Announcements

Exam on October 10th in class
Can we move the exam to 3:00pm-5:00pm?
Monday will do a review of material
Simple Main Memory

Consider these parameters:
- 1 cycle to send address
- 6 cycles to access each word
- 1 cycle to send word back

Miss penalty for a 4-word block
- \((1 + 6 + 1) \times 4 = 32\)

How can we speed this up?

Wider Main Memory

Make memory wider:
- read out all words in parallel

Miss penalty for 4-word block:
- \(1 + 6 + 1 = 8\)

Cost
- wider bus
- larger expansion size
- error-correction is harder
Interleaved Main Memory

Instead of larger width,

Break memory into M banks so word A is in:

- $A \mod M$ at $A \div M$

![Diagram showing memory banks]

Doubleword in bank  |  Bank  |  Word in doubleword

Interleaved Main Memory (Cont.)

Use parallelism in memory banks

Give address to all

Get data out one at a time

Excellent for caches:

- sequential word access
- word/doubleword matches cache & bus width

Assume:

- Access time $A=2$
- Cycle time $A+B=4$
- Transfer time $T=1$
Interleaving

All banks see addresses
Unit-stride (sequential) access

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Address</th>
<th>Bank 0</th>
<th>Bank 1</th>
<th>Bank 2</th>
<th>Bank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>3</td>
<td>b/t</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>b/t</td>
<td>b</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>a</td>
<td>a</td>
<td>t/a</td>
<td>a</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>t/a</td>
</tr>
<tr>
<td>7</td>
<td>b/t</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>b</td>
<td>b/t</td>
<td>b</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Independent Memory Banks

Can access different parts of banks
Unit stride example

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Address</th>
<th>Bank 0</th>
<th>Bank 1</th>
<th>Bank 2</th>
<th>Bank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>b/t</td>
<td>a</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>b</td>
<td>b/t</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>a</td>
<td>b</td>
<td>b/t</td>
<td>a</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b/t</td>
</tr>
<tr>
<td>7</td>
<td>18</td>
<td>b/t</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
<td>b</td>
<td>b/t</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>b</td>
<td></td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>
### Independent Banks (Stride of 3)

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Address</th>
<th>Bank 0</th>
<th>Bank 1</th>
<th>Bank 2</th>
<th>Bank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>b/t</td>
<td>a</td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b/t</td>
</tr>
<tr>
<td>5</td>
<td>24</td>
<td>a</td>
<td>a</td>
<td>b/t</td>
<td>b</td>
</tr>
<tr>
<td>6</td>
<td>27</td>
<td>a</td>
<td>b/t</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>7</td>
<td>30</td>
<td>b/t</td>
<td>b</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>8</td>
<td>33</td>
<td>b</td>
<td>a</td>
<td>b/t</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b/t</td>
</tr>
</tbody>
</table>

### Independent Banks (Stride of 2)

Bank conflict!

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Address</th>
<th>Bank 0</th>
<th>Bank 1</th>
<th>Bank 2</th>
<th>Bank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>b/t</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>b</td>
<td>b/t</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>a</td>
<td></td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td>a</td>
<td></td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>b/t</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>b</td>
<td>b/t</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b</td>
</tr>
</tbody>
</table>
Interleaving Conclusions

Interleaving for sequential accesses:
- load cache words
- good for write-back caches
Independent banking otherwise
Do both
- banks: interleaving for high bandwidth
- superbanks: multiple cache misses
  - non-blocking caches and/or multiprocessors
How many banks?

Processor/Memory Bandwidth Balance

DRAM bandwidth has not kept up with processor performance
- DRAM improves 7%/year
- processor improves 40%-50%/year
E.g., balancing bandwidths
- processor bandwidth requirement
  - 4ns clock
  - no data cache
  - 1 64-bit ld/st per clock
- minimum memory supply
  - 120ns DRAM => at least 32 banks
  - 16Mb x 1 DRAM => 2048 DRAMs (4 Gbyte)!
  - 4Mb x 4 DRAM => 512 DRAMs (1 Gbyte)!
Bandwidth Balance (Example)

Let us add a cache
- 5% miss rate
- 4 words per block
- write-back 25% dirty lines
- => approx. 1 word per 4 processor cycles (cut by 4)

Minimum memory still large:
- 16Mb x 1 DRAM => 512 DRAMs (1 Gbyte)
- 4Mb x 4 DRAM => 128 DRAMs (256 Mbyte)

Solution

Additional cache levels:
- add SRAM cost
Make memory wider
Make better use of very large DRAM chip bandwidth
Special DRAMs

DRAM-specific organization

Nibble-mode:
- extra bits from sequential locations with one RAS

Page-mode:
- an SRAM-like access to row buffer (like a cache)

Static-column:
- like page-mode without strobing CAS

Special DRAMs (Cont.)

RAMBus:
- becoming more commonplace in high-end
- e.g., Alpha 21364
- RAS/CAS bottleneck => eliminate the interface
- packet-switched bus to each DRAM
- treat DRAM as a memory system rather component
- each DRAM can return variable amount of data!
Special DRAMs (Cont.)

Embedded DRAM:
- logic in DRAM technology
- huge on-chip DRAM bandwidth => compute in DRAM
- used as graphics chips
- can this be used in general-purpose computing?
- what are the implementation problems?

Virtual Memory

Original motivation:
- make small memory look large
- avoid overlays => permit common software on wide product line
- use main memory as a disk cache

Current motivation:
- relocation, protection, fast start-up, sharing, sparse use
- memory mapped files, network communication

Engineered different from CPU caches:
- miss access time >> miss transfer time
Virtual Memory (Cont.)

Memory page (placeholders are called page frames)
- typically 4K-16K
- fixed-size per system

Architecture presents programs with a simple view
- memory addressed with 32-bit addresses
- load 0x10000028, R1 => 0x10000028 is the virtual address (VA)
- system maps VA to physical address (PA)
- e.g., 0x10000028 maps to 0x0000F028 (page 15 for 4K page)

Someone else and I run “netscape”
- load 0x10000028, R1
- VA must map to different PA

Thus, VA allows
- use more physical memory (unlikely)
- think it is the only program running in memory
- think it always start at address 0x0
- be protected from rogue programs (trusting OS)
- start running when most of the program is on disk
Virtual Memory (Cont.)

A VA miss is called a page fault
- VM hardware detects an exception
- an asynchronous trap (“like an automatic trap instruction”)
- OS grains control and initiates disk access
- OS usually runs someone else in the meantime
- interrupt when disk access is complete (“another exception”)
- original instruction restarts

Unlike cache misses
- OS is used to handle page faults. Why?

Virtual Memory (Cont.)

Page placement
- OS
- fully associative => why?

Page identification
- address translation - virtual to physical
- indirection through page tables
- translation cached in translation buffer (TLB)

Page replacement
- OS policy
- approximate LRU
- maintain “working set”

Write strategy: write-back (dirty bits)
Virtual Memory Architecture

Per-process address space common
- each process is given a virtual “address space” when created
- gone when process dies - physical pages reclaimed

System-wide shared virtual address space
- a.k.a. single address-space OS
- can persist over system lifetime
- requires large virtual address spaces
- used in many recent systems: IBM PowerPC, HP

System-Wide VM: IBM PowerPC

Uses both segmentation & paging:
- segments to reduce page table size
- pages to provide protection
- 32-bit effective address
- 16 segment registers per process
- combined to 52-bit virtual address
- protection via OS loading of segment registers
- 16M segments in system
IBM PowerPC (Cont.)

Segment Registers

Page table entries from memory

TLB

Per Process VM

Page Table status:
- protection
- valid
- reference
Per Process VM (Cont.)

Logical path
- multiple memory operations
- often multiple levels of page tables
- too slow!

Use a cache of page table entries (PTEs)
- Translation Lookaside Buffer (TLB)
- Typically 8 to 1024 entries
- Can be fully-associative

Translation Lookaside Buffer (TLB)

<table>
<thead>
<tr>
<th>VPN state</th>
<th>PTE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Compare & Select

PPN

Hit/Miss
TLB Miss & Reach (Cont.)

TLB designed w.r.t. caches:
- TLB Reach: the total cache space accessible through TLB

Miss handling:
- hardware table walk: e.g., Sun SPARC
- software table walk: e.g., MIPS