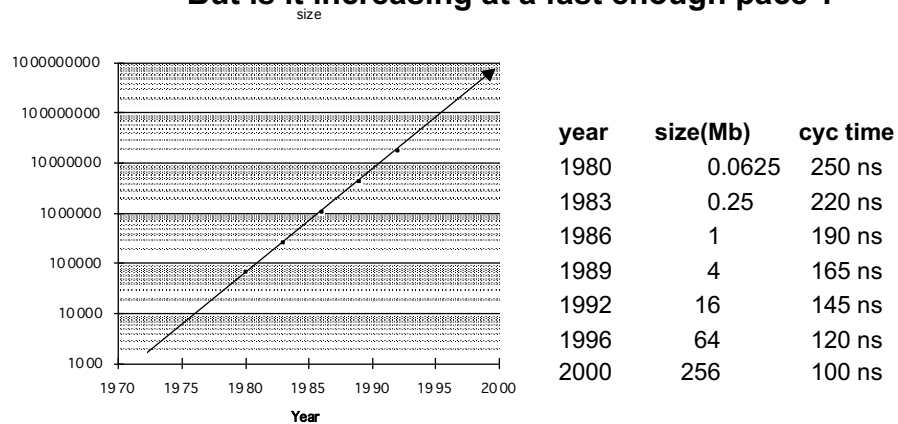
Big problem with heat dissipation & power

7

Tech Trends: Memory Capacity (Single Chip DRAM)

Memory density is also increasing...

But is it increasing at a fast enough pace ?

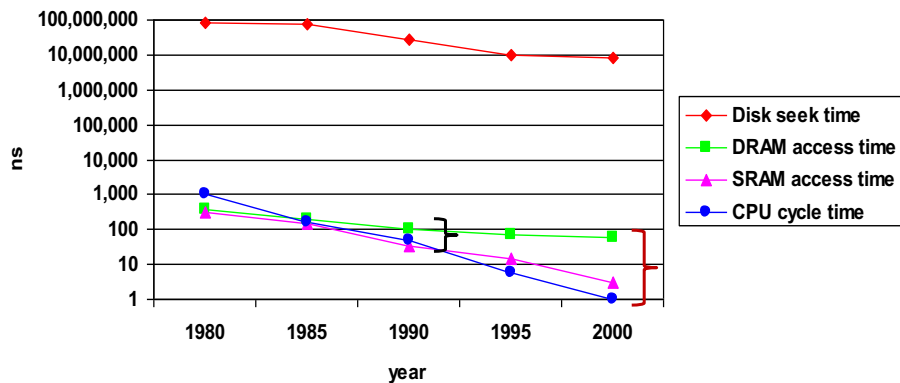


8

8

The CPU-Memory Gap

- The increasing gap between DRAM, disk, and CPU speeds...



9

9

Performance Trends: Summary

- Workstation performance (measured in Spec Marks) improves roughly 50% per year (2X every 18 months)
 - Performance will include not just processor, but memory and disk I/O
- Improvement in cost **performance** estimated at 70% per year

10

10

Performance of Programs

- “Complexity” of algorithms
- How good/efficient is your algorithm
 - Measure using Big-Oh notation: $O(N \log N)$
- Next question : How well is the code executing on the machine ???????
 - Actual time to run the program
 - Effect of H/W features on SW performance

11

11

How to Model Performance

- The asymptotic complexity – “big O”
 - Time = $O(f(n))$: function of the size of the input
 - Sorting $O(n \log n)$
 - This measures efficiency of your algorithm
 - i.e., how ‘good’ is solution technique
 - Is this enough when we talk of actual time measured on the processor
???
- *There’s more to performance than asymptotic complexity*
 - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- Must understand system to optimize performance
 - How programs are compiled and executed, data storage, data structures, I/O management

12

12

Performance – what to measure ?

Which of these airplanes has the better performance ?

Plane	DC to Paris	Speed	Passengers	Performance ?
Airbus A380	7.5 hours	730 mph	500	
BAD/Sud Concorde	3 hours	1350 mph	130	

13

13

The Bottom Line: Performance metric depends on application

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	7.5 hours	730 mph	500	365,000
BAD/Sud Concorde	3 hours	1350 mph	130	175,500

- **Time to run the task (Execution Time/Response Time/Latency)**
 - Time to travel from DC to Paris
- **Tasks per unit time (Throughput/Bandwidth)**
 - Passenger miles per hour; *how many passengers transported per unit time*

14

14

Computer Performance: TIME, TIME, TIME

- Response Time (latency)
 - How long does it take for my job to run?
 - How long does it take to execute a job?
 - How long must I wait for the database query?
- Throughput
 - How many jobs can the machine run at once?
 - What is the average execution rate?
 - How much work is getting done?

Metric chosen usually depends on user community: sys admin vs single user ?

- *If we upgrade a machine with a new processor what do we increase?*
- *If we add a new machine to the cluster/lab what do we increase?*

15

15

Execution Time

- Elapsed Time
 - counts everything (*disk and memory accesses, I/O , etc.*)
 - a useful number, but often not good for comparison purposes
- CPU time
 - doesn't count I/O or time spent running other programs
 - can be broken up into system time, and user time
- Our focus in this course: **user CPU time**
 - time spent executing the lines of code that are "in" our program

16

16

Processor time: how to measure ?

- Number of clock cycles it takes to complete the execution of your program
- What is your program
 - A number of instructions
 - Different types: load, store, ALU, branch
 - **Stored in memory**
 - Executed on the CPU

17

17

Model for CPU Performance

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

$$CPU = IC * CPI * Clk$$

simple but very effective and standard way to measure

18

18

The three parameters...who determines them?

- To improve performance, reduce CPU execution time....
 - i.e., reduce $CPU = IC * CPI * Clk$
 - *Reduction in any one of these will reduce the time and increase perf.*
- **Clock cycle**: depends on semi-conductor (& fabrication) technology
 - How fast can we switch the transistor (how small is transistor)
- **CPI**: depends on the ISA...and on program/compiler (!!)
 - Different operations take different time
- **Instruction Count (IC)**: depends on program and on compiler
 - Compiler generates the machine code
 - Number of instructions would depend on your algorithm as well as on compiler and the programming language

19

19

CPI

- Cycles per instruction: Different instructions may take different time
 - Example in LC 3 ?
- observe that not every instruction needs to go through all the instruction execution steps
 - Eg: no need to calculate effective address, fetch from memory or registers
- **Reality #1**: different times associated with different operations
 - Especially true of memory operations
- **Reality #2**: the 'average' CPI depends on the instruction mix in the program
 - How many ALU operations, how many load/store, etc.
 - Weighted average (since each type takes different no. of cycles)

20

20

Average CPI

- Application has an “instruction mix”
 - Profile of application instruction types
 - ALU, Load/Store (memory), Branch, Jumps, etc.
 - x_1, x_2, x_3, \dots as percentage ($x_1=0.4$)
- Processor has CPI for each type of instruction
 - Part of ISA of a processor....specifications doc
 - Example: ALU=1.0 cycle, Load/Store=2.0 cycle, etc.
 - t_1, t_2, t_3, \dots
- What is effective CPI ?
- Weighted average
 - $CPI = x_1 * t_1 + x_2 * t_2 + \dots$

21

21

CPI: Cycles per instruction

- **Depends on the instruction**

$$CPI_i = \text{Execution time of instruction } i / \text{Cycle time}$$

- **Average cycles per instruction**

$$CPI = \sum_{i=1}^n CPI_i * F_i \quad \text{where } F_i = \frac{IC_i}{IC_{tot}}$$

- **Example:**

Op	Freq	Cycles	CPI _i	%time
ALU	50%	1	0.5	33%
Load	20%	2	0.4	27%
Store	10%	2	0.2	13%
Branch	20%	2	0.4	27%
CPI _{total}			1.5	

22

Improving Performance of processors: ...

- Are “real” processors like LC 3?
- Design principles to improve the performance of the processor ?
 - Pipelined processors: overlap execution of instructions
 - Superscalar processors: have multiple pipelined execution units
 - Multi-threaded processors: execute multiple threads
 - Multi-core: executed multiple threads and multiple programs on many cores/processors on a single chip
- *will return to this....*

23

23

Improving System Performance: Principles of Computer Architecture Design: Thumb Rules

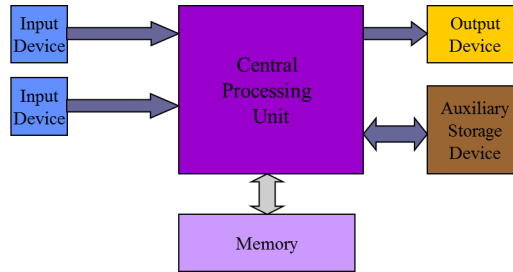
- Common case fast
 - Focus on improving those instructions that are frequently used
 - **Amdahl's Law**
 - Fraction enhanced/optimized runs faster
 - Where to focus optimizations....maximize “returns”
- Principle of Locality:
 - program spends 90% of its time in 10% of code
 - Eg: word processing
 - Spatial: items near each other tend to be accessed
 - Temporal: recently used items tend to be used again
- Concurrency/Parallelism
 - Overlap the instruction execution steps
 - Pipeline processors
 - Multi-core processors

24

24

Next : Role of Memory organization on performance

- Key component in von Neumann computer ?
- Memory



- How are “real” memory systems organized ?
 - How do they affect performance ?

25

25

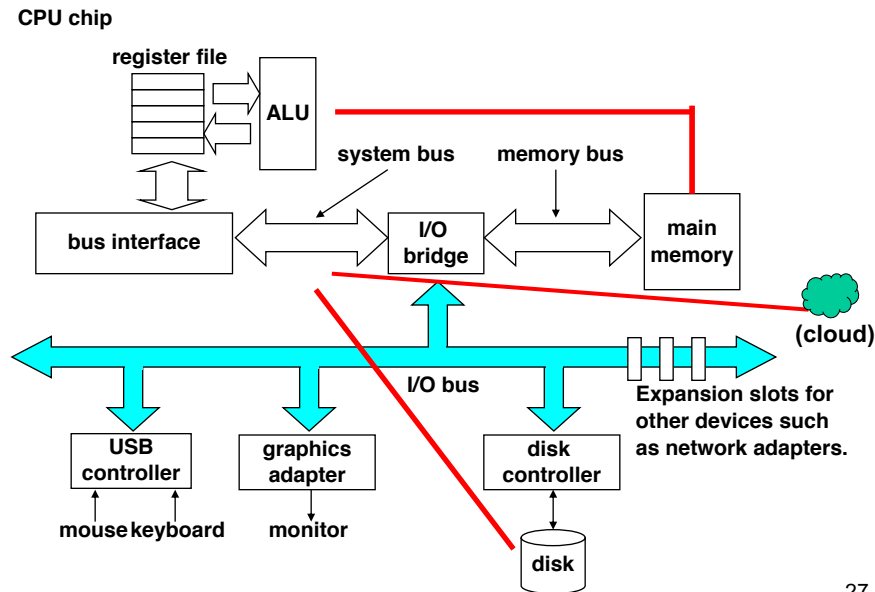
Recall – Programmers view of Memory Unit

- 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)
 - Time depends on how is memory actually organized ?
 - Can everything we need fit into a memory that is close to the CPU ?

26

26

Today's Memory Systems...CPU and I/O Bus



27

27

Memory Technology

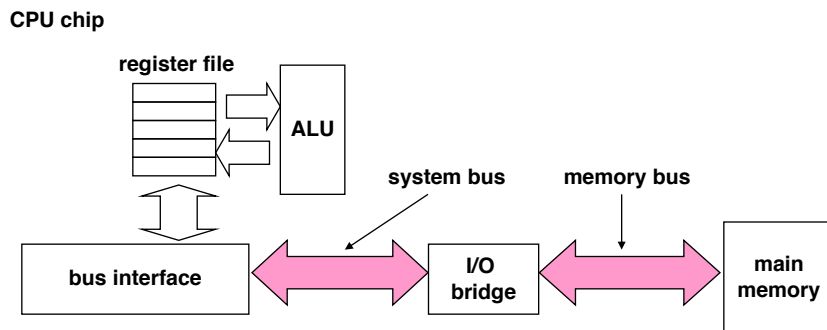
- Random access memory
 - Can read from any location by supplying address of data
 - This is the model we have been using
 - Other types: sequential access....tapes anyone ?
- Memory Comes in Many Flavors
 - Main RAM memory Key features
 - **RAM** is packaged as a chip.
 - Basic storage unit is a **cell** (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. – *Non-Volatile*
 - Read only memories – store OS
 - “Secondary memory” Disks, Tapes, Flash etc.
- Difference in speed, price and “size”
 - Fast is small and/or expensive
 - Large is slow and/or cheap

28

28

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.

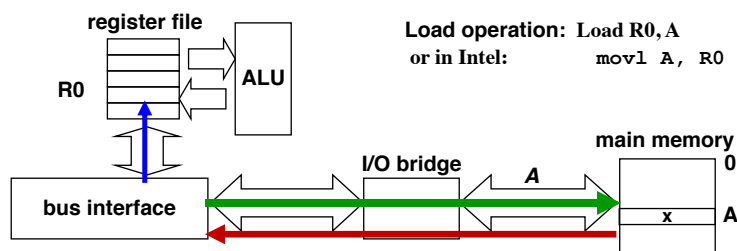


29

29

Memory Read Transaction

1. CPU places address *A* on memory bus
2. main memory reads *A* from bus, retrieves word *x*, and places it on bus
3. CPU reads word *x* from the bus and copies it into register



30

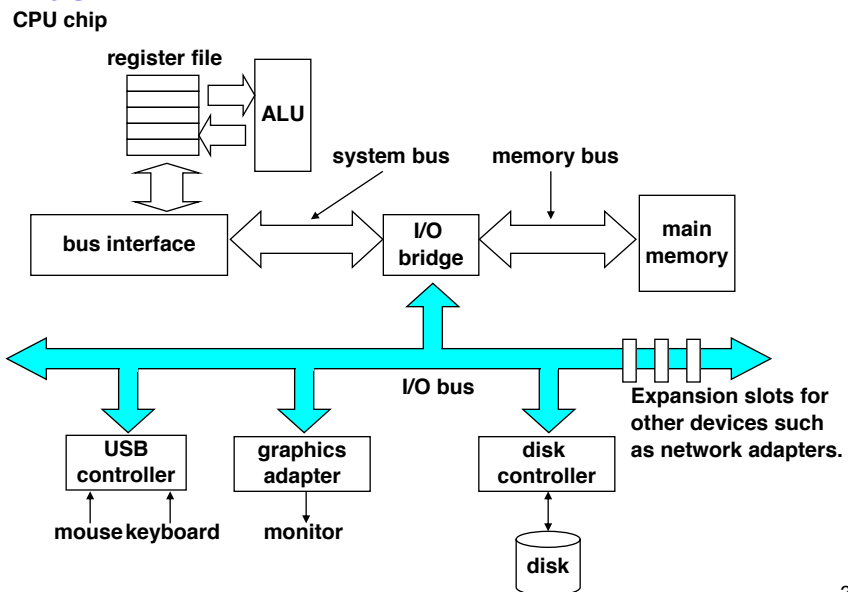
30

Reality...Memory access

31

31

I/O Bus



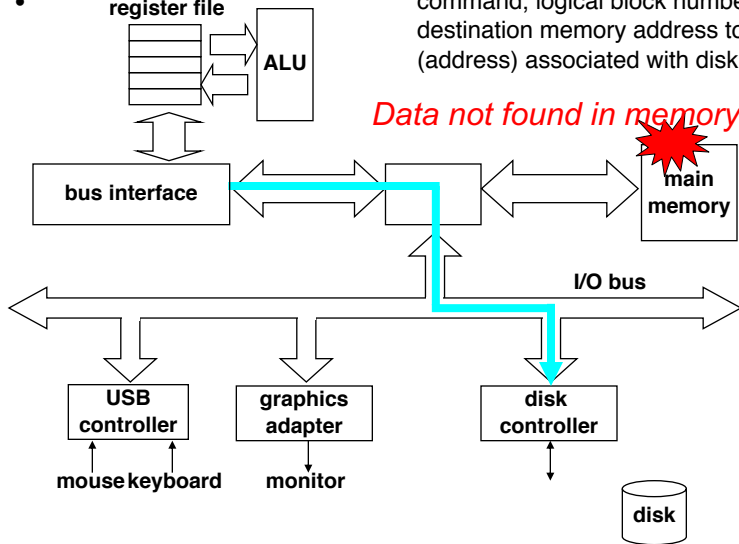
32

32

What happens if not found in main memory ...read from Disk

CPU chip

-



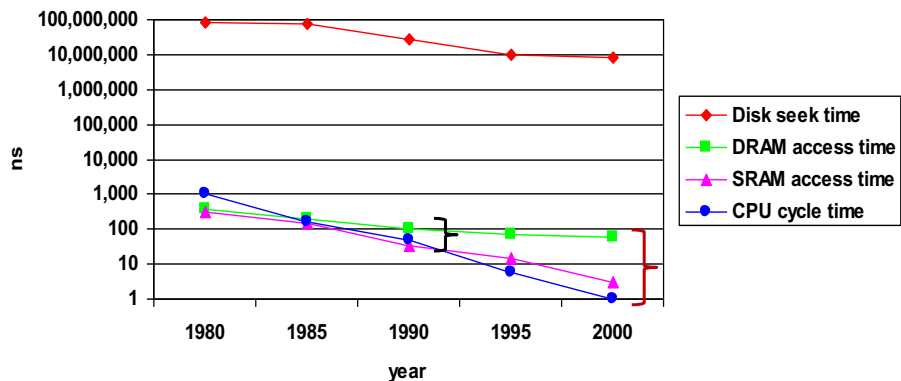
CPU initiates a disk read by writing a command, logical block number, and destination memory address to a **port** (address) associated with disk controller.

33

33

Recall the CPU-Memory Gap

- The increasing gap between DRAM, disk, and CPU speeds...



34

34

Link to Performance?

- CPU time = IC * CPI * Clk
- So what is the CPI for a memory access
 - Load/Store of data or
 - Instruction fetch
- 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

35

35

Memory Access time and Performance ?

CPU time	=	Seconds	=	Instructions	x	Cycles	x	Seconds
		Program		Program		Instruction		Cycle

$$CPU = IC * CPI * Clk$$

- How does memory access time fit into CPU time equation ?
- More stall cycles = increase in CPI

36

36

Summarize CPU & Memory Interaction

- 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 ?

37

37

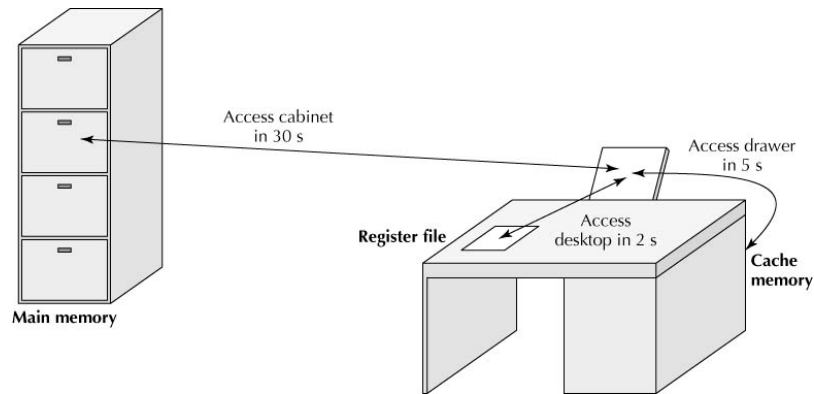
Memory Hierarchies

- 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
- Why does this work....principle of locality

38

38

“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!

39

Why does this work: Key Concept – 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.
 - Nearby in time
 - **Spatial locality:** Items with nearby addresses tend to be referenced close together in time.
 - Nearby in space

40

40

Locality: Example

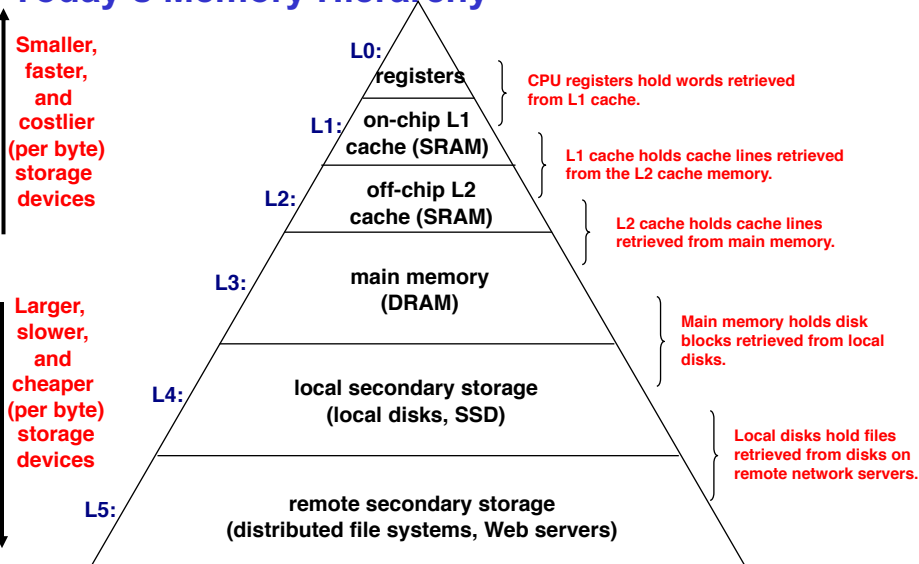
```
sum = 0;
for (i = 0; i < n;
    i++)
    sum += a[i];
return sum;
```

- **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**

41

41

Today's Memory Hierarchy



42

42

Another look at Performance equation: Simplified Goal ?

- CPU execution time = IC * CPI * Clock
 - More stall cycles = more time
- Improve performance = decrease stall cycles
 - Decrease time to access memory
 - *How ?*
 - *Organize memory as a hierarchy!*
 - *Most of the time you should access data in the fastest memory*
 - *Why does it work – principle of locality of programs*

43

43

Next: Design of Memory Hierarchy

- Focus on Cache design
 - Brief overview of cache design
 - How does Cache memory work ?
 - How are addresses mapped to Cache
 - Can we design a circuit to implement cache memory control ?
- How to rewrite code to get better cache performance: **code optimization**

44

44

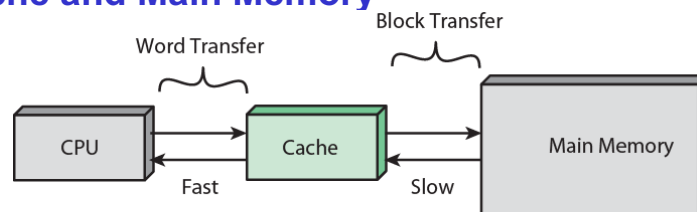
Why Cache ?

- 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
- What ? : Place a small but fast memory close to the processor
- How ? : Why does this work
 - Principle of Locality

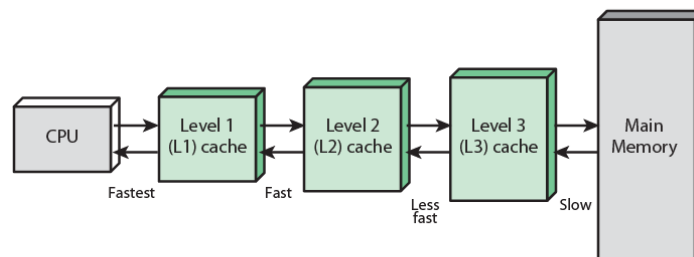
45

45

Cache and Main Memory



(a) Single cache



(b) Three-level cache organization

46

46

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

47

47

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!

48

48

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
-

49

49

Memory Access times using Simplified Model

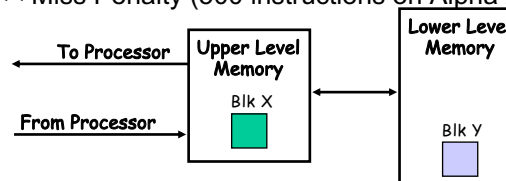
- If data is found in Cache then time = 1
 - Called a cache hit
- Else time is Main memory access time
 - Cache miss, means read from next level
- Note: need a 'control unit' to determine if location is in cache or not
 - Cache controller
- Why does concept of caching work ?
 - Principle of Locality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

50

50

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** = $1 - (\text{Hit Rate})$
 - **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)



51

51

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!**

52

52

Average Memory Access Time

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

- **Hit time**: basic time of every access.
 - Always look to check if data is in cache
- **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
 - Access time to read/write to memory module
 - Can extend the same equation to disks/networks

53

53

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 ?

54

54

Example. .

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

55

55

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
- The **Cache Controller** hardware must look at this N bit address and decide (a) if it is in cache and (b) where to place it in cache ?

56

56

Cache Memory Design: Definitions

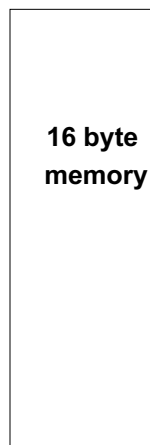
- Cache has a total size – number of bytes in cache
- Transfers take place in **blocks**
 - A whole block is transferred between memory and cache
- Locating a block requires two attributes:
 - Size of block
 - Organization of blocks within the cache
- **Block size** (also referred to as **line size**)
 - Smallest usable block size is the natural word size of the processor
 - Else would require splitting an access across blocks and slows down translation

57

57

Memory viewed as blocks

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

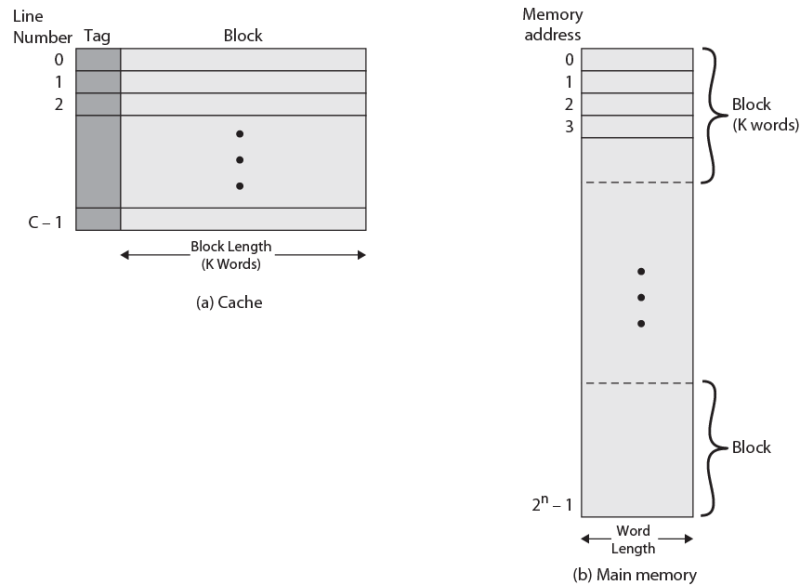


**16 byte memory,
With 4 byte sized
Cache blocks;
4 blocks of memory**

58

58

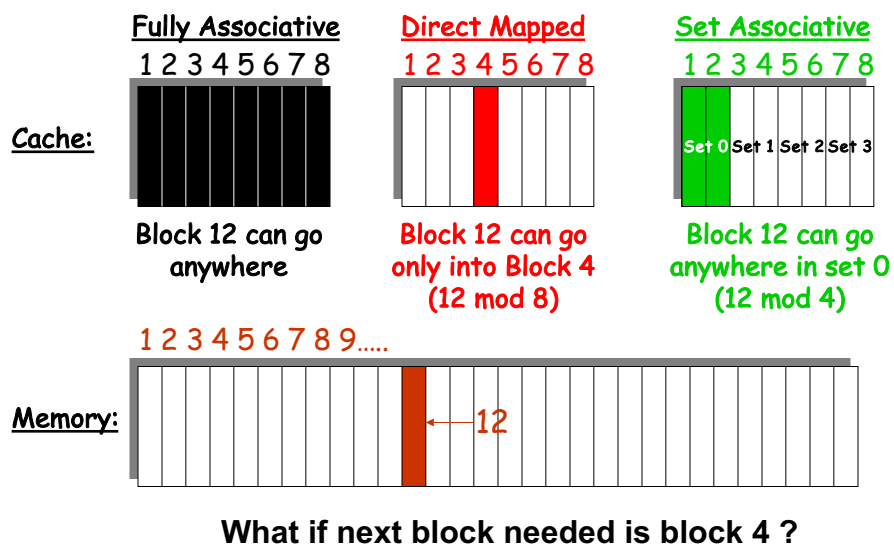
Cache/Main Memory Structure



59

59

Where can a block be placed in a cache?



60

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”
 - 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

61

61

Cache Design: How is data found in Cache?

- Note: Time to find data is the hit time: affects performance
- 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

62

62

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 ?

63

63

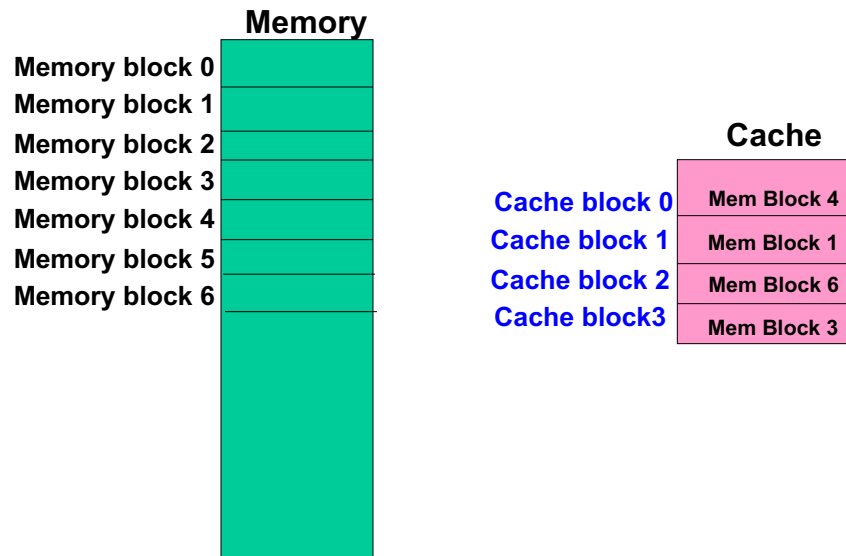
Addressing.

- Focus on Direct mapped:
 - **How to implement $N \text{ Mod } P$ 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!**

64

64

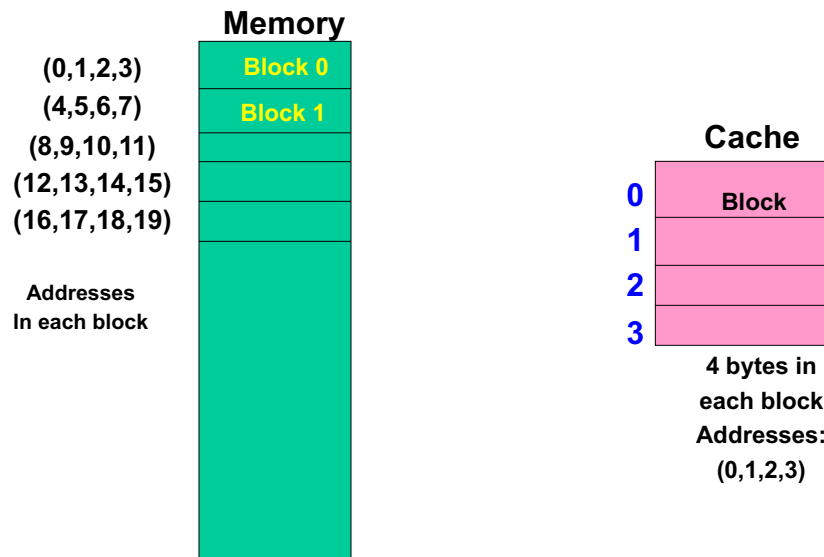
Example



65

65

Example



66

66

Addressing -- Example

- 8 bit address
 - Byte addressability
- 4 byte cache blocks
 - Each cache block has 4 bytes
 - Main memory has $2^8 = 256$ bytes....or 64 blocks
- Total size of cache = 16 bytes
 - 4 blocks, each of 4 bytes
- Into 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

67

67

Addressing -- Example

- 8 bit address
 - Byte addressability & 256 bytes in memory
- 4 byte cache blocks
 - Each cache block has 4 bytes = need 2 bits to specify which byte within the block
 - 4 bytes in each block – memory is $256/4 = 64$ blocks
- Total size of cache = 16 bytes
 - 4 blocks in cache = apply Modulo 4 mapping
- Into 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

68

68

Addressing -- Example

- The three fields are:

- **Tag**: the 4 most significant bits
 - Tell you which of the memory blocks are in the cache
- **Offset** (block offset): the 2 least significant bits
 - Gives the address within the cache block
- **Index**: the remaining 2 bits
 - Tells you into which cache block a memory address is placed

- In general: For N bit address, 2^K bytes in cache block, 2^j blocks of total cache size

- **Tag of $(N-k-j)$ bits**: the most significant bits
- **Offset of K bits**: the least significant k bits
- **Index of j bits**

- Memory block M gets placed in cache block $(M \bmod 2^j)$.

- M is $(N-k)$ most significant bits

69

69

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 ?

70

70

Finding cache block

- Given the i bits of address, how do you find the cache block ?
 - 01101101
 - Mod 4 (Mod 2^j where 2^j is number of cache blocks)
- Given j bits of index, one of these is selected...
- what device ?
- Decoder!

71

71

Finding word within a block

- Following four addresses are all in one block
 - 01101100
 - 01101101
 - 01101110
 - 01101111
- How do you select one of these four ?
- Multiplexer !

72

72

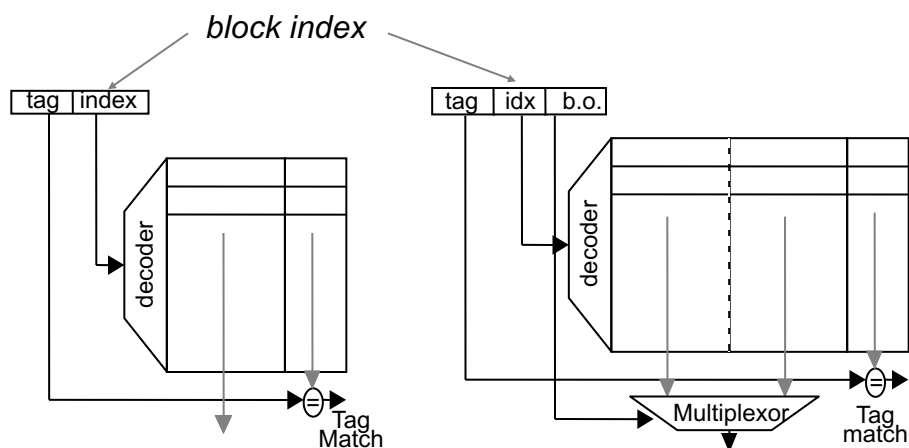
Matching the tag of a memory block

- Number of memory blocks could all be mapped to the same cache block
- Following two will get mapped to same block
 - 01101101
 - 01111101
- You are searching to see if **0110**1101 is in the cache.....
- How do you check if cache contains this block with tag 0110 ?
- **Comparator**

73

73

Direct Mapped Caches



74

74

Performance Impact of cache parameters

- Cache performance is related to cache design
 - Size of cache, size of each block, organization
- Two ways of improving performance:
 - decreasing the miss ratio
 - decreasing the miss penalty

75

75

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.
- How about rewriting the code to improve the cache hit rates ?
 - This is part of what code optimization accomplishes!!

76

76

Notes on Secondary Memory (Disks) and Virtual Memory

77

77

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

78

78

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 Secondary memory (Disk drive) !

79

79

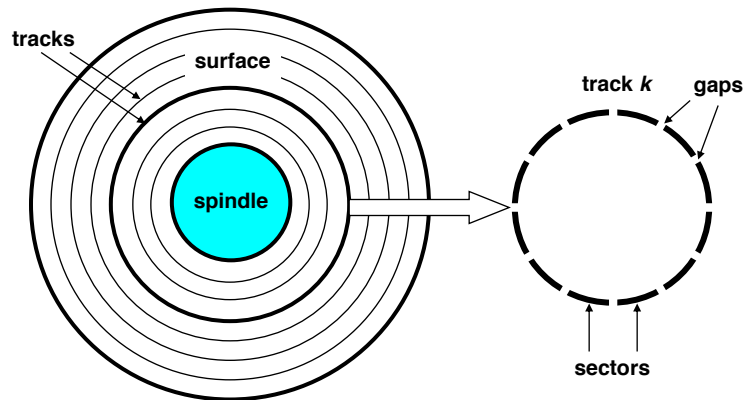
How do disks work ?

80

80

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**.

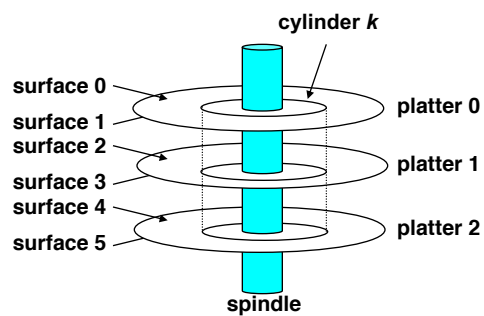


81

81

Disk Geometry (Multiple-Platter View)

- Aligned tracks form a cylinder.

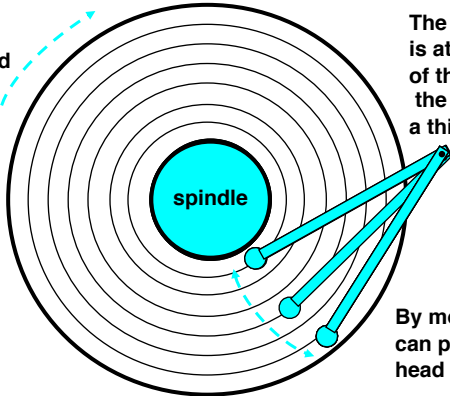


82

82

Disk Operation (Single-Platter View)

- The disk surface spins at a fixed rotational rate



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.

83

83

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** ($T_{\text{avg seek}}$)

 - Time to position heads over cylinder containing target sector.
 - Typical $T_{\text{avg seek}} = 9 \text{ ms}$
- Rotational latency** ($T_{\text{avg rotation}}$)

 - Time waiting for first bit of target sector to pass under r/w head.
 - $T_{\text{avg rotation}} = \frac{1}{2} \times \frac{1}{\text{RPMs}} \times 60 \text{ sec/1 min}$
- Transfer time** ($T_{\text{avg transfer}}$)

 - Time to read the bits in the target sector.
 - $T_{\text{avg transfer}} = \frac{1}{\text{RPM}} \times \frac{1}{(\text{avg \# sectors/track})} \times 60 \text{ secs/1 min.}$

84

84

Accessing a Disk Page

- Time to access (read/write) a disk block:
 - *seek time* (moving arms to position disk head on track)
 - *rotational delay* (waiting for block to rotate under head)
 - *transfer time* (actually moving data to/from disk surface)
- Seek time and rotational delay dominate.
- Key to lower I/O cost: **reduce seek/rotation delays!**
Hardware vs. software solutions?

85

85

Placement of Data on Disk

- Placement of data on disk can affect performance of program
- Example: If you are reading an entire array, then:
 - place the data in consecutive disk blocks
 - seek time to get first disk block
 - Remaining blocks will incur no seek time
 - **$\text{Time} = T_{\text{seek}} + N(T_{\text{transfer}} + T_{\text{rot}})$**
 - Naïve approach: $N(T_{\text{seek}} + T_{\text{transfer}} + T_{\text{rot}})$
 - Speedup: $N \cdot T_{\text{seek}} \sim O(N)$

86

86

Logical Disk Blocks

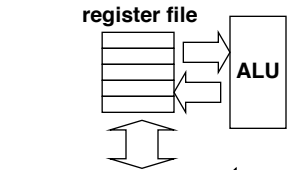
- Modern disks present a simpler abstract view of the complex sector geometry:
 - The set of available sectors is modeled as a sequence of b-sized **logical blocks** (0, 1, 2, ...)
- Mapping between logical blocks and actual (physical) sectors
 - Maintained by hardware/firmware device called *disk controller*.
 - 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**”.

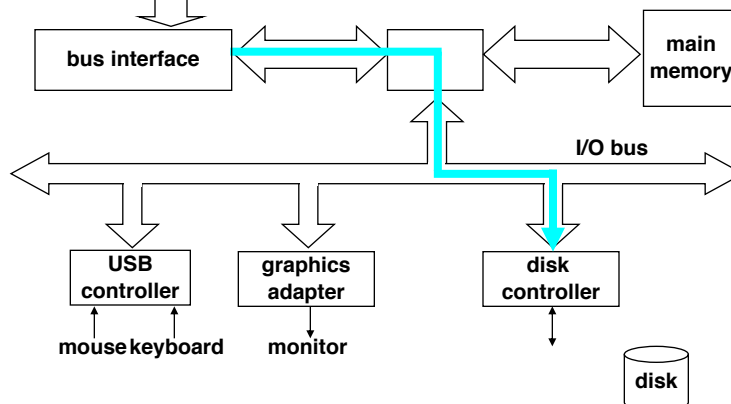
87

87

Reading a Disk Sector (1)

CPU chip

-  CPU initiates a disk read by writing a command, logical block number, and destination memory address to a **port** (address) associated with disk controller.

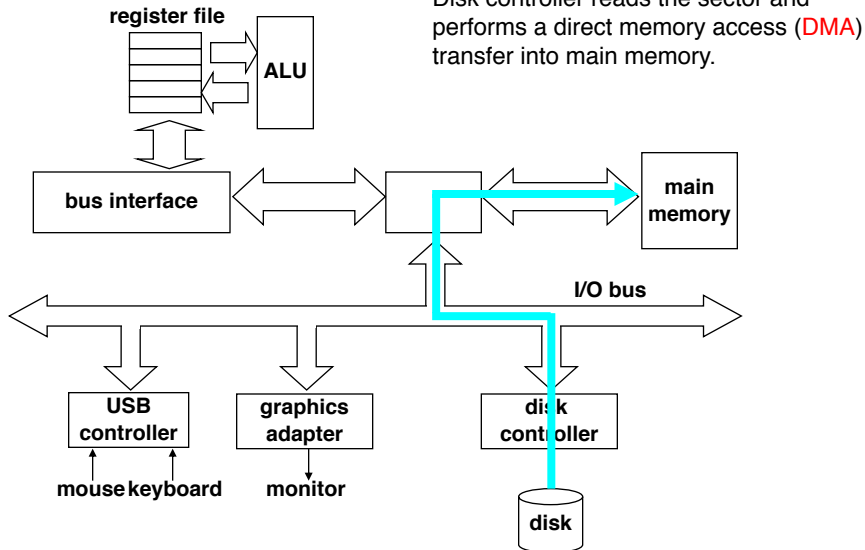


88

88

Reading a Disk Sector (2)

CPU chip

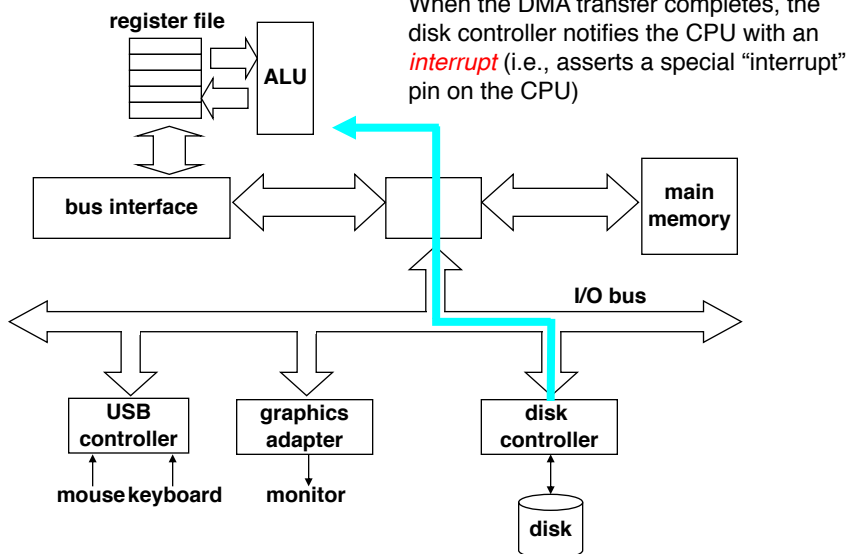


89

89

Reading a Disk Sector (3)

CPU chip



90

90

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??

91

91

What is multiprogramming and Why ?

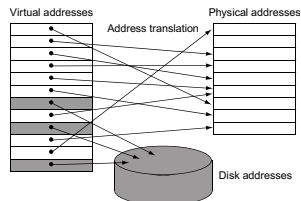
- Processor overlaps execution of two processes/programs
 - When one is waiting for I/O, the other is “swapped” in to the processor
 - Save the “state” of the process being swapped out
- Processes need to share memory
 - Each has its own address space
- Leads to better throughput and utilization

92

92

Virtual Memory

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

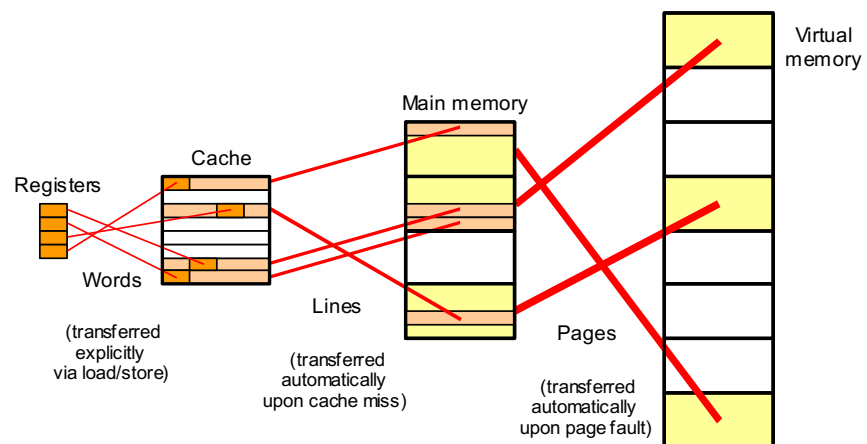


- Advantages:
 - illusion of having more physical memory
 - program relocation
 - protection

93

93

Memory Hierarchy: The Big Picture



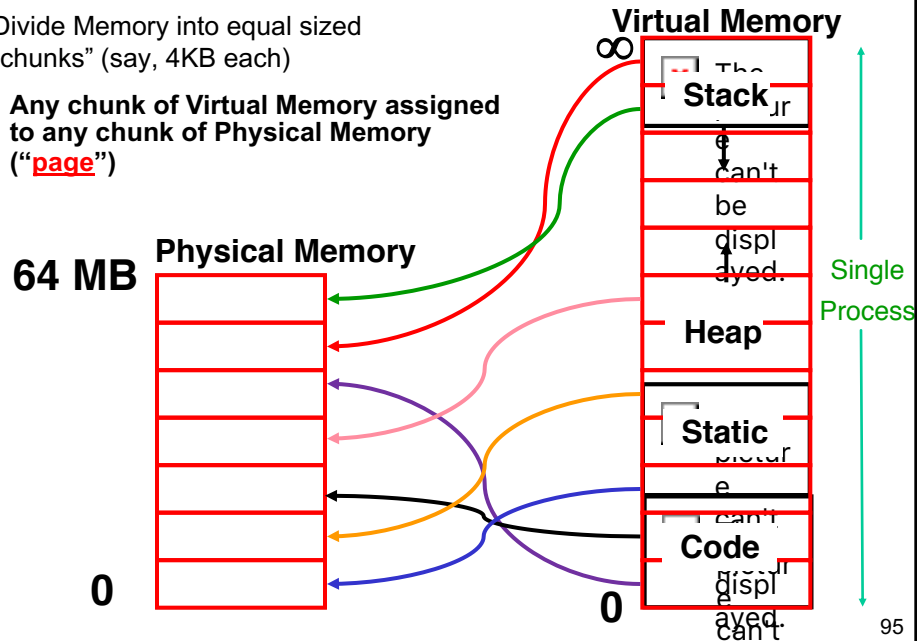
Data movement in a memory hierarchy.

94

94

Mapping Virtual Memory to Physical Memory

- Divide Memory into equal sized "chunks" (say, 4KB each)
- Any chunk of Virtual Memory assigned to any chunk of Physical Memory ("**page**")



95

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
- More on virtual memory....operating systems!

96