“The severity of the decline in the market is further evidence that the ‘post-PC era’ heralded several years ago by Steven P. Jobs, Apple’s former chief executive, was not an empty slogan. Jobs … predicted that PCs would endure, but that smartphones and tablets would become the devices people favored for most of their computing needs”

How many hours $h$ on last project?

a) $0 \leq h < 8$

b) $8 \leq h < 16$

c) $16 \leq h < 32$

d) $32 \leq h < 64$

e) $64 \leq h$
I understand **Virtual Memory**.

a) Strongly disagree  
b) Mildly disagree  
c) Neutral  
d) Mildly agree  
e) Strongly agree
Review

- Manage memory to disk? Treat as cache
  - Included protection as bonus, now critical
  - Use Page Table of mappings for each user vs. tag/data in cache
  - TLB is cache of Virtual ⇒ Physical addr trans

- Virtual Memory allows protected sharing of memory between processes

- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well
## Review: Address Mapping: Page Table

### Virtual Address:

- **page no.**
- **offset**

### Page Table

- **Page Table Base Reg**
- **index into page table**

### Page Table located in physical memory

<table>
<thead>
<tr>
<th>V</th>
<th>A.R.</th>
<th>P. P. A.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Val-id</td>
<td>Access Rights</td>
<td>Physical Page Address</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Physical Memory Address**
Fetching data on a memory read

- **Check TLB** (input: VPN, output: PPN)
  - hit: fetch translation
  - miss: check page table (in memory)
    - Page table hit: fetch translation
    - Page table miss: page fault, fetch page from disk to memory, return translation to TLB

- **Check cache** (input: PPN, output: data)
  - hit: return value
  - miss: fetch value from memory, remember it in cache, return value
Address Translation using TLB

Virtual Address

VPN

TLB Tag INDEX Offset

TLB

P. P. N.

Physical Page Number

Tag used just like in cache

Data Cache

Tag Data

Tag Data

PPN Offset

Physical Address

Tag INDEX Offset
### Typical TLB Format

<table>
<thead>
<tr>
<th>Tag</th>
<th>Physical Page #</th>
<th>Dirty</th>
<th>Ref</th>
<th>Valid</th>
<th>Access Rights</th>
</tr>
</thead>
</table>

- TLB just a cache on the page table mappings
- TLB access time comparable to cache (much less than main memory access time)
- **Dirty**: since use write back, need to know whether or not to write page to disk when replaced
- **Ref**: Used to help calculate LRU on replacement
  - Cleared by OS periodically, then checked to see if page was referenced
What if not in TLB?

- Option 1: Hardware checks page table and loads new Page Table Entry into TLB
- Option 2: Hardware *traps* to OS, up to OS to decide what to do
  - MIPS follows Option 2: Hardware knows nothing about page table
  - A trap is a synchronous exception in a user process, often resulting in the OS taking over and performing some action before returning to the program.
  - More about exceptions next lecture
What if the data is on disk?

- We load the page off the disk into a free block of memory, using a **DMA transfer** (Direct Memory Access – special hardware support to avoid processor)
  - Meantime we switch to some other process waiting to be run

- When the DMA is complete, we get an interrupt and update the process's page table
  - So when we switch back to the task, the desired data will be in memory
What if we don’t have enough memory?

- We chose some other page belonging to a program and transfer it onto the disk if dirty
  - If clean (disk copy is up-to-date), just overwrite that data in memory
  - We chose the page to evict based on replacement policy (e.g., LRU)
- And update that program's page table to reflect the fact that its memory moved somewhere else
- If continuously swap between disk and memory, called Thrashing
Question (1/3)

- 40-bit virtual address, 16 KB page
  - Virtual Page Number (? bits)  Page Offset (? bits)

- 36-bit physical address
  - Physical Page Number (? bits)  Page Offset (? bits)

- Number of bits in Virtual Page Number/Page offset, Physical Page Number/Page offset?
  - a: 22/18 (VPN/PO), 22/14 (PPN/PO)
  - b: 24/16, 20/16
  - c: 26/14, 22/14
  - d: 26/14, 26/10
  - e: 28/12, 24/12
(1/3) Answer

- 40-bit virtual address, 16 KB page
  - Virtual Page Number (26 bits) | Page Offset (14 bits)

- 36-bit physical address
  - Physical Page Number (22 bits) | Page Offset (14 bits)

- Number of bits in
  - Virtual Page Number/Page offset, Physical Page Number/Page offset?
    - a: 22/18 (VPN/PO), 22/14 (PPN/PO)
    - b: 24/16, 20/16
    - c: 26/14, 22/14
    - d: 26/14, 26/10
    - e: 28/12, 24/12
Question (2/3): 40b VA, 36b PA

- 2-way set-assoc. TLB, 512 entries:
  - TLB Tag (\?) bits
  - TLB Index (\?) bits
  - Page Offset (14 bits)

- TLB Entry: Valid bit, Dirty bit, Access Control (say 2 bits), Physical Page Number

- Number of bits in TLB Tag / Index / Entry?
  - a: 12 / 14 / 38 (TLB Tag / Index / Entry)
  - b: 14 / 12 / 40
  - c: 18 / 8 / 44
  - d: 17 / 9 / 43
  - e: 18 / 8 / 58
(2/3) Answer

- 2-way set-assoc data cache, 256 (28) “sets”, 2 TLB entries per set → 8 bit index

<table>
<thead>
<tr>
<th>TLB Tag (18 bits)</th>
<th>TLB Index (8 bits)</th>
<th>Page Offset (14 bits)</th>
</tr>
</thead>
</table>

Virtual Page Number (26 bits)

- TLB Entry: Valid bit, Dirty bit, Access Control (2 bits), Virtual Page Number, Physical Page Number

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>Access (2 bits)</th>
<th>TLB Tag (18 bits)</th>
<th>Physical Page No. (22 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a: 12 / 14 / 38 (TLB Tag / Index / Entry)
b: 14 / 12 / 40
c: 18 / 8 / 44
d: 17 / 9 / 43
e: 18 / 8 / 58
Question (3/3)

- 2-way set-assoc, 64KB data cache, 64B block

```
Cache Tag (? bits)  Cache Index (? bits)  Block Offset (? bits)
```

Physical Address (36 bits)

- Data Cache Entry: Valid bit, Dirty bit, Cache tag + ? bits of Data

```
V  D  Cache Tag (? bits)  Cache Data (? bits)
```

- Number of bits in Data cache Tag / Index / Offset / Entry?

a: 12 / 9 / 14 / 87 (Tag/Index/Offset/Entry)
b: 20 / 10 / 6 / 86
c: 20 / 10 / 6 / 534
d: 21 / 9 / 6 / 87
e: 21 / 9 / 6 / 535
(3/3) Answer

- 2-way set-associative data cache, 64K/1K (2^{10}) “sets”, 2 entries per set => 9 bit index

- Data Cache Entry: Valid bit, Dirty bit, Cache tag + 64 Bytes of Data

<table>
<thead>
<tr>
<th></th>
<th>V</th>
<th>D</th>
<th>Cache Tag (21 bits)</th>
<th>Cache Data (64 Bytes = 512 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>12</td>
<td>9</td>
<td>14 / 87 (Tag/Index/Offset/Entry)</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>20</td>
<td>10</td>
<td>6 / 86</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>20</td>
<td>10</td>
<td>6 / 534</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>21</td>
<td>9</td>
<td>6 / 87</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>21</td>
<td>9</td>
<td>6 / 535</td>
<td></td>
</tr>
</tbody>
</table>
Virtual Memory Summary

- **User program view:**
  - Contiguous
  - Start from some set address
  - Infinitely large
  - Is the only running program

- **Reality:**
  - Non-contiguous
  - Start wherever available memory is
  - Finite size
  - Many programs running at a time

- **Virtual memory provides:**
  - Illusion of contiguous memory
  - All programs starting at same set address
  - Illusion of ~ infinite memory (232 or 264 bytes)
  - Protection, Sharing

- **Implementation:**
  - Divide memory into chunks (pages)
  - OS controls page table that maps virtual into physical addresses
  - Memory as a cache for disk
  - TLB is a cache for the page table
Bonus slides

- These are extra slides that used to be included in lecture notes, but have been moved to this, the “bonus” area to serve as a supplement.

- The slides will appear in the order they would have in the normal presentation.
4 Qs for any Memory Hierarchy

- **Q1: Where can a block be placed?**
  - One place (direct mapped)
  - A few places (set associative)
  - Any place (fully associative)

- **Q2: How is a block found?**
  - Indexing (as in a direct-mapped cache)
  - Limited search (as in a set-associative cache)
  - Full search (as in a fully associative cache)
  - Separate lookup table (as in a page table)

- **Q3: Which block is replaced on a miss?**
  - Least recently used (LRU)
  - Random

- **Q4: How are writes handled?**
  - Write through (Level never inconsistent w/lower)
  - Write back (Could be “dirty”, must have dirty bit)
Q1: Where block placed in upper level?

- Block #12 placed in 8 block cache:
  - Fully associative
  - Direct mapped
  - 2-way set associative
    - Set Associative Mapping = Block # Mod # of Sets

Fully associative: block 12 can go anywhere

Direct mapped: block 12 can go only into block 4 (12 mod 8)

Set associative: block 12 can go anywhere in set 0 (12 mod 4)
Q2: How is a block found in upper level?

- Direct indexing (using index and block offset), tag compares, or combination
- Increasing associativity shrinks index, expands tag
Q3: Which block replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

<table>
<thead>
<tr>
<th>Miss Rates</th>
<th>Associativity: 2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LRU</td>
<td>Ran</td>
<td>LRU</td>
</tr>
<tr>
<td>Size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 KB</td>
<td>5.2%</td>
<td>5.7%</td>
<td>4.7%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%</td>
<td>2.0%</td>
<td>1.5%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</td>
</tr>
</tbody>
</table>
Q4: What to do on a write hit?

- **Write-through**
  - update the word in cache block and corresponding word in memory

- **Write-back**
  - update word in cache block
  - allow memory word to be “stale”
  - => add ‘dirty’ bit to each line indicating that memory be updated when block is replaced
  - => OS flushes cache before I/O !!!

- **Performance trade-offs?**
  - WT: read misses cannot result in writes
  - WB: no writes of repeated writes