Class Announcements

1. Collect your Midterm1 graded exams and Programming Assignment 1 grade sheet with feedback from me now if you have not already done so.

2. If you have not finished at least 2 bombs already, this is high time you put more effort into Programming Assignment 1.
Looking for Project Partners for P3?

One Student I know who has got permission for extended deadlines is looking for Project Partners. Let me know after class if you want to pair up.
Student Note Takers needed.

Students who are looking for an easy way to earn some extra money should read this email. The McBurney Center is recruiting a paid notetaker for your CS/ECE354 class. You'll receive a stipend of about $30 per credit for notes provided for the entire duration and scope of the class. No extra time outside of class is required, except for a short orientation for new notetakers. Detailed instructions will be on the Notetaker Information Form you'll get from the McBurney student as soon as you are hired.

If interested, make copies of sample notes from the last lecture and email or submit them to me as soon as possible. Make sure you include your name, phone number and email address with your sample notes. If your notes are selected, you will be contacted directly by the student who needs the notetaker.
Lecture Overview

1. Memory Hierarchy
2. Locality of Reference
3. Cache Organization
Smaller, faster, and costlier (per byte) storage devices

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

Remote secondary storage (distributed file systems, Web servers)

Local secondary storage (local disks)

Main memory (DRAM)

L2 cache (SRAM)

L1 cache (SRAM)

CPU registers hold words retrieved from cache memory.

L1 cache holds cache lines retrieved from the L2 cache.

L2 cache holds cache lines retrieved from L3 cache

L3 cache holds cache lines retrieved from memory.

Main memory holds disk blocks retrieved from local disks.

Local disks hold files retrieved from disks on remote network servers.
## Memory Hierarchy

<table>
<thead>
<tr>
<th>Name</th>
<th>Approx. Size</th>
<th>Approx. Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>&lt;1KB</td>
<td>&lt;1 nano secs</td>
</tr>
<tr>
<td>Cache</td>
<td>&lt;10MB</td>
<td>1ns-20 nano secs</td>
</tr>
<tr>
<td>DRAM</td>
<td>1-2GB per chip</td>
<td>50-100 nano secs</td>
</tr>
<tr>
<td>Local Flash Disks</td>
<td>8-16GB per chip</td>
<td>10-100 micro secs</td>
</tr>
<tr>
<td>Local Magnetic Disks</td>
<td>2-4TB</td>
<td>1-10 milli secs</td>
</tr>
<tr>
<td>Remote Storage Services</td>
<td>Several TBs</td>
<td>Depends on the network</td>
</tr>
</tbody>
</table>
# Memory Hierarchy Table from CSAPP Textbook

<table>
<thead>
<tr>
<th>Type</th>
<th>What cached</th>
<th>Where cached</th>
<th>Latency (cycles)</th>
<th>Managed by</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU registers</td>
<td>4-byte or 8-byte word</td>
<td>On-chip CPU registers</td>
<td>0</td>
<td>Compiler</td>
</tr>
<tr>
<td>TLB</td>
<td>Address translations</td>
<td>On-chip TLB</td>
<td>0</td>
<td>Hardware MMU</td>
</tr>
<tr>
<td>L1 cache</td>
<td>64-byte block</td>
<td>On-chip L1 cache</td>
<td>1</td>
<td>Hardware</td>
</tr>
<tr>
<td>L2 cache</td>
<td>64-byte block</td>
<td>On/off-chip L2 cache</td>
<td>10</td>
<td>Hardware</td>
</tr>
<tr>
<td>L3 cache</td>
<td>64-byte block</td>
<td>On/off-chip L3 cache</td>
<td>30</td>
<td>Hardware</td>
</tr>
<tr>
<td>Virtual memory</td>
<td>4-KB page</td>
<td>Main memory</td>
<td>100</td>
<td>Hardware + OS</td>
</tr>
<tr>
<td>Buffer cache</td>
<td>Parts of files</td>
<td>Main memory</td>
<td>100</td>
<td>OS</td>
</tr>
<tr>
<td>Disk cache</td>
<td>Disk sectors</td>
<td>Disk controller</td>
<td>100,000</td>
<td>Controller firmware</td>
</tr>
<tr>
<td>Network cache</td>
<td>Parts of files</td>
<td>Local disk</td>
<td>10,000,000</td>
<td>AFS/NFS client</td>
</tr>
<tr>
<td>Browser cache</td>
<td>Web pages</td>
<td>Local disk</td>
<td>10,000,000</td>
<td>Web browser</td>
</tr>
<tr>
<td>Web cache</td>
<td>Web pages</td>
<td>Remote server disks</td>
<td>1,000,000,000</td>
<td>Web proxy server</td>
</tr>
</tbody>
</table>
memory woes:

physically separate memory makes memory accesses SLOW!

P and M co-located? very expensive! or memory too small!
So, design the HW + SW to make this problem less bad.

Look at memory reference patterns. Design a special memory system.
The patterns come from the fetch + execute cycle:

1. fetch instruction
2. update PC
3. decode
4. get operands
5. do operation
6. put result(s) away

They exhibit locality.
temporal locality

Recently referenced memory locations are likely to referenced again (soon!)

loop: instr 1 @ A1
      instr 2 @ A2
      instr 3 @ A3
      jmp/ b loop @ A4

Instruction stream references:

A1 A2 A3 A4 A1 A2 A3 A4 A1 A2 A3 ...

Note that the same memory location is repeatedly read (for the fetch).
Spatial locality

Memory locations *near to* referenced locations are likely to also be referenced.

Code must do something to each element of the array.

Must load each element.
The *fetch* of the code exhibits a high degree of spatial locality.

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
</tr>
</thead>
</table>
| I2 is next to I1.

If these instructions are not branches, then we fetch

I1
I2
I3
etc.
Design a cache to attempt to hold copies of memory locations.

Which locations?

Put cache on chip for speed but it will hold fewer bytes than main memory.
P sends memory request to C.

- **hit**: requested location's copy *is* in the C
- **miss**: requested location's copy *is NOT* in the C. So, send the memory access to M.
Needed terminology:

\[
\text{miss ratio} = \frac{\text{# of misses}}{\text{total # of accesses}}
\]

\[
\text{hit ratio} = \frac{\text{# of hits}}{\text{total # of accesses}}
\]

or \[1 - \text{miss ratio}\]

You already assumed that \[\text{total # of accesses} = \text{# of misses} + \text{# of hits}\]
When a memory access causes a miss, place that location's bytes and its neighbors (spatial locality) into the cache. Keep the block of bytes there for as long as possible (temporal locality).

A statistic to measure how well this works:

\[ \text{Average Memory Access Time} = T_c + (\text{miss ratio})(T_m) \]
Quick example:

\[ T_c = 1 \text{ nsec} \]
\[ T_m = 20 \text{ nsec} \]

hit ratio is \( .98 \)

for measured program

\[ \text{AMAT} = 1 + (.02)(20) \]
\[ = 1.4 \text{ nsec} \]

Note: individual memory access takes either
1 nsec (hit)
or 21 nsec (miss).
Example Programs

```c
int sumvec(int v[N]){  
    int i, sum = 0;  
    for(i=0;i<N;i++){
        sum += v[i];
    }  
    return sum;
}
```

**Stride-k reference pattern:** accesses kth element of a contiguous array every time

Array Copy Example Program from 1st lecture