Main Memory

- Main memory generally utilizes Dynamic RAM (DRAM), which use a single transistor to store a bit, but require a periodic data refresh by reading every row (~every 8 msec).
- Static RAM may be used if the added expense, low density, power consumption, and complexity is feasible (e.g. Cray Vector Supercomputers).
- Main memory performance is affected by:
  - **Memory latency**: Affects cache miss penalty. Measured by:
    - **Access time**: The time it takes between a memory access request is issued to main memory and the time the requested information is available to cache/CPU.
    - **Cycle time**: The minimum time between requests to memory (greater than access time in DRAM to allow address lines to be stable)
  - **Memory bandwidth**: The sustained data transfer rate between main memory and cache/CPU.
Classic DRAM Organization

- bit (data) lines
- Each intersection represents a 1-T DRAM Cell
- word (row) select

Row and Column Address together:
- Select 1 bit at a time
Logical Diagram of A Typical DRAM

- Control Signals (RAS_L, CAS_L, WE_L, OE_L) are all active low
- Din and Dout are combined (D):
  - WE_L is asserted (Low), OE_L is disasserted (High)
    - D serves as the data input pin
  - WE_L is disasserted (High), OE_L is asserted (Low)
    - D is the data output pin
- Row and column addresses share the same pins (A)
  - RAS_L goes low: Pins A are latched in as row address
  - CAS_L goes low: Pins A are latched in as column address
Four Key DRAM Timing Parameters

• $t_{\text{RAC}}$: Minimum time from RAS (Row Access Strobe) line falling to the valid data output.
  – Usually quoted as the nominal speed of a DRAM chip
  – For a typical 4Mb DRAM $t_{\text{RAC}} = 60$ ns

• $t_{\text{RC}}$: Minimum time from the start of one row access to the start of the next.
  – $t_{\text{RC}} = 110$ ns for a 4Mbit DRAM with a $t_{\text{RAC}}$ of 60 ns

• $t_{\text{CAC}}$: Minimum time from CAS (Column Access Strobe) line falling to valid data output.
  – 15 ns for a 4Mbit DRAM with a $t_{\text{RAC}}$ of 60 ns

• $t_{\text{PC}}$: Minimum time from the start of one column access to the start of the next.
  – About 35 ns for a 4Mbit DRAM with a $t_{\text{RAC}}$ of 60 ns
DRAM Performance

• A 60 ns ($t_{RAC}$) DRAM chip can:
  – Perform a row access only every 110 ns ($t_{RC}$)
  – Perform column access ($t_{CAC}$) in 15 ns, but time between column accesses is at least 35 ns ($t_{PC}$).

• In practice, external address delays and turning around buses make it 40 to 50 ns

• These times do not include the time to drive the addresses off the CPU or the memory controller overhead.
DRAM Write Timing

- Every DRAM access begins at:
  - The assertion of the RAS_L
  - 2 ways to write: early or late v. CAS

- Early Wr Cycle: WE_L asserted before CAS_L
- Late Wr Cycle: WE_L asserted after CAS_L
DRAM Read Timing

° Every DRAM access begins at:
  • The assertion of the RAS_L
  • 2 ways to read: early or late v. CAS

DRAM Read Cycle Time

RAS_L

CAS_L

A: Row Address Col Address Junk X: Row Address Col Address Junk

WE_L

OE_L

D: High Z Junk Data Out X: High Z Data Out

Read Access Time

Early Read Cycle: OE_L asserted before CAS_L

Late Read Cycle: OE_L asserted after CAS_L
Simplified Asynchronous DRAM Read Timing

Source: http://arstechnica.com/paedia/r/ram_guide/ram_guide.part2-1.html
Page Mode DRAM: Motivation

- **Regular DRAM Organization:**
  - N rows x N column x M-bit
  - Read & Write M-bit at a time
  - Each M-bit access requires a RAS / CAS cycle

- **Fast Page Mode DRAM**
  - N x M “register” to save a row
Fast Page Mode DRAM: Operation

- **Fast Page Mode DRAM**
  - \( N \times M \) "SRAM" to save a row

- **After a row is read into the register**
  - Only CAS is needed to access other M-bit blocks on that row
  - RAS_L remains asserted while CAS_L is toggled

---

**Diagram:**

- Column Address
- N cols
- N rows
- DRAM
- M bits
- N x M "SRAM"

**Timelines:**

- 1st M-bit Access
- 2nd M-bit
- 3rd M-bit
- 4th M-bit

**Waveforms:**

- RAS_L
- CAS_L
- A
- Row Address
- Col Address
- Col Address
- Col Address
- Col Address
Simplified Asynchronous Fast Page Mode (FPM) DRAM Read Timing

Typical timing at 66 MHZ : 5-3-3-3
For bus width = 64 bits = 8 bytes cache block size = 32 bytes
It takes = 5+3+3+3 = 14 memory cycles or 15 ns x 14 = 210 ns to read 32 byte block
Read Miss penalty for CPU running at 1 GHZ = 15 x 14 = 210 CPU cycles

Source: http://arstechnica.com/paedia/r/ram_guide/ram_guide.part2-1.html
Simplified Asynchronous Extended Data Out (EDO) DRAM Read Timing

- Extended Data Out DRAM operates in a similar fashion to Fast Page Mode DRAM except the data from one read is on the output pins at the same time the column address for the next read is being latched in.

**EDO Read**

**EDO DRAM speed rated using tRAC ~ 40-60ns**

![EDO DRAM Read Timing Diagram](http://arstechnica.com/paedia/r/ram_guide/ram_guide.part2-1.html)

Typical timing at 66 MHZ: 5-2-2-2

For bus width = 64 bits = 8 bytes Max. Bandwidth = 8 x 66 / 2 = 264 Mbytes/sec

It takes = 5+2+2+2 = 11 memory cycles or 15 ns x 11 = 165 ns to read 32 byte cache block

Read Miss penalty for CPU running at 1 GHZ = 11 x 15 = 165 CPU cycles

Source: http://arstechnica.com/paedia/r/ram_guide/ram_guide.part2-1.html
Synchronous Dynamic RAM (SDRAM) Organization

SDRAM speed is rated at max. clock speed supported:
66MHz = PC66
100MHz = PC100
133MHz = PC133
150MHz = PC150
Typical timing at 133 MHz (PC133 SDRAM) : 4-1-1-1
For bus width = 64 bits = 8 bytes      Max. Bandwidth = 133 x 8 = 1064 Mbytes/sec
It takes = 4+1+1+1 = 8 memory cycles or 7.5 ns x 8 = 60 ns to read 32 byte cache block
Read Miss penalty for CPU running at 1 GHz = 7.5 x 8 = 60 CPU cycles

Memory Bandwidth Improvement Techniques

• **Wider Main Memory:**
  Memory width is increased to a number of words (usually the size of a second level cache block).
  ⇒ Memory bandwidth is proportional to memory width.
    
    e.g. Doubling the width of cache and memory doubles memory bandwidth

• **Simple Interleaved Memory:**
  Memory is organized as a number of banks each one word wide.
  – Simultaneous multiple word memory reads or writes are accomplished by sending memory addresses to several memory banks at once.
  – Interleaving factor: Refers to the mapping of memory addresses to memory banks.
    
    e.g. using 4 banks, bank 0 has all words whose address is:
    
    (word address) (mod) 4 = 0
Three examples of bus width, memory width, and memory interleaving to achieve higher memory bandwidth

Simplest design:
Everything is the width of one word

Wider memory, bus and cache

Narrow bus and cache with interleaved memory
Memory Bank Interleaving

Access Pattern without Interleaving:

D1 available
Start Access for D1
Start Access for D2

Access Pattern with 4-way Interleaving:

Access Bank 0
Access Bank 1
Access Bank 2
Access Bank 3

We can Access Bank 0 again
### Four way interleaved memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Bank 0</th>
<th>Address</th>
<th>Bank 1</th>
<th>Address</th>
<th>Bank 2</th>
<th>Address</th>
<th>Bank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>5</td>
<td></td>
<td>6</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>9</td>
<td></td>
<td>10</td>
<td></td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>13</td>
<td></td>
<td>14</td>
<td></td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

### Three memory banks address interleaving:
- Sequentially interleaved addresses on the left, address requires a division
- Right: Alternate interleaving requires only modulo to a power of 2

<table>
<thead>
<tr>
<th>Address within bank</th>
<th>Memory bank</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequentially interleaved</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
</tr>
</tbody>
</table>
Miss Rate Vs. Cache Block Size

Increasing the cache block size tends to decrease the miss rate due to increased use of spatial locality:

<table>
<thead>
<tr>
<th>Block Size (bytes)</th>
<th>Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0%</td>
</tr>
<tr>
<td>32</td>
<td>0%</td>
</tr>
<tr>
<td>64</td>
<td>0%</td>
</tr>
<tr>
<td>128</td>
<td>0%</td>
</tr>
<tr>
<td>256</td>
<td>0%</td>
</tr>
</tbody>
</table>

Graph showing miss rate vs. cache block size for different block sizes (1K, 4K, 16K, 64K, 256K).
Memory Width, Interleaving: An Example

Given a base system with following parameters:

- Cache Block size = 1 word,
- Memory bus width = 1 word,
- Miss rate = 3%
- Miss penalty = 32 cycles, broken down as follows:
  - 4 cycles to send address,
  - 24 cycles access time/word,
  - 4 cycles to send a word
- Memory access/instruction = 1.2
- Ideal execution CPI (ignoring cache misses) = 2
- Miss rate (block size=2 word) = 2%
- Miss rate (block size=4 words) = 1%

- The CPI of the base machine with 1-word blocks = \(2 + (1.2 \times 0.03 \times 32) = 3.15\)
- Increasing the block size to two words gives the following CPI:
  - 32-bit bus and memory, no interleaving = \(2 + (1.2 \times 0.02 \times 2 \times 32) = 3.54\)
  - 32-bit bus and memory, interleaved = \(2 + (1.2 \times 0.02 \times (4 + 24 + 8)) = 2.86\)
  - 64-bit bus and memory, no interleaving = \(2 + (1.2 \times 0.02 \times 1 \times 32) = 2.77\)

- Increasing the block size to four words; resulting CPI:
  - 32-bit bus and memory, no interleaving = \(2 + (1.2 \times 1\% \times 4 \times 32) = 3.54\)
  - 32-bit bus and memory, interleaved = \(2 + (1.2 \times 1\% \times (4 + 24 + 16)) = 2.53\)
  - 64-bit bus and memory, no interleaving = \(2 + (1.2 \times 2\% \times 2 \times 32) = 2.77\)
Computer System Components

**SDRAM**
- PC100/PC133
- 100-133MHz
- 64-128 bits wide
- 2-way interleaved
- ~ 900 MBYTES/SEC

**Double Date Rate (DDR) SDRAM**
- PC2100
- 266MHz (effective 133x2)
- 64-128 bits wide
- 4-way interleaved
- ~2.1 GBYTES/SEC (second half 2000)

**RAMbus DRAM (RDRAM)**
- 400-800MHz
- 16 bits wide channel
- ~ 1.6 GBYTES/SEC (per channel)

**CPU**
- 600MHZ - 1.7GHZ (a multiple of system bus speed)
- Pipelined (7-21 stages)
- Superscalar (max ~ 4 instructions/cycle)

**Caches**
- Examples: Alpha, AMD K7: EV6, 200MHz
  - Intel PII, PIII: GTL+ 100MHz
  - Intel P4 400MHz

**System Bus**

**Memory**

**Memory Controller**

**Memory Bus**

**Controllers**

**Controllers**

**I/O Buses**
- Example: PCI, 33MHz
  - 32 bits wide
  - 133 MBYTES/SEC

**I/O Devices:**
- Disks
- Displays
- Keyboards

**Networks**

**NICs**

**Examples:**
- PCI, 33MHz
  - 32 bits wide
  - 133 MBYTES/SEC
X86 CPU Cache/Memory Performance Example

AMD Athlon T-Bird 1GHz
L1: 64K INST, 64K DATA (3 cycle latency), both 2-way
L2: 256K 16-way 64 bit
  Latency: 7 cycles
  L1,L2 on-chip

Memory:
PC2100
133MHz DDR SDRAM 64bit
Peak bandwidth: 2100 MB/s

PC133
133MHz SDRAM 64bit
Peak bandwidth: 1000 MB/s

PC800
Rambus DRDRAM
400 MHz DDR 16-bit
Peak bandwidth: 1600 MB/s

Intel PIII GHZ
L1: 16K INST, 16K DATA (3 cycle latency), both 4-way
L2: 256K 8-way 256 bit, Latency: 7 cycles
  L1,L2 on-chip


Intel 840 uses two PC800 channels
X86 CPU Cache/Memory Performance Example:
AMD Athlon T-Bird Vs. Intel PIII

This Linpack data size range (data working set) causes L2 data misses and relies on main memory for data access.

Memory Performance - Linpack

Intel PIII cannot utilize available PC800 bandwidth due to its low 133MHZ system bus speed.

**X86 CPU Cache/Memory Performance Example:**

**AMD Athlon T-Bird Vs. Intel PIII, Vs. P4**

**AMD Athlon T-Bird 1GHZ**
- L1: 64K INST, 64K DATA (3 cycle latency), both 2-way
- L2: 256K 16-way, 64 bit
  - Latency: 7 cycles
  - L1,L2 on-chip

**Intel P4, 1.5 GHZ**
- L1: 8K INST, 8K DATA (2 cycle latency), both 4-way
  - 96KB Execution Trace Cache
- L2: 256K 8-way, 256 bit
  - Latency: 7 cycles
  - L1,L2 on-chip

**Intel PIII 1 GHZ**
- L1: 16K INST, 16K DATA (3 cycle latency), both 4-way
- L2: 256K 8-way, 256 bit
  - Latency: 7 cycles
  - L1,L2 on-chip


Intel P4 utilizes PC800 bandwidth much better than PIII due to P4’s higher 400MHZ system bus speed.
X86 CPU Cache/Memory Performance Example:
AMD Athlon T-Bird Vs. Duron

AMD Athlon T-Bird
750MHZ-1GHZ
L1: 64K INST, 64K DATA,
both 2-way
L2: 256K 16-way 64 bit
Latency: 7 cycles
L1,L2 on-chip

Memory:
PC2100
133MHZ DDR SDRAM 64bit
Peak bandwidth: 2100 MB/s

PC1600
100MHZ DDR SDRAM 64bit
Peak bandwidth: 1600 MB/s

AMD Athlon Duron
750MHZ-1GHZ
L1: 64K INST, 64K DATA
both 2-way
L2: 64K 16-way 64 bit
Latency: 7 cycles
L1,L2 on-chip

Source: http://www1.anandtech.com/showdoc.html?id=1345&p=10
A Typical Memory Hierarchy

Processor
  Control
  Datapath
    Registers
    On-Chip Level One Cache $L_1$
    Second Level Cache (SRAM) $L_2$
  Main Memory (DRAM)
  Virtual Memory, Secondary Storage (Disk)
  Tertiary Storage (Tape)

Faster
Larger Capacity

Speed (ns): 1s 10s 100s 10,000,000s (10s ms) 10,000,000,000s (10s sec)
Size (bytes): 100s Ks Ms Gs Ts
Virtual Memory

- Virtual memory controls two levels of the memory hierarchy:
  - Main memory (DRAM)
  - Mass storage (usually magnetic disks)

- Main memory is divided into blocks allocated to different running processes in the system:
  - Fixed size blocks: Pages (size 4k to 64k bytes).
  - Variable size blocks: Segments (largest size 216 up to 232)

- At a given time, for any running process, a portion of its data/code is loaded in main memory while the rest is available only in mass storage.

- A program code/data block needed for process execution and not present in main memory results in a page fault (address fault) and the block has to be loaded into main memory from disk by the OS handler.

- A program can be run in any location in main memory or disk by using a relocation mechanism controlled by the operating system which maps the address from the virtual address space (logical program address) to physical address space (main memory, disk).
Virtual Memory

Benefits

- Illusion of having more physical main memory
- Allows program relocation
- Protection from illegal memory access

<table>
<thead>
<tr>
<th>Virtual address</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27</td>
</tr>
<tr>
<td>...............</td>
</tr>
<tr>
<td>15 14 13 12</td>
</tr>
<tr>
<td>11 10 9 8</td>
</tr>
<tr>
<td>...............</td>
</tr>
<tr>
<td>3 2 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Virtual page number</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Translation

<table>
<thead>
<tr>
<th>Physical page number</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical address

| 29 28 27 |
| ............... |
| 15 14 13 12 |
| 11 10 9 8 |
| ............... |
| 3 2 1 0 |

<table>
<thead>
<tr>
<th>Physical address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
### Paging Versus Segmentation

<table>
<thead>
<tr>
<th></th>
<th>Page</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words per address</td>
<td>One</td>
<td>Two (segment and offset)</td>
</tr>
<tr>
<td>Programmer visible?</td>
<td>Invisible to application programmer</td>
<td>May be visible to application programmer</td>
</tr>
<tr>
<td>Replacing a block</td>
<td>Trivial (all blocks are the same size)</td>
<td>Hard (must find contiguous, variable-size, unused portion of main memory)</td>
</tr>
<tr>
<td>Memory use inefficiency</td>
<td>Internal fragmentation (unused portion of page)</td>
<td>External fragmentation (unused pieces of main memory)</td>
</tr>
<tr>
<td>Efficient disk traffic</td>
<td>Yes (adjust page size to balance access time and transfer time)</td>
<td>Not always (small segments may transfer just a few bytes)</td>
</tr>
</tbody>
</table>
Virtual → Physical Addresses Translation

Contiguous virtual address space of a program

Physical location of blocks A, B, C
Mapping Virtual Addresses to Physical Addresses Using A Page Table
Virtual Address Translation

Virtual page number

Page table

<table>
<thead>
<tr>
<th>Valid</th>
<th>Physical page or disk address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Physical memory

Disk storage
Two memory accesses needed:
First to page table
Second to item
## Typical Parameter Range For Cache and Virtual Memory

<table>
<thead>
<tr>
<th>Parameter</th>
<th>First-level cache</th>
<th>Virtual memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block (page) size</td>
<td>16–128 bytes</td>
<td>4096–65,536 bytes</td>
</tr>
<tr>
<td>Hit time</td>
<td>1–2 clock cycles</td>
<td>40–100 clock cycles</td>
</tr>
<tr>
<td>Miss penalty</td>
<td>8–100 clock cycles</td>
<td>700,000–6,000,000 clock cycles</td>
</tr>
<tr>
<td></td>
<td>(Access time)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(6–60 clock cycles)</td>
<td>(500,000–4,000,000 clock cycles)</td>
</tr>
<tr>
<td></td>
<td>(Transfer time)</td>
<td>(200,000–2,000,000 clock cycles)</td>
</tr>
<tr>
<td>Miss rate</td>
<td>0.5–10%</td>
<td>0.00001– 0.001%</td>
</tr>
<tr>
<td>Data memory size</td>
<td>0.016–1MB</td>
<td>16–8192 MB</td>
</tr>
</tbody>
</table>
Virtual Memory Issues/Strategies

• **Main memory block placement:** Fully associative placement is used to lower the miss rate.

• **Block replacement:** The least recently used (LRU) block is replaced when a new block is brought into main memory from disk.

• **Write strategy:** Write back is used and only those pages changed in main memory are written to disk (dirty bit scheme is used).

• To locate blocks in main memory a **page table** is utilized. The page table is indexed by the virtual page number and contains the physical address of the block.
  – In paging: Offset is concatenated to this physical page address.
  – In segmentation: Offset is added to the physical segment address.

• To limit the size of the page table to the number of physical pages in main memory a hashing scheme is used.

• Utilizing address locality, a **translation look-aside buffer (TLB)** is usually used to cache recent address translations and prevent a second memory access to read the page table.
Speeding Up Address Translation: Translation Lookaside Buffer (TLB)

- TLB: A small on-chip fully-associative cache used for address translations.
- If a virtual address is found in TLB (a TLB hit), the page table in main memory is not accessed.

<table>
<thead>
<tr>
<th>Virtual Page Number</th>
<th>Physical Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Page or Disk Address</td>
<td>Disk Storage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TLB (on-chip)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128-256 Entries</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical Page Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page Table (in main memory)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
Operation of The Alpha AXP 21064 Data TLB During Address Translation

Virtual address

Valid
Read Permission
Write Permission

TLB = 32 blocks
Data cache = 256 blocks
TLB access is usually pipelined
TLB & Cache Operation

Virtual address

TLB access

TLB hit?

Yes

No

Physical address

Cache operation

Write?

Yes

No

Try to read data from cache

Cache hit?

Yes

No

Cache miss stall

Write?

Yes

No

Write access bit on?

Yes

No

Write protection bit on?

Write data into cache, update the tag, and put the data and the address into the write buffer

Deliver data to the CPU
### Event Combinations of Cache, TLB, Virtual Memory

<table>
<thead>
<tr>
<th>Cache</th>
<th>TLB</th>
<th>Virtual Memory</th>
<th>Possible?</th>
<th>When?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
<td>Possible, no need to check page table</td>
<td></td>
</tr>
<tr>
<td>Hit</td>
<td>Miss</td>
<td>Hit</td>
<td>TLB miss, found in page table</td>
<td></td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
<td>Hit</td>
<td>TLB miss, cache miss</td>
<td></td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
<td>Miss</td>
<td>Page fault</td>
<td></td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
<td>Impossible, cannot be in TLB if not in memory</td>
<td></td>
</tr>
<tr>
<td>Hit</td>
<td>Hit</td>
<td>Miss</td>
<td>Impossible, cannot be in TLB or cache if not in memory</td>
<td></td>
</tr>
<tr>
<td>Hit</td>
<td>Miss</td>
<td>Miss</td>
<td>Impossible, cannot be in cache if not in memory</td>
<td></td>
</tr>
</tbody>
</table>