Lecture 22
Introduction to Memory Hierarchies

Let's go back to a course goal...

- **At the end of the semester, you should be able to...**

  – **...describe the fundamental components required in a single core of a modern microprocessor**

   - **(Also, explain how they interact with each other, with main memory, and with external storage media...)**

Question?

- **How much of a chip is "memory"?**
  – **10%**
  – **25%**
  – **50%**
  – **75%**
  – **85%**

Let's go back to a course goal...

- **At the end of the semester, you should be able to...**
  – **...describe the fundamental components required in a single core of a modern microprocessor**
    - **(Also, explain how they interact with each other, with main memory, and with external storage media...)**

Example
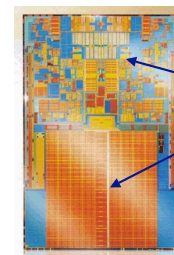
How do on-chip memory, processor logic, main memory, disk interact?

2.0 GB

$200.00

Apple Memory Module 2GB 667MHz DDR2 (PC2–5300) 2x1GB SO-DIMMs
Estimated Ship: Within 24 hours
Free Shipping

750GB SATA Hard Disk Drive Kit for...
Ships: Within 24hrs
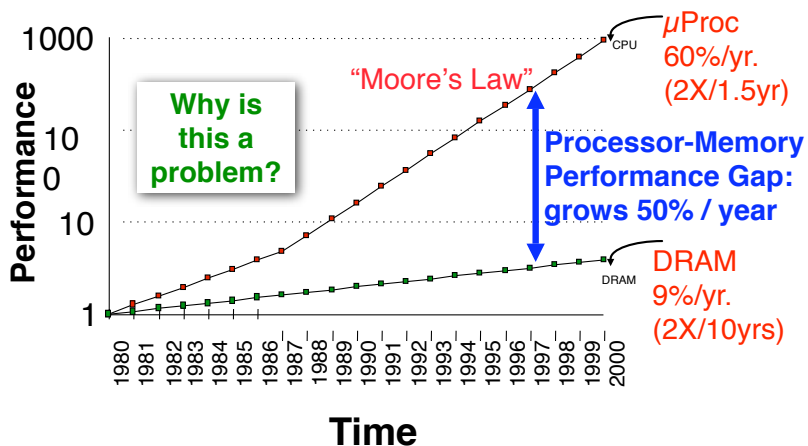**Free Shipping**
★★★★★
$299.00

Memory and Pipelining

- **In our 5 stage pipe, we've constantly been assuming that we can access our operand from memory in 1 clock cycle…**
  - We'd like this to be the case, and its is possible...but its also complicated
  - We'll discuss how this happens in the next several lectures
- **We'll talk about…**
  - **Memory Technology**
  - **Memory Hierarchy**
    - Caches
    - Memory
    - Virtual Memory

If I say "Memory" what do you think of?

- **Memory Comes in Many Flavors**
  - **SRAM (Static Random Access Memory)**
  - **DRAM (Dynamic Random Access Memory)**
  - **ROM, Flash, etc.**
  - **Disks, Tapes, etc.**
- **Difference in speed, price and "size"**
  - **Fast is small and/or expensive**
  - **Large is slow and/or expensive**
- **The search is on for a "universal memory"**
  - **What's a "universal memory"**
    - **Fast and non-volatile.**
  - **May be MRAM, PCRAM, etc. etc.**

Let's start with DRAM. Its generally the largest piece of RAM.

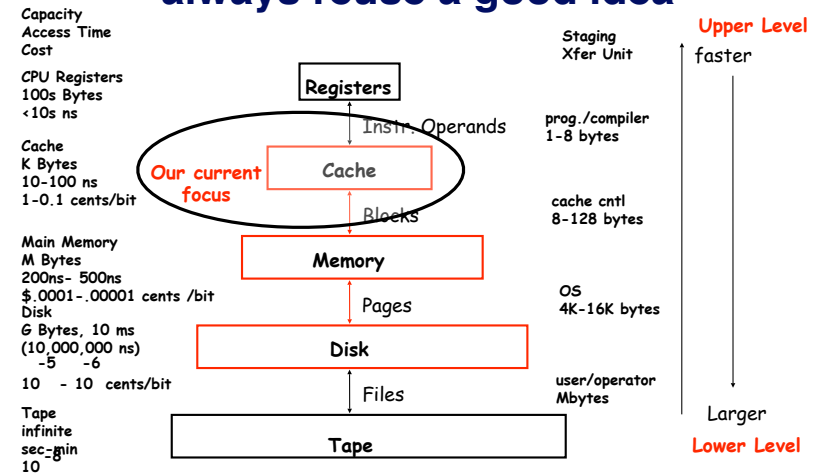# Is there a problem with DRAM?

Processor-DRAM Memory Gap (latency)

μProc 60%/yr. (2X/1.5yr)

"Moore's Law"

Why is this a problem?

Processor-Memory Performance Gap: grows 50% / year

DRAM 9%/yr. (2X/10yrs)

Performance

Time

More on DRAM

- **DRAM takes about 100 ns...**
- **What if clock rate for our 5 stage pipeline was 1 GHZ?**
  - **Would the memory *and* fetch stages stall for 100 cycles?**
    - **Actually, yes ... if only DRAM**
- **Caches come to the rescue**
  - **As transistors have gotten smaller, its allowed us to put more (faster) memory closer to the processing logic**
    - **The idea is to "mask" the latency of having to go off to main memory**

The principle of locality…

- …says that most programs don't access all code or data uniformly
  - i.e. in a loop, small subset of instructions might be executed over and over again…
  - …& a block of memory addresses might be accessed sequentially…

- This has lead to "memory hierarchies"
- Some important things to note:
  - Fast memory is expensive
  - Levels of memory usually smaller/faster than previous
  - Levels of memory usually "subset" one another
    - All the stuff in a higher level is in some level below it

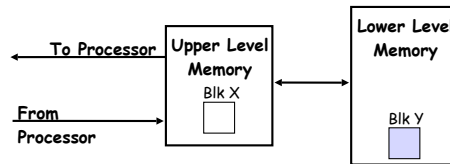# The Full Memory Hierarchy
"always reuse a good idea"

Capacity
Access Time
Cost

CPU Registers
100s Bytes
<10s ns

Cache
K Bytes
10-100 ns
1-0.1 cents/bit

Main Memory
M Bytes
200ns- 500ns
$.0001-.00001 cents /bit
Disk
G Bytes, 10 ms
(10,000,000 ns)
10^{-5} - 10^{-6} cents/bit

Tape
infinite
sec-min
10^{-8}

Registers

Our current focus

Cache

Memory

Disk

Tape

Instr. Operands

Blocks

Pages

Files

Staging
Xfer Unit

prog./compiler
1-8 bytes

cache cntl
8-128 bytes

OS
4K-16K bytes

user/operator
Mbytes

Upper Level
faster

Larger
Lower Level

Let's talk about caches more on the chalkboard

Accessing data in different levels of memory hierarchy:
A performance perspective

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 21264)**

```
To Processor  →          Upper Level          Lower Level
                           Memory               Memory

From          ←            Blk X                 Blk Y
Processor
```

Average Memory Access Time

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

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

Caches:
Memory between registers and DRAM

A brief description of a cache

- **Cache = next level of memory hierarchy up from register file**
  - **All values in register file should be in cache**

- **Cache entries usually referred to as "blocks"**
  - **Block is minimum amount of information that can be in cache**

- **If we're looking for item in a cache and find it, have a cache hit; it not a cache miss**

- **Cache miss rate = fraction of accesses not in the cache**

- **Miss penalty is # of clock cycles required b/c of the miss**

  Mem. stall cycles = Inst. count x Mem. ref./inst. x Miss rate x Miss penalty

Cache Basics

- **Fast (but small) memory close to processor**

- **When data referenced**
  - **If in cache, use cache instead of memory**
  - **If not in cache, bring into cache
    (actually, bring entire block of data, too)**
  - **Maybe have to kick something else out to do it!**

- **Important decisions**
  - **Placement: where in the cache can a block go**
  - **Identification: how do we find a block in cache**
  - **Replacement: what to kick out to make room in cache**
  - **Write policy: What do we do about writes**

Cache Basics

- **Cache consists of block-sized lines**
  - **Line size typically power of two**
  - **Typically 16 to 128 bytes in size**

- **Example**
  - **Suppose block size is 128 bytes**
    - **Lowest seven bits determine offset within block**
  - **Read data at address A=0x7fffa3f4**
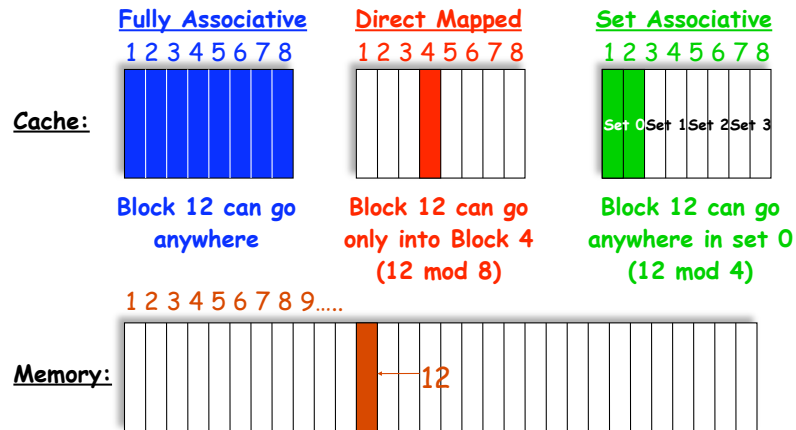 - **Address begins to block with base address 0x7fffa380**

Some initial questions to consider

- **Where can a block be placed in an upper level of
  memory hierarchy (i.e. a cache)?**

- **How is a block found in an upper level of memory
  hierarchy?**

- **Which cache block should be replaced on a cache miss
  if entire cache is full and we want to bring in new data?**

- **What happens if a you want to write back to a memory
  location?**
  - **Do you just write to the cache?**
  - **Do you write somewhere else?**

Where can a block be placed in a cache?

- **3 schemes for block placement in a cache:**
  - **Direct mapped cache:**
    - **Block (or data to be stored) can go to only 1 place in cache**
    - **Usually: (Block address) MOD (# of blocks in the cache)**

  - **Fully associative cache:**
    - **Block can be placed anywhere in cache**

  - **Set associative cache:**
    - **"Set" = a group of blocks in the cache**
    - **Block mapped onto a set & then block can be placed
      anywhere within that set**
    - **Usually: (Block address) MOD (# of sets in the cache)**
    - **If n blocks, we call it n-way set associative**

Where can a block be placed in a cache?

Fully Associative
1 2 3 4 5 6 7 8

Direct Mapped
1 2 3 4 5 6 7 8

Set Associative
1 2 3 4 5 6 7 8

Cache:

Set 0 Set 1 Set 2 Set 3

Block 12 can go anywhere

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

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

1 2 3 4 5 6 7 8 9…..

Memory:

12

Associativity

- If you have associativity > 1 you have to have a replacement policy
  - FIFO
  - LRU
  - Random

- "Full" or "Full-map" associativity means you check every tag in parallel and a memory block can go into any cache block
  - Virtual memory is effectively fully associative
  - (But don't worry about virtual memory yet)

How is a block found in the cache?

- **Cache's have address tag on each block frame that provides block address**
  - **Tag of every cache block that might have entry is examined against CPU address (in parallel! – why?)**

- **Each entry usually has a valid bit**
  - **Tells us if cache data is useful/not garbage**
  - **If bit is not set, there can't be a match…**

- **How does address provided to CPU relate to entry in cache?**
  - **Entry divided between block address & block offset…**
  - **…and further divided between tag field & index field**

How is a block found in the cache?

Block Address		Block Offset
Tag	Index	

- **Block offset field selects data from block**
  - **(i.e. address of desired data within block)**
- **Index field selects a specific set**
- **Tag field is compared against it for a hit**

- **Could we compare on more of address than the tag?**
  - **Not necessary; checking index is redundant**
    - **Used to select set to be checked**
    - **Ex.: Address stored in set 0 must have 0 in index field**
  - **Offset not necessary in comparison –entire block is present or not and all block offsets must match**

Which block should be replaced on a cache miss?

- If we look something up in cache and entry not there, generally want to get data from memory and put it in cache
  - B/c principle of locality says we'll probably use it again

- <u>Direct</u> <u>mapped</u> caches have 1 choice of what block to replace
- <u>Fully</u> <u>associative</u> or <u>set</u> <u>associative</u> offer more choices
- Usually 2 strategies:
  - Random – pick any possible block and replace it
  - LRU – stands for "Least Recently Used"
    - Why not throw out the block not used for the longest time
    - Usually approximated, not much better than random – i.e. 5.18% vs. 5.69% for 16KB 2-way set associative

---

Caches:
Replacement Policies

What happens on a write?

- FYI most accesses to a cache are reads:
  - Used to fetch instructions (reads)
  - Most instructions don't write to memory
    - For MIPS only about 7% of memory traffic involve writes
    - Translates to about 25% of cache data traffic

- Make common case fast! Optimize cache for reads!
  - Actually pretty easy to do…
  - Can read block while comparing/reading tag
  - Block read begins as soon as address available
  - If a hit, address just passed right on to CPU

- Writes take longer. Any idea why?

What happens on a write?

- Generically, there are 2 kinds of write policies:
  - <u>Write</u> <u>through</u> (or `store through`)
    - With write through, information written to block in cache *and* to block in lower-level memory
  - <u>Write</u> <u>back</u> (or `copy back`)
    - With write back, information written only to cache. It will be written back to lower-level memory when cache block is replaced

- The <u>dirty</u> <u>bit</u>:
  - Each cache entry usually has a bit that specifies if a write has occurred in that block or not…
  - Helps reduce frequency of writes to lower-level memory upon block replacement

What happens on a write?

- **Write back versus write through:**
  - <u>**Write back**</u> **advantageous because:**
    - **Writes occur at the speed of cache and don't incur delay of lower-level memory**
    - **Multiple writes to cache block result in only 1 lower-level memory access**
  - <u>**Write through**</u> **advantageous because:**
    - **Lower-levels of memory have most recent copy of data**

- **If CPU has to wait for a write, we have** <u>**write stall**</u>
  - **1 way around this is a** <u>**write buffer**</u>
  - **Ideally, CPU shouldn't have to stall during a write**
  - **Instead, data written to buffer which sends it to lower-levels of memory hierarchy**

What happens on a write?

- **What if we want to write and block we want to write to isn't in cache?**

- **There are 2 common policies:**
  - <u>**Write allocate:**</u> **(or fetch on write)**
    - **The block is loaded on a write miss**
    - **The idea behind this is that subsequent writes will be captured by the cache (ideal for a write back cache)**
  - <u>**No-write allocate:**</u> **(or write around)**
    - **Block modified in lower-level and not loaded into cache**
    - **Usually used for write-through caches**
      - (subsequent writes still have to go to memory)

Memory access equations

- **Using what we defined on previous slide, we can say:**
  - **Memory stall clock cycles =**
    - **Reads x Read miss rate x Read miss penalty +**
      Writes x Write miss rate x Write miss penalty

- **Often, reads and writes are combined/averaged:**
  - **Memory stall cycles =**
    - **Memory access x Miss rate x Miss penalty (approximation)**

- **Also possible to factor in instruction count to get a "complete" formula:**

$$CPU\ time = IC \times \left(CPI_{execution} + \frac{Memory\ stall\ clock\ cycles}{Instruction} \right) \times Clock\ cycle\ time$$

Reducing cache misses

- **Obviously, we want data accesses to result in cache hits, not misses –this will optimize performance**

- **Start by looking at ways to increase % of hits….**

- **…but first look at 3 kinds of misses!**
  - **Compulsory misses:**
    - **Very 1st access to cache block will not be a hit –the data's not there yet!**
  - **Capacity misses:**
    - **Cache is only so big. Won't be able to store every block accessed in a program – must swap out!**
  - **Conflict misses:**
    - **Result from set-associative or direct mapped caches**
    - **Blocks discarded/retrieved if too many map to a location**